



# Next Generation Intelligent LCDs

## iLCD Software and Command Documentation

Unless otherwise noted, all materials contained in this document are copyrighted by demmel products and may not be used except as provided in these terms and conditions or in the copyright notice (documents and software) or other proprietary notice provided with the relevant materials.

Copyright © by demmel products gmbh 2004 - 2022

## Table of Contents

Introduction.....	1
Getting Started.....	1
Get used to Commands and Macros.....	17
Product Documentation.....	21
iLCD Manager XE Features.....	21
File.....	22
Graphics.....	24
Fonts.....	26
Macros.....	27
Messages.....	29
Settings.....	31
Device.....	32
Status Bar.....	34
Java IDE.....	35
iLCD Command Set.....	37
Command Overview.....	37
Command Description.....	70
Data Sent by the iLCD Controller.....	291
Error Codes.....	296
Controlling Options.....	299
Sample Source Code.....	301
Java.....	302
General Info.....	303
Package comm.....	314
Package hw.....	389
Package ilcd.....	449
Package java.io.....	691
Package java.lang.....	904
Package java.util.....	1025
Package javax.....	1219
Package javax.events.....	1223
Firmware Version History.....	1246
iLCD Manager XE Version History.....	1252

## Introduction

We fully understand that you probably won't want to read the whole of this documentation before starting to use your iLCD. Nevertheless, we strongly recommend that you read the chapters listed below as these will give you a good foundation about communicating with an iLCD controller. This will help you later on when you come to construct your own command strings and macros.

Starters with iLCD will be interested in principal answers to hardware and software issues, the use of commands and macros and where to find documentation:

- [Hardware Interconnection](#)
- [Software Startup](#)
- [Get used to Commands and Macros](#)
- [Product Documentation](#)

An overview over the iLCD Manager XE IDE functionality is given here:

- [iLCD Manager XE Features](#)

The following chapters provide a rationale of commands:

- [Command Structure](#)
- [Syntax Used in iLCD Manager XE](#)
- [Command Overview](#)
- [The Concept of iLCD's Touch Fields](#) (if you want to use touch fields)

The full command reference manual is provided in the

- [Command Description](#) section.

iLCD Manager XE contains some samples with comments (use the "New" button on the "File" page) allowing you to carry out your first steps. Besides that you can load miscellaneous demo files and have a look at the well commented macros on the "Macros" page to learn more about how to communicate with iLCD.

---

## Getting Started

When you first power up the iLCD a pre-installed Demo Project is started. Check it out to get information on the various features of the iLCD. You can always reload the demo later via the **File** tab under **New**.

## What can you do with an iLCD?

The iLCD can be controlled with a host controller via USB, RS232, I<sup>2</sup>C, SPI and Ethernet or used as stand alone device.

- Display Graphics
- Draw Dots, Lines, Rectangles, Circles, Ellipses, etc.
- Use Animations
- Display Text using Unicode Fonts with Anti-Aliasing
- React on touch events (e.g. a button pressed)

- Use multiple screens and viewports
  - Use I/Os for relays, speaker, matrix keyboard and LEDs
- 

## Install iLCD Manager XE

The iLCD Manager XE software is used to configure and program the iLCD controller.

The latest version is available at <http://www.demmel.com>

---

## Establish connection between iLCD Manager XE and iLCD panel

### Power supply

For supplying power, different possibilities are available:

- Power supply via the evaluation board with the AC/DC adapter (please refer to the "Power Supply Connector (Power)" chapter of the *Technical Specification* document also known as [iLCD Panel Specification.pdf](#)).
- Power supply via the evaluation board with USB.
- Power supply on the iLCD panel.

Please read the annotations in the *Technical Specification* document also known as [iLCD Panel Specification.pdf](#), chapter "Power Supply Configuration Connector (Vsel)". There is also a local *iLCD Panel Specification.pdf* version available, and can be found under the iLCD Manager XE installation path within the *Documentation/Specification* directory.

### Communication Interface

Connect the evaluation board (Control connector) with the display via the enclosed 20/24-pin FFC cable with the contact surface upwards (top-contact FFC connector).

Connect the evaluation board with the PC via USB.

Install and start iLCD Manager XE Software. On the **File** tab under **New** press the Auto Set Button.

---

Automatically detect connected iLCD panel

Auto-Set

---

**See also:**

[Controlling Options](#)

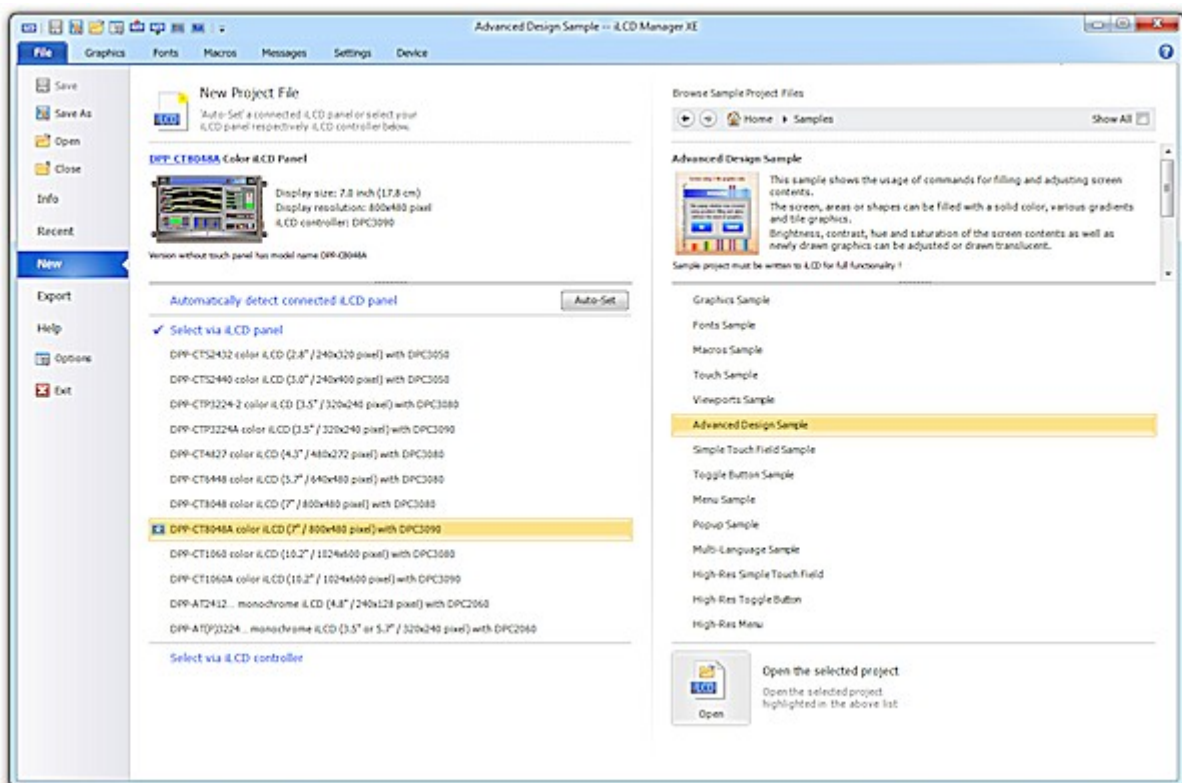
## Software Startup

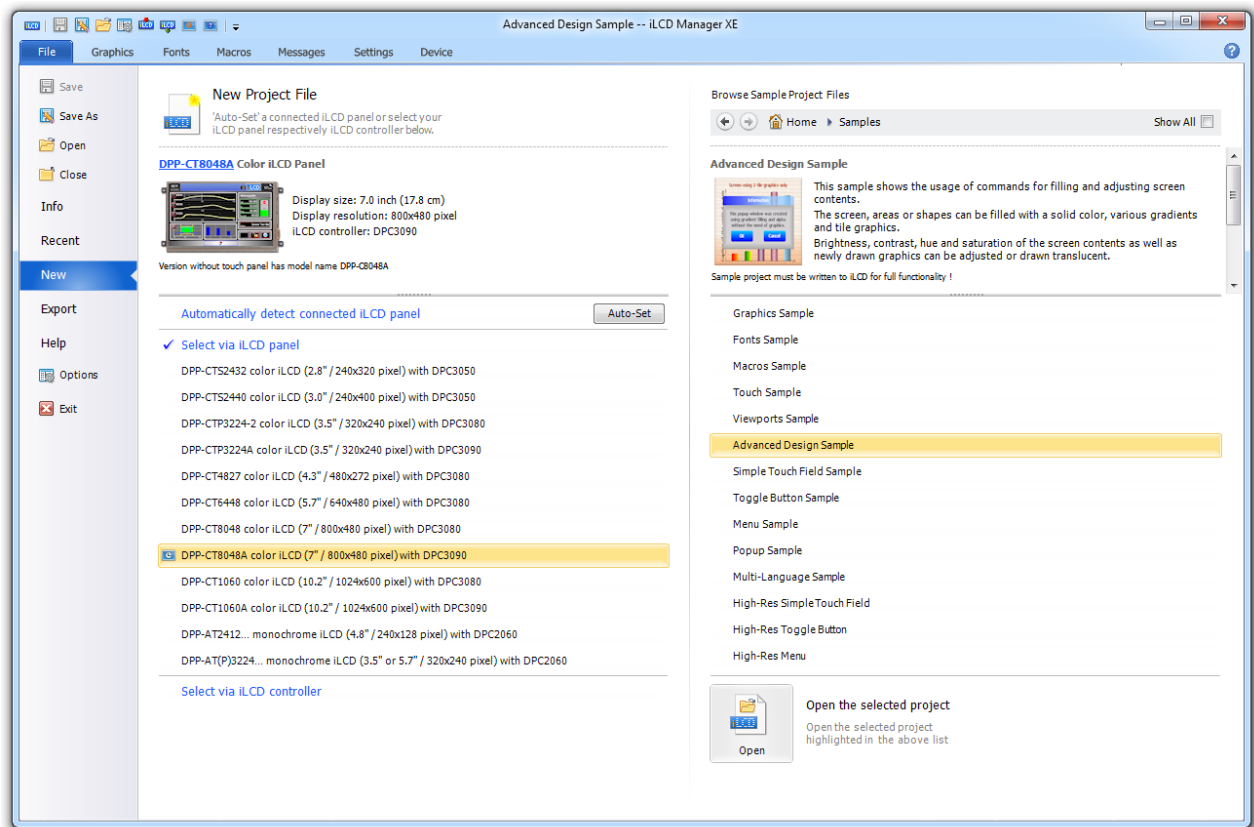
The Evaluation Kit is delivered with a demo project installed in the iLCD's flash memory. A number of demo and sample projects are delivered with the iLCD Manager XE which can be used in any iLCD development environment. They are available under File → New.

In order to select one, set the desired iLCD Panel (detect automatically or choose by panel type or by controller) – the appropriate demo/sample projects are displayed for selection. Then browse for demo/sample projects and mind the information on the project. Picture 1 shows the look-and-feel of the window.

Please note that demos and samples for smaller display panels also run on displays with higher resolution - mind the applicability statement in the demo/sample info. There is also the option to

**Show All**  demos/samples.



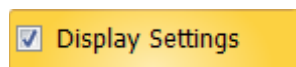


**Picture 1: Selection of demos and samples.**

To get a new or changed project working on the connected iLCD panel , it is to be written into the panel 's flash memory. This can be achieved in 3 ways:

- The "Write" button in the window's quick access toolbar
- The **[Ctrl+W]** shortcut (working in any window state)
- Clicking on the "Write" button in the "Device" ribbon

Perhaps the display orientation appears not as desired. In the "Settings" panel, click on "Display Settings" and check the "Display Settings" checkbox.



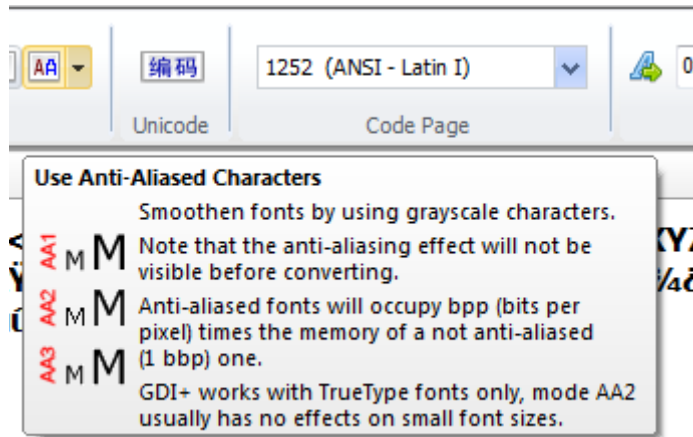
Select the display orientation. Changed settings take effect after writing the project into the flash memory.

After the installation of the iLCD Manager XE, the baud rate for the serial port currently in use is set to the value of 115200 Baud. Setting baud rates deviating from 115200 Baud can be done by means of the "Set Baud Rate" command. This setting is valid until the next power up or reboot of the iLCD panel. In order to change the baud rate permanently, it has to be set in the iLCD Manager XE on the "Settings" page, too. After downloading this new setup data via the USB port to the iLCD panel, the baud rate of the setup software is changed automatically according to the new setting, a message box appears.

## Reading Guide

### Getting Help

To get help on the various iLCD Manager XE controls (Buttons, etc.) move the mouse cursor over the controls and read the Tooltips.



To get help on iLCD Commands position the cursor on the command line and press **[F1]**.

### Hints

- If you don't want to enter all the commands in the examples simply copy the listed code to the *Commands to iLCD* panel on the **Device** tab.

---

## Basic Features & Concepts

- [Command Syntax](#)
- [Attributes](#)
- [Graphics](#)
- [Fonts](#)
- [Touch Fields](#)
- [Viewports](#)
- [Macros](#)
- [Input/Output](#)

### Command Syntax

Every Command in iLCD Manager XE starts with `\i` (command introducer - 0xAA).

```
[ \i ] [Command Code] ([Parameters:Values, Strings]) ( {Comment} )
```

[Command Code]

1-3 Bytes (ASCII-Code of the cmd characters)

[Parameters]

n-Bytes of Data (Note that 0xAA in the data has to be quoted (0xAA 0xAA), see [Syntax used in iLCD Manager XE](#)).

## Example

```
\iG\mName of Graphic\0 {Display Local Graphic}
```

Note: Not well formed commands will be marked with a wavy line.

```
\iAVD\0lkj
```

For possible parameter types and the exact syntax see the [Command Structure](#) chapter in the help.

## Attributes

In most cases the attributes for certain commands are set using separate commands. For example, when defining a Touch Field with the Command *Create/Define Touch Field* the x/y-dimensions are set with the additional commands *Set Touch Field Height* and *Set Touch Field Width*.

The whole command sequence is:

```
\iTH\D200 {Set Touch Field Height}  
\iTW\D200 {Set Touch Field Width}  
\iTA\0k {Create/Define Touch Field}
```

Normally the set attribute values are valid for all subsequent commands. Exceptions are the alignment commands *Set Graphic Alignment* and *Set Text Alignment*, they are only valid for the following graphic or text command.

Frequently used attribute commands are:

- *Set Cursor Position*
- *Set Filling Color*
- *Set Graphic Alignment*
- *Set Text Alignment*
- *Set Foreground (Background)*
- *Set Touch Field Width (Height)*
- *Set Touch Field Make (Break) Macro*
- *Set Font*
- *Set Bold Mode*
- *Set Underline Mode*

## Graphics

You can load static and animated graphics to the Flash memory or the micro SD Card of the iLCD and scale them.

The following formats are supported:

- Bitmap (\*.bmp)
- GIF (\*.gif)
- JPEG (\*.jpg, \*.jpeg)
- PNG (\*.png)
- Raw iLCD Graphics Image (\*.rii)

You can also draw graphical objects like Dots, Lines, Circles, Ellipses, etc. and apply colors and filling patterns.



## **Fonts**

You can load a selection of characters using any Windows font to the Flash memory or the micro SD Card of the iLCD. Both ASCII and Unicode is supported. Text can be aligned, wrapped and scaled.

## **Touch Fields**

Up to 64 rectangular Touch Fields can be defined. The Touch Fields can be associated with macros which are executed when the Touch Field is pressed (Make Macro) or released (Break Macro).

## **Screens and Viewports**

There is a *Main Screen* and 8 additional screens (0-7). This allows to draw in the background on a *Draw Screen* that is not visible. The visible screen is referred to as *View Screen* (See [Screen Memory Related Commands](#)).

In addition to the screens, screen areas can be defined as viewports with their own coordinates and orientation. In addition to viewport 0 (whole screen), there can be 8 viewports (1-8) in total for all screens (See [Viewport Related Commands](#)). A viewport can be defined on the currently active Draw Screen with the command \iCVD ([Define Viewport](#)).

## **Macros**

Command sequences can be collected in macros and e.g. called by the host controller. Touch Events can be handled by associating Make (Press) and Break (Release) macros to Touch Fields. Macros can call other macros up to a nesting depth of 8.

## **Input/Output**

Most ports of the 32 bit iLCD Controller can be assigned as digital or analog inputs or outputs.

---

## **Step by Step Guide**

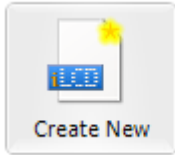
The following examples will guide you through the most important steps in creating simple iLCD projects. If you are not interested in creating your own projects right now you can go to the section [Sample Projects](#) to check out ready to go samples included in the iLCD Manager XE Software.

---

### **Creating a first simple project**

The following example demonstrates how to display a graphic.

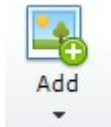
1. On the **File** tab under **Create New** create an empty project.



## Create a new empty project


The selection on the left panel will be used for the new project

2. On the **Graphics** tab add a graphic.



Some sample graphics (e.g. lemons.bmp) can be found in the iLCD Manager XE install folder (\demmel products\iLCD Manager XE\Data\Graphics). The following formats are supported:

- Bitmap (\*.bmp)
- GIF (\*.gif)
- JPEG (\*.jpg, \*.jpeg)
- PNG (\*.png)
- Raw iLCD Graphics Image (\*.rii)

In the **Graphics Items** panel a list of the loaded graphics is shown. Notice that each graphic is associated with a unique index. The graphic can now be written to the iLCD Flash memory by pressing the Write Button  in the Quick Access Toolbar.

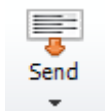
Note: You can also use the shortcut **[Ctrl + W]** or the Write-Button on the **Device** tab to write the project data to the iLCD memory.

Note: You can also use animated graphics. Try dp.gif and press the Animate Button!

3. On the **Device** tab enter the following commands into the *Commands to iLCD* panel:

```
\i!{Reset All}  
\iG\D0 {Display Local Graphic}
```

The command sequence can be executed with **[Ctrl + Enter]** or with the Send Button.



Note: To execute only the actual line use **[Ctrl + Shift + Enter]**.

Note: The comments in {} are not needed.

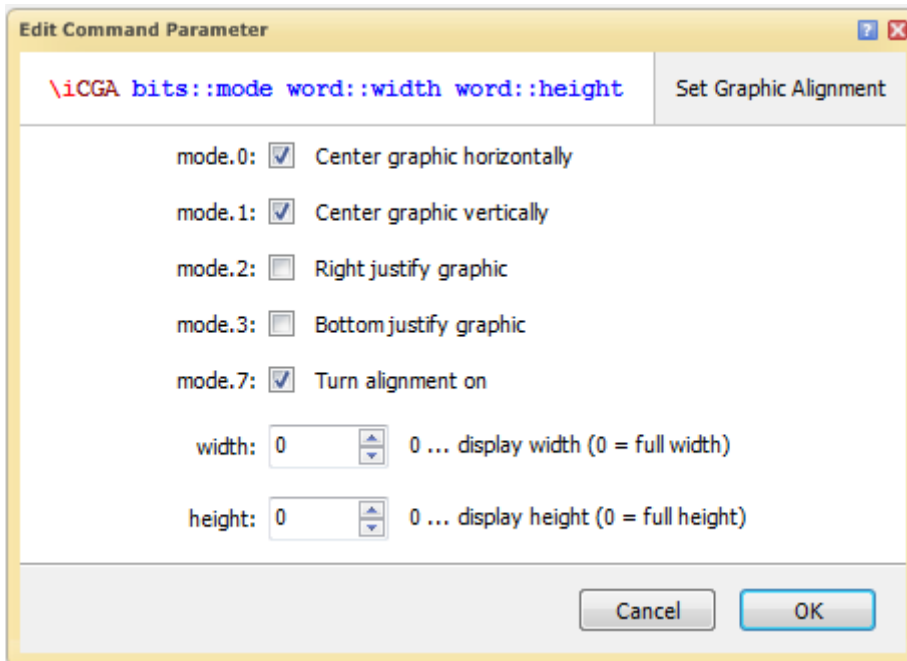
Note: Use the interactive command list in the **Commands** toolbar to browse available commands by categories ( Click  ).

Note: Position cursor on a command and press **[F1]** to get detailed help information.

Note: To view the byte sequence sent to the display controller click the button .

Note: Notice the response from the iLCD panel in the **Response from iLCD** area below.

4. To center the graphic use the *Set Graphic Alignment* command (to be found in the LCD Control category) before the above *Display Local Graphic* command.



```
\iCGA\x83\D0\D0 {Set Graphic Alignment}
```

Note: Turn alignment on has to be checked.

Note: The alignment attributes are only valid for the next subsequent *Display Local Graphic* Command.

Also see the Graphic Sample in [Sample Projects](#).

---

## **Button Example using the Touch Screen**

The following example demonstrates how to use the Touch Screen. It will create a button that can be pressed. Therefore the following steps are necessary:

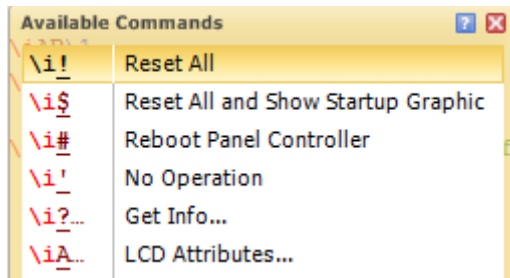
1. Draw a rectangle
2. Define a touch field

1. Go to the **Device** tab and enter the following commands to draw a rectangle:

Click  or use [**Ctrl + Space**] and select the following commands:

- *Reset All*
- *Draw... : Draw Rectangle* (set width and height to 200)

Note: Enter the underlined characters to quickly access the categories. E.g. A to open the *LCD Attributes...* category.



```
\i! {Reset All}
\iDR\x0\D200\D200 {Draw Rectangle}
```

Note: You can also draw other graphical objects like Dots, Lines, Circles, Ellipses, etc. See the section *Draw* in the interactive command list and the help for more details.

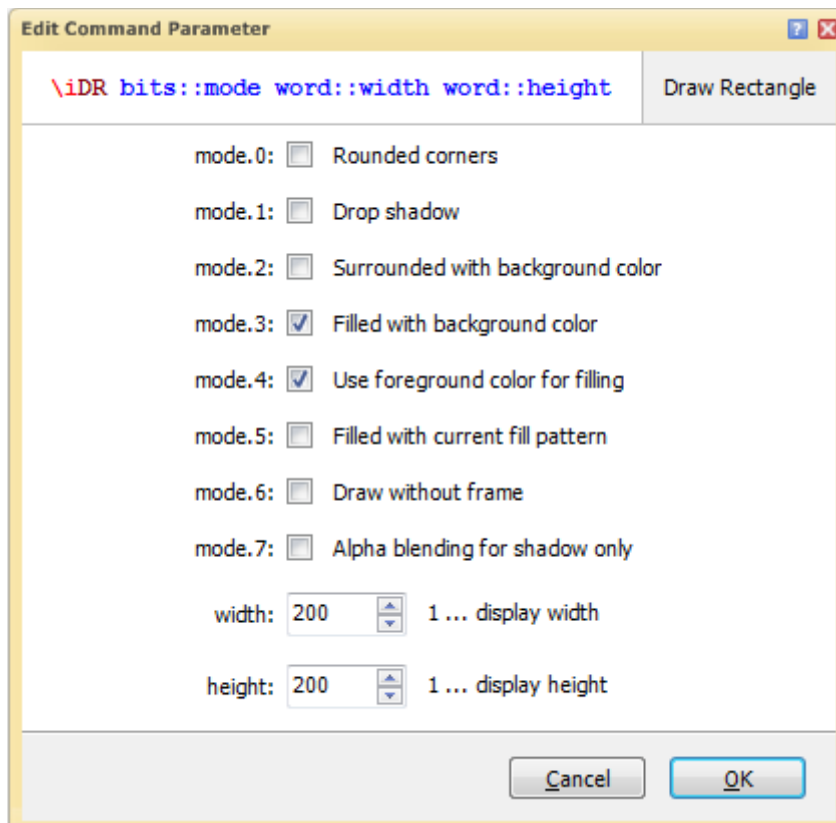
2. Send command sequence to the iLCD with **[Ctrl + Enter]**.

3. Position the rectangle with command *LCD Control: Set Cursor Position* (this command has to be sent **before** the above *Draw Rectangle* command in order to take effect, see [Basic Concepts](#), Attributes).

```
\iCK\D100\D100 {Set Cursor Position}
```

Note: Positioning of Graphics, Text, Touch Fields is always done

4. The rectangle can be filled by e.g. setting the foreground color with the command *LCD Attributes: Set Foreground Color* (Press **[F1]** to see what color values are possible.) and then setting the below parameters for the command *Draw Rectangle*.



```
\iACF\#00CCFF {Set Foreground Color}  
\iDR\x18\D200\D200 {Draw Rectangle}
```

Note: *Filled with background color* has to be checked in order to enable the fill mode.

Note: If you go to the More Colors Dialog Window and select Pick Color you can choose whatever color you find on your screen.

An alternative way to fill the rectangle is to use option Nr. 5 *Filled with current fill pattern*. In order to use this option you have to set a filling color before the *Draw Rectangle* command. This is done with the *Set Filling Color* command:

```
\iAPC\#00CCFF {Set Filling Color}
```

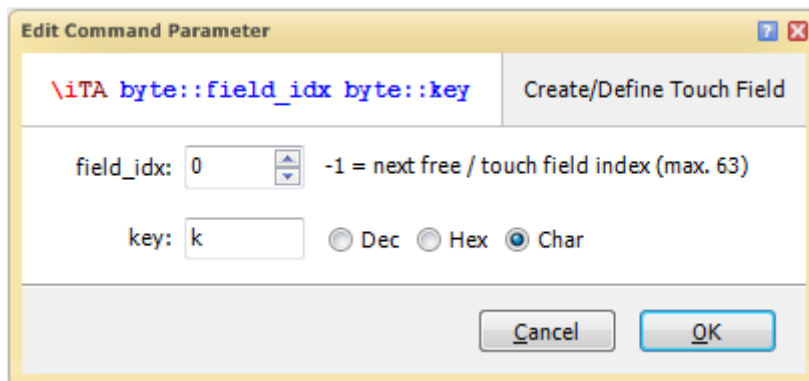
Note: In contrast to the *Set Graphic Alignment* command, *Set Filling Color* is valid vor all subsequent commands. This is done to minimize data traffic. The strategy to set command attributes for subsequent commands separatly is used with many iLCD commands. It is also descibed in the section [Basic Concepts](#).

If you want to add some eye candy to the fill pattern, try the *Set Filling Gradient* Command.

```
\iAPG\4\D0\#00FFFF\#0000FF {Set Filling Gradient}
```

See the help ([F1]) for details on this command.

5. Define a Touch Field with the command *Create/Define Touch Field*.



Note: You can identify the Touch Field by its user defined key (k in this case).

The Attributes *Height* and *Width* of the Touch Field are set with the commands *Set Touch Field Height* and *Set Touch Field Width*. Notice that the Cursor Position is still at 100/100 from drawing the rectangle.

```
\iTH\D200 {Set Touch Field Height}  
\iTW\D200 {Set Touch Field Width}  
\iTA\0k {Create/Define Touch Field}
```

When you press the button and hold it you'll see the following sequence in the *Response from iLCD* panel:

#### Response from iLCD

Kk [ACK]

The upper case "K" indicates a touch screen pressed event (make). When you release the button a touch screen release event (break) will be reported:

#### Response from iLCD

Kk [ACK] kk [ACK]

The second character ("k") is the key defined in the *Create/Define Touch Field* command. You can use this key to recognize that the button was pressed.

Note: Assigning a key value of 0 will inhibit the reporting of the make/break events via the interface to the host controller. Associated make/break macros however will still be executed. This is useful when communication with the host controller is not necessary.

The following events are possible:

Event	Description
K	touch has been pressed (make)
k	touch has been released (break)
M	location of the press has been moved
NULL	no event has occurred

Note: Up to 64 touch fields can be defined.

Also see the Touch Sample in [Sample Projects](#).

## Using Fonts

To show text on the iLCD a large number of Unicode Fonts can be used. The following example demonstrates how to use add Fonts to a project and how to use them.

1. Go to the **Fonts** tab and press the Add-Button:



Now you can choose from various fonts and select the characters you want to use in your project. The Font selections are listed in the *Fonts Items* panel and addressed (like graphics) by their index.

Note: You can check out different Anti-Aliasing Algorithms.

Fonts are written to the iLCD together with graphics, macros and messages with **[Ctrl + W]**.

```
\i! {Reset All}  
\iCK\D100\D100 {Set Cursor Position}  
\iAF\D1 {Set Font}  
\iDTThis is a Text using Font #1!\0 {Write Text}
```

Note: The cursor position moves to the end of the text.

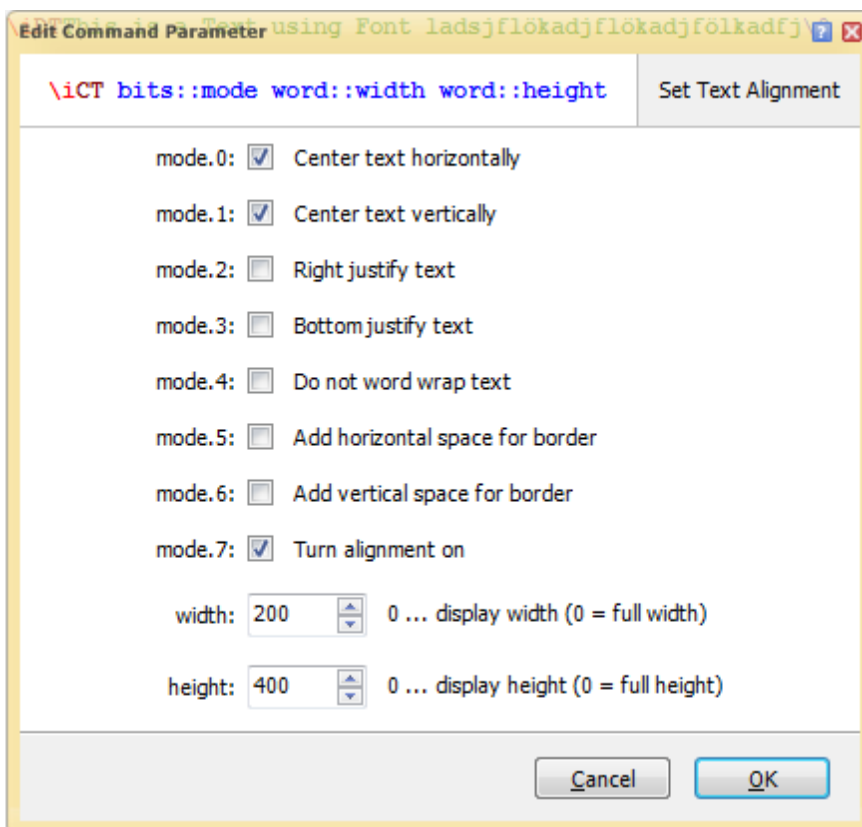
Note: Strings have to be terminated with \0.

Note: If you want to write Unicode Text (e.g. chinese symbols) use the command *Write Unicode Text*.

2. Set Text Format with the commands *Set Underline Mode* and *Set Bold Mode* in the *LCD Attributes* category. Set Text Color with the command *Set Foreground Color*.

```
\iAU\1 {Set Underline Mode}  
\iAB\1 {Set Bold Mode}  
\iACF\#FF0000 {Set Foreground Color}
```

3. Set Alignment of the Text before *Write Text*.



```
\iCT\x83\D100\D100 {Set Text Alignment}
```

Note: The text wrapping is done word wise.

4. Place Text inside a rectangle.

Set width and height of Set Text Alignment to the dimensions of the rectangle. Draw the rectangle before the *Write Text* command otherwise the rectangle will be positioned at the end of the text. The whole sequence looks like this:

```

\i! {Reset All}
\iCK\D100\D100 {Set Cursor Position}
\iDR\x1\D200\D200 {Draw Rectangle} \iAF\D1 {Set Font}
\iCT\x83\D200\D200 {Set Text Alignment}
\iDTThis is a Text using Font #1!\0 {Write Text}

```

Also see the Font Sample in [Sample Projects](#).

## Macro Example

Command sequences can be executed using macro definitions. Macros can call other macros up to a nesting depth of 8. To react on Touch Events, Touch Fields can be associated with Make and Break Macros.

The following example toggles the color of a rectangle when it is pressed.

1. Go to the **Macro** tab and press the Add-Button:



Now enter a name for the macro in the Properties Group of the Ribbon. (macros can be called by name or by index): "Create Touch Field". Next enter code for creating the button:

*Create Touch Field:*

```

\iTW\D200 \iTH\D200 {Set Touch Field Width & Height}
\iTA\0\x1 {Create/Define Touch Field, Key: 1}
\iDR\x0\D200\D200 {Draw Rectangle}

```

Note: With **[Ctrl + Enter]** you can execute individual macros, just put the cursor inside the macro.

Note: The 3rd parameter in #macro\_start(0, "Create Touch Field", 1252), i.e. 1252, refers to the code page used for interpreting string characters inside the macro. If you see a blue wavy line below a string, some of the characters in the string cannot be interpreted.

```

191 #macro_start(14, "", 1252)
192 \iDT\κ\λ\υ\ξ\ο\π\ρ\σ\τ\υ\φ\χ\ψ\ω\έ\ή\ά\ο\ύ\β\0
193 #macro_end

```

The greek letters are not defined in code page 1252 !

2. Add another macro and name it "Make Macro". This is the macro called when rectangle is pressed. The following code will draw a red rectangle:

*Make Macro:*

```

\iAPC\#FF0000 {Set Filling Color: Red}
\iDR\x20\D200\D200 {Draw Rectangle}

```

3. Add another macro and name it "Break Macro". This is the macro called when rectangle is released.



*Break Macro:*

```
\iAPC\#0000FF {Set Filling Color: Blue}  
\iDR\x20\D200\D200 {Draw Rectangle}
```

4. Add another macro and name it "Start". Now the Make- and Break Macros have to be associated with the Touch Field. This is done with the commands *Set Touch Field Make Macro* and *Set Touch Field Break Macro*. Notice that the make & break macro association is valid for all subsequent *Create/Define Touch Field* commands (see the Attributes Concept in [Basic Concepts](#)).

*Start:*

```
{Define Make & Break Macros}  
\iTM\mMake Macro\0 \iTB\mBreak Macro\0
```

Notice: You can also associate break & make macros by their index (-1 removes the macro association).

Now add call the first macro with command *Execute Macro*.

```
\iOE\mCreate Touch Field\0 {Execute Macro}
```

5. Write Macros to the iLCD (**[Ctrl + W]**)

6. Got to the **Device** tab and execute the start macro.

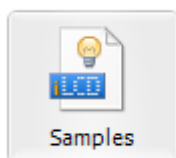
Note: A macro call via the communication port stops the currently running macro and immediately starts the called macro.

Also see the Macro Sample in [Sample Projects](#).

---

## Sample Projects

On the **File** tab under **New** small sample projects on various topics can be opened.



### Browse for sample projects

Sample projects can help you to learn how to operate your iLCD panel

Explore the samples following these steps:

1. Load project
2. Write current project to the iLCD panel **[Ctrl + W]**
3. Goto Macros Tab and execute individual Macros with **[Ctrl + Enter]**

Note: If you want to execute only a part of the command sequence in a macro, highlight the commands you want to execute and press **[Ctrl + Enter]**.

## Browse Sample Project Files



Home ▶ Samples

Show All

### Graphics and Animations Sample



This sample shows the usage of commands for displaying graphics and animations from the iLCD's flash memory.

Graphics and Animations can be addressed by an index or a custom name.

Animations can be started, stopped and limited in their repetitions.

Sample project must be written to iLCD for full functionality !

Graphics Sample

Fonts Sample

Macros Sample

Touch Sample

Viewports Sample

Advanced Design Sample

Simple Touch Field Sample

Toggle Button Sample

Menu Sample

Popup Sample

Multi-Language Sample

QWERTY Sample (800x480)

High-Res Simple Touch Field

High-Res Toggle Button

High-Res Menu



Open

#### Open the selected project

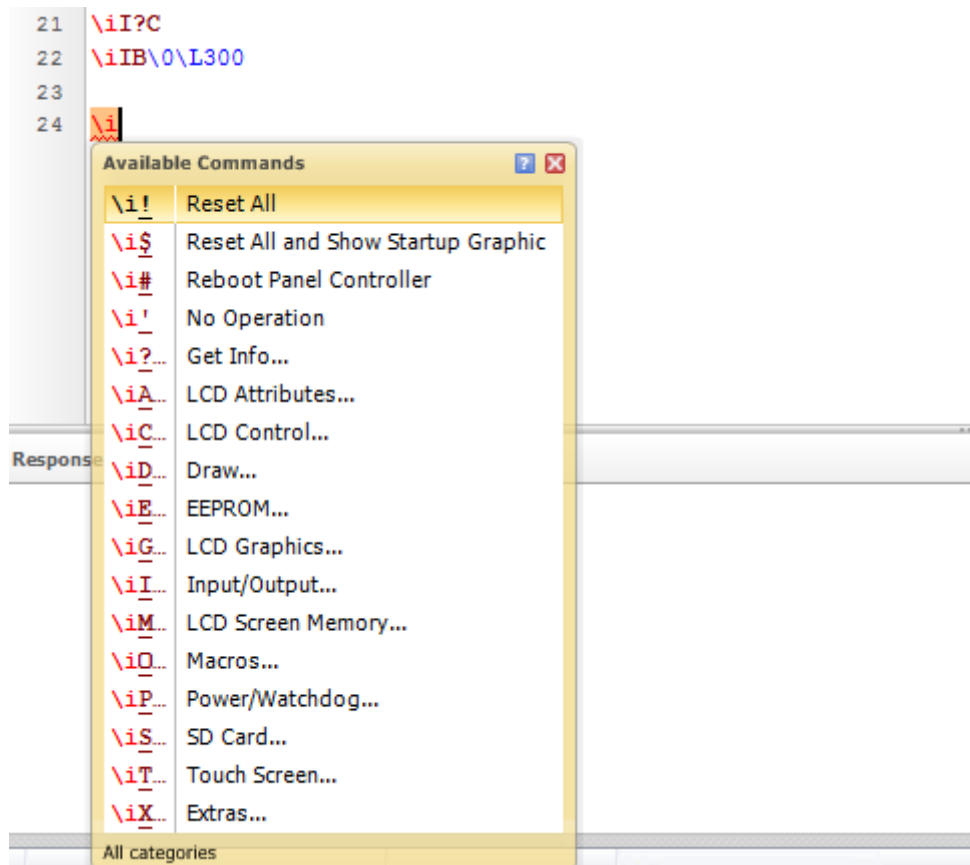
Open the selected project highlighted in the above list

## Get used to Commands and Macros

Macros are a very powerful feature of the iLCD panel controllers and allow the compression of multiple commands into one macro which can be executed at any time via a command.

There are well commented macros within the sample and demo projects. They allow to learn quickly how to communicate with iLCD by means of macros and commands.

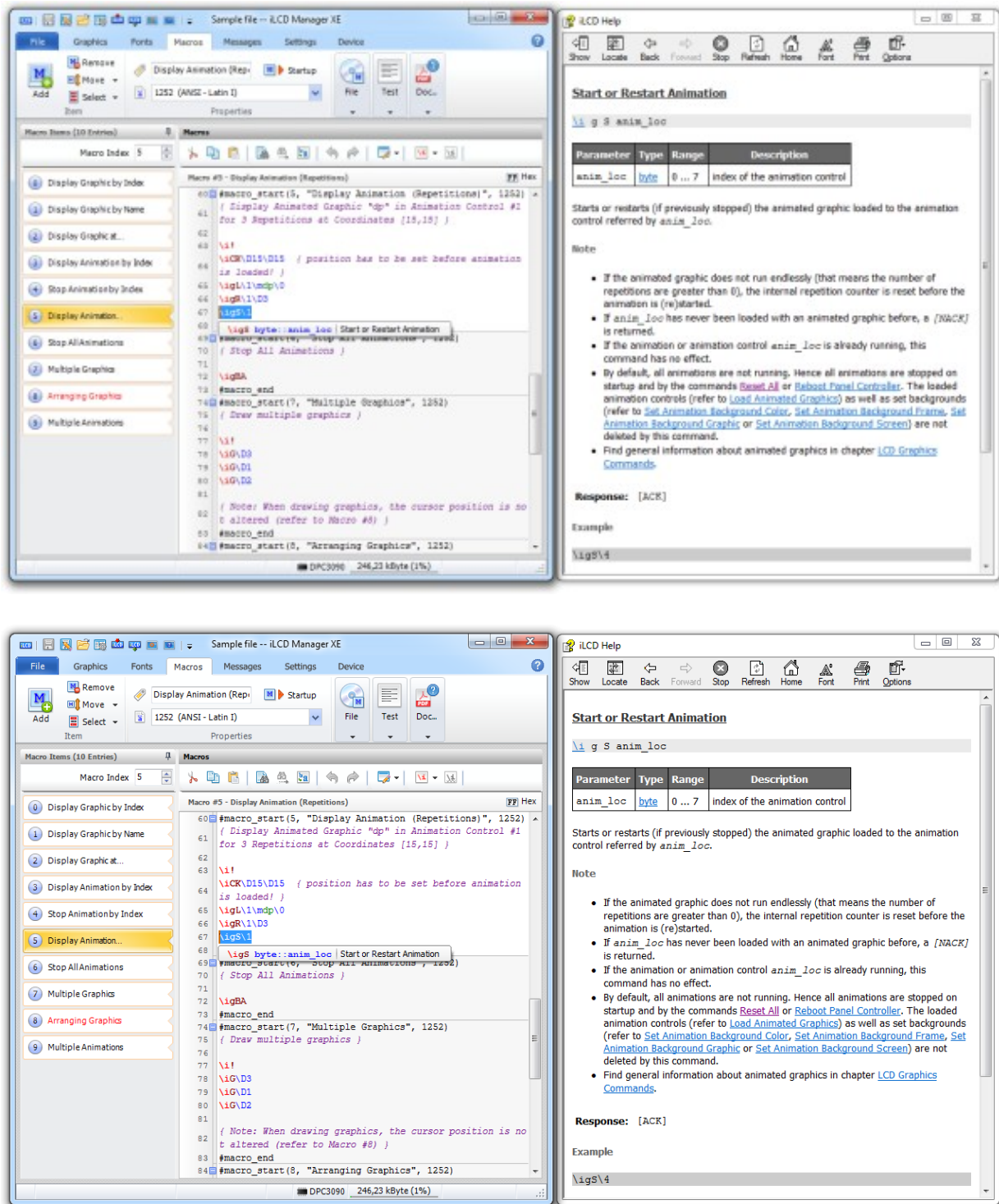
All commands start with a special command introducer (`\i`), followed by a command character and the data block. Picture 1 shows the principally available command groups.




**Picture 1: Overview over the available command groups shown by the Auto Parameter Completion.**

On the "Macros" page, the "Macro Items" are listed on the left side, sorted according to their index and displaying their names. A click on a macro in this panel causes the cursor in the "Macros" panel jump to the start of the respective macro. In the "Macros" panel, the mouse arrow turns into the caret indicating that text editing is possible.


Moving the mouse over a command causes a tooltip to occur on the one hand. The tooltip consists of a concise command description. On the other hand, the Auto Help function opens a window that shows comprehensive information on the command. If you are working with a single computer monitor, please note: in order to have a view on the help window, the iLCD Manager XE window should not be maximized. Otherwise, the help window will be hidden – refer to Picture 2.

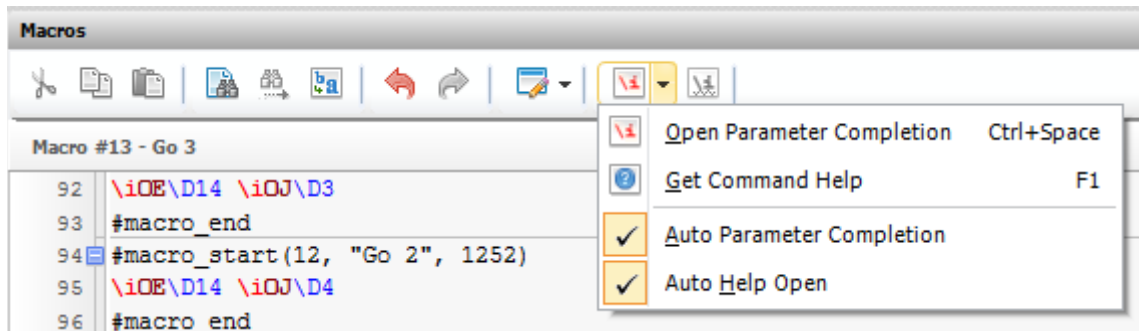


Picture 2: The "iLCD Manager XE" and the "iLCD Help" windows on a screen.

When typing a command, after a certain delay the Parameter Completion feature is activated which guides you through command writing. Once the feature is closed by the user, it will not occur again. You can force it to appear with the shortcut **[Ctrl+Space]** or a click on the  icon.

Editing and exploring commands by means of the Parameter Completion feature makes it convenient to discover the possibilities of the command set.

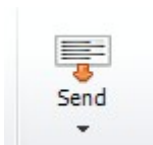
Both, the Auto Help and the Parameter Completion feature can be disabled and enabled again in the drop-down menu of the  icon (Picture 3).



**Picture 3: The Command Help Drop-down Menu for turning off/on Command Help.**

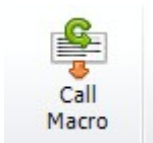
Edited or newly inserted commands can be tested separately using the **[Shift+Ctrl+Enter]** shortcut to send only the command at the caret position or the **[Ctrl+Enter]** shortcut to send the command sequence to the iLCD panel.

Please note: there is a difference in carrying out commands on the iLCD panel:



The "Send" button with its 2 options makes the iLCD Manager XE application sending the commands to the iLCD panel.

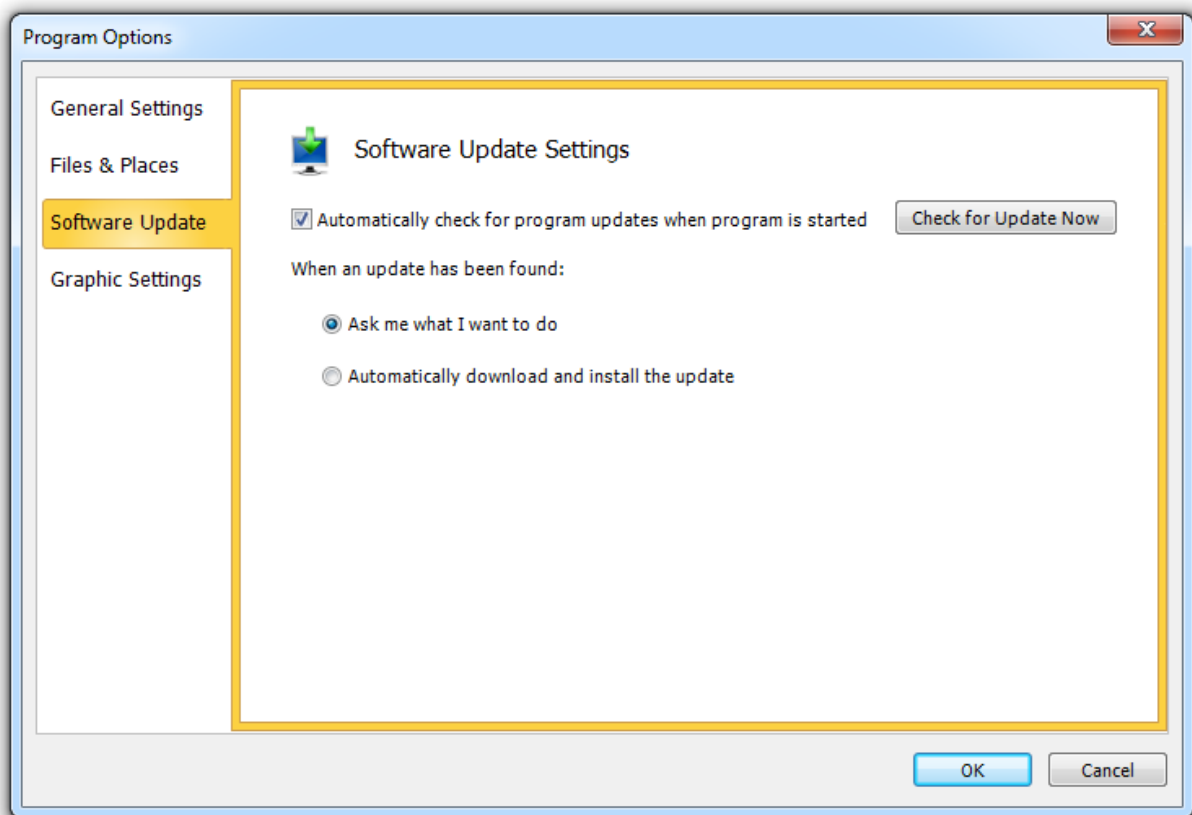
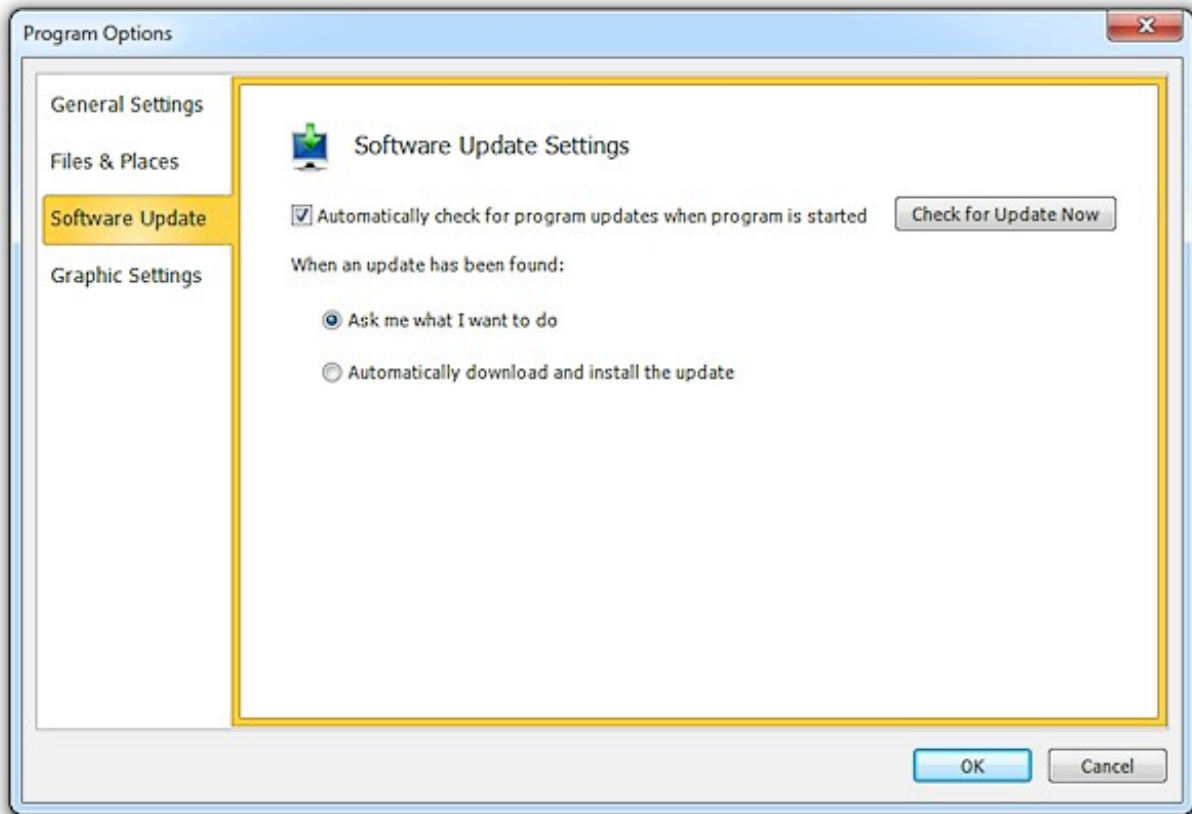
The 2 options have the shortcuts **[Ctrl+Enter]** and **[Shift+Ctrl+Enter]**.



The "Call Macro" button invokes a macro (= command sequence) on the iLCD panel itself.

The corresponding shortcut is **[Shift+Enter]**.

The iLCD Manager XE automatically checks for updates when started. This feature is activated by default and may be changed under File → Options. The Program Options dialog (see Picture 4) can also be invoked from the info (File → Info), where the current project and iLCD Manager XE information can be viewed.




Picture 4: The Options for iLCD Manager XE Update.

## Product Documentation

The iLCD Manager XE is the setup, configuration, programming and management IDE for demmel products' intelligent displays.

A comprehensive documentation is available directly from the iLCD Manager XE window. A click onto the PDF-symbol on the right edge of the "Macros" or the "Device" ribbon or the shortcut **[Shift+Ctrl+F1]** opens the iLCD PDF-document. It contains a short introduction, a function description of the iLCD Manager XE and the detailed command description.

**[F1]** opens the help on commands, if clicked within the active Macros or Device page of iLCD Manager XE.

**[Ctrl+F1]** or a click onto the  question mark in the upper right corner of the IDE opens the contextual help on the active ribbon of iLCD Manager XE.

Furthermore, all available documentation can be found under File → Help and on our website <http://www.demmel.com>.

## iLCD Manager XE Features

This section lists the most important help texts of the iLCD Manager XE IDE. This provides an overview over the whole functionality of this Integrated Development Environment.

Find the functional description of the following pages in this chapter:

- [File](#)
- [Graphics](#)
- [Fonts](#)
- [Macros](#)
- [Messages](#)
- [Settings](#)
- [Device](#)
- [Status Bar](#)
- [Java](#)

## File

### Info

#### **Info**

On the left panel side, general information about the iLCD panel and the memory sizes are displayed. The right panel side gives information about the iLCD Manager XE software version and about previous updates. Further, there is the possibility to change the update settings.

### Recent

#### **Recent**

The projects and storage places are listed for selection. Per default, the 10 most recent projects and places are listed. This value is changeable in the "Files and Places" tab of the Options dialog window.

### New

#### **New**

A number of demo and sample projects are delivered with the iLCD Manager XE. In order to select one, first set the connected iLCD Panel (detect automatically or choose by panel type or by controller). Then browse for demo/sample projects and choose one by double-clicking.

### Export

#### **Export**

The storage options are offered:

- one-by-one raw image for writing into flash or
- raw image for writing onto SD card.


A selection of DPC controllers is provided, the most recent connected preset and an auto-set option.

### Help

#### **Help**

This option contains a page where comprehensive help appropriate to the desired topic is available. Whether it is in the form of a webhelp or a PDF-File: you are given support on software and hardware at every stage of development from the first steps till the end of your project.



 Options

### **Options**

Click onto Options in order to get displayed the "Program Options" dialog that offers the following possibilities:

- General Settings: Show All Warnings, Hints and Messages Again and Reset All Window Positions and Sizes to Default
- Files and Places Settings: Adjust the displayed number of recently used files and places.
- Software Update Settings: Automatic or manual check for software updates and how to proceed with an update.
- Edit Preferences Used for Graphics: Settings for new and for temporary graphic files.
- Simulator Settings: Settings when the simulator is used instead of an iLCD panel.
- Java Settings: Adjust settings for java source files.

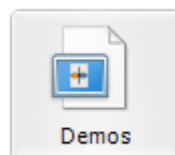


### **Create a new empty project**

The selection on the left panel will be used for the new project

### **Create a new empty project**

The selection on the left panel will be used for the new project.

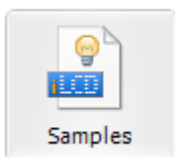


### **Browse for demo projects**

Browse demo projects for currently selected or for all iLCD panels

### **Browse for demo projects**

Browse demo projects for currently selected or for all iLCD panels.



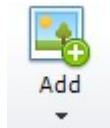
### **Browse for sample projects**

Sample projects can help you to learn how to operate your iLCD panel

### **Browse for sample projects**

Sample projects can help you to learn how to operate your iLCD panel.

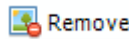
## Graphics



### **Add New Graphic Item [Ctrl+A]**

Add new graphic item by either

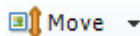
- loading an image from a file
- pasting from the clipboard or
- reading back from the iLCD device.



### **Remove Graphic Item**

Remove graphic item from the list.

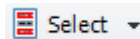
Please note, that removing the item can't be undone!



### **Move Graphic Item**

Move graphic item up or down in the list by changing the actual index in the list.

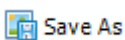
Note that moving an item can be done by dragging it with the mouse as well.



### **Select Graphic Items**

Select multiple graphic items enabling you to perform operations like Remove or Reload (but not Move) on multiple items with a single mouse click. Selected items are marked with a red item index.

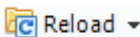
Note that you can select a range by holding down the shift button while clicking on the item.



### **Save Graphic Item**

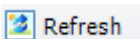
Save graphic item to disk or SD card.

Select the RII image format in the file dialog window to enable the iLCD device to display the image directly from SD card.



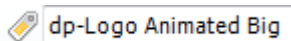
### **Reload Graphic Item**

Reload graphic item from the disk. A different file can be selected and the selection frame can be modified as well.



### **Refresh Graphic Item**

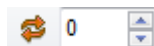
Refresh graphic item by reloading it from the disk and keeping the original selection frame. Note, that refreshing a graphic item can be done only when the image file is available at the original location.



### **Graphic Item Name [Ctrl+N]**

Enter the name to be used for the actual graphic item. Names can be used instead of index numbers to address the item.

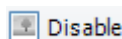
Note that the first item name found will be used so names should be unique.



### **Repetition Count**

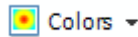
Set repetition count of an animated image.

Note, that a value of 0 sets the repetitions to endless.



### **Disable Graphic**

Disable a graphic to not occupy any space in the flash memory. The graphic item index does still exist, but when trying to attempt to display the image, an error will be returned by the iLCD.



### **Select Number of Colors to Use**

iLCDs can operate with monochrome and color images. Color images can have 256 (8 bit) or 64k (16 bit) colors.

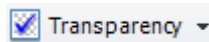
Note that using 8 bit colors reduces the size used in the flash memory by nearly 50 percent, an appropriate palette is automatically created by selecting the most-used 256 colors out of the original picture.



### **Dithering: Select Color Reduction Method**

Dithering can be used to make a color-reduced picture looking more pleasant especially when the original picture has many different colors.

Please try the different methods to see which one fits your needs best. Note, that the best matching method depends on the picture.



### **Pick Transparency Color or Remove Transparency**

Either remove an existing transparency or pick the transparency color to be used by clicking on the appropriate position in the graphic preview.

Note that on monochrome graphics the transparency color can not be selected but the graphic or frame can be set to transparent.

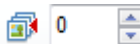


### **Animate Graphics [Ctrl+I]**

Animate the image in the preview window.

Note that the currently selected repetition rate is ignored for the preview, the animation is repeated endlessly.

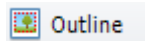
When animated preview is turned on, single frames can not be selected anymore.



### **Select Frame**

Select the frame of an animated picture to be displayed in the preview.

Note that turning on animated preview disables this field.



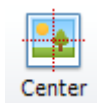
### **Show Graphic Outline**

Display a dashed outline border showing the size of the picture or frame.



### **Proof Transparency**

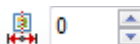
Show transparent areas of the image filled with a pattern.



### **Center Graphic**

Automatically center the graphic in the preview.

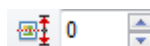
Note that enabling auto-centering disables the X and Y position fields.



### **Set X Coordinate**

Set the X coordinate of the graphic in the preview window.

Note that this value is used when setting the graphic as a startup image as well.



### **Set Y Coordinate**

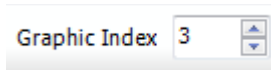
Set the Y coordinate of the graphic in the preview window.

Note that this value is used when setting the graphic as a startup image as well.



### **Set/Unset Startup Graphic**

You can set the graphic as a static or animated startup image.

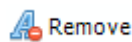
**Graphic Index [Ctrl+J]**

Select index of graphic to be shown.

**Fonts****Add New Font [Ctrl+A]**

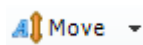
Add new font item by either

- converting a Windows font or
- loading a font from a file.

**Remove Font**

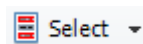
Remove font item from the list.

Please note, that removing the item can't be undone!

**Move Font Item**

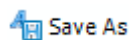
Move font item up or down in the list by changing the actual index in the list.

Note that moving an item can be done by dragging it with the mouse as well.

**Select Font Items**

Select multiple font items enabling you to perform operations like Remove or Reload (but not Move) on multiple items with a single mouse click. Selected items are marked with a red item index.

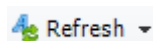
Note that you can select a range by holding down the shift button while clicking on the item.

**Save as: Save Font Item**

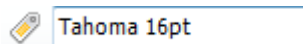
Save font item to disk.

**Reload Font Item**

Reload font item either by loading it from a file or by converting a Windows font.

**Refresh Font Item**

Refresh font item by reloading it from the disk or re-converting the appropriate Windows font depending on the original position. Note that refreshing can be done only when the file is available at the original location respectively the original Windows font exists on your system


**Font Item Name [Ctrl+N]**

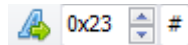
Enter the name to be used for the actual font item. Names can be used instead of index numbers to address the item.

Note that the first item name found will be used so names should be unique.

**Use Hex Number Base**

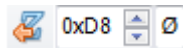
Use Hex number base for all character ASCII codes when pressed.

When the button is released () , decimal values are used.



### **First Character / First Character (ASCII)**

Set first character to be used for the current font item by selecting the appropriate decimal/hex ASCII value. Dec/hex switching is carried out with the "Use Hex Number Base" Button. Make use of the up/down arrows. It is also possible to fill in the dec/hex value via the keyboard, followed by an [Enter]. In the text field to the right of the up/down arrows, set the first character to be used for the current font item by typing the appropriate character.



### **Last Character / Last Character (ASCII)**

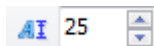
Set last character to be used for the current font item by selecting the appropriate decimal/hex ASCII value. Dec/hex switching is carried out with the "Use Hex Number Base" Button. Make use of the up/down arrows. It is also possible to fill in the dec/hex value via the keyboard, followed by an [Enter]. In the text field to the right of the up/down arrows, set the last character to be used for the current font item by typing the appropriate character.

## **Unicode Character Filters**



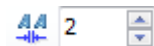
### **Character Width**

Set the maximum character width in pixels.



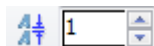
### **Character Height**

Set the maximum character height in pixels.



### **Horizontal Spacing**

Set the horizontal space between characters in pixels.



### **Vertical Spacing**

Set the vertical space between lines in pixels.

**Fixed**

### **Set As Fixed Pitch Font**

Set as fixed pitch font with all characters having the same width.

**Symbol**

### **Set As Symbol Font**

Set as symbol font having no space between characters and lines.

**Startup**

### **Set As Startup Font**

Activate as startup font being active after power up and reset. Note that there has to be one startup font. To deactivate it again, another font must be set.

**System**

### **Set As System Font**

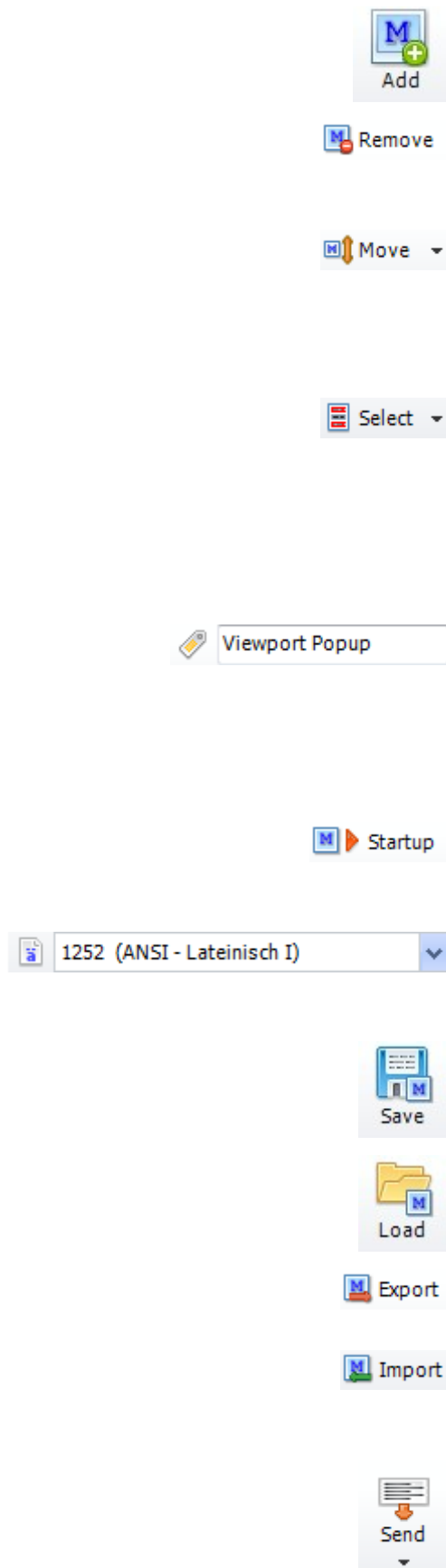
Activate as system font used for system messages. Note that there has to be one system font. To deactivate it again, another font must be set.

Font Index 1

### **Font Index [Ctrl+J]**

Select index of font to be shown.

## Macros



### **Add New Macro Item [Ctrl+A]**

Add new macro item by appending a new empty macro at the end.

### **Remove Macro Item**

Remove macro item from the list.  
Please note, that removing the item can't be undone!

### **Move Macro Item**

Move macro item up or down in the list by changing the actual index in the list.  
Note that moving an item can be done by dragging it with the mouse as well.

### **Select Macro Items**

Select multiple macro items enabling you to perform operations like Remove (but not Move) on multiple items with a single mouse click. Selected items are marked with a red item index.  
Note that you can select a range by holding down the shift button while clicking on the item.

### **Macro Item Name [Ctrl+N]**

Enter the name to be used for the actual macro item. Names can be used instead of index numbers to address the item.  
Note that the first item name found will be used so names should be unique.

### **Set/Unset Startup Macro**

Pushing the button sets the current macro as the startup macro executing on power-up automatically.

### **Code Page**

Select code page from the list to be used for the current macro.

### **Save Macros**

Save macro data to a file.

### **Load Macros**

Load macro data from a file. This will replace all existing macros with the new data from the file.

### **Export Macro Item**

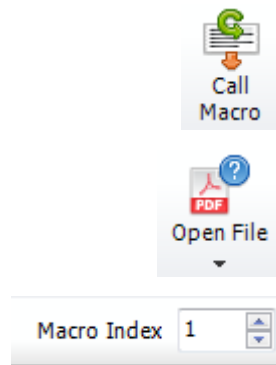
Export current macro to a file.

### **Import Macro Item**

Import a macro by adding the content of the file to a new macro appended.

### **Send Commands [Ctrl+Enter]**

Send commands from the editor window to the iLCD panel.

**Call Macro [Shift+Enter]**

Call current macro by sending \iOE\D0 to the iLCD panel.

**Open Documentation [Shift+Ctrl+F1]**

Open the command documentation PDF file with the default PDF viewer.

**Macro Index [Ctrl+J]**

Select index of macro to be shown.

## Messages

**Add New Message Item [Ctrl+A]**

Add new message item by appending a new empty message at the end.

**Remove Message Item**

Remove message item from the list.  
Please note that removing the item can't be undone!

**Move Message Item**

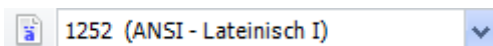
Move message item up or down in the list by changing the actual index in the list.  
Note that moving an item can be done by dragging it with the mouse as well.

**Select Message Items**

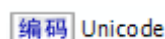
Select multiple message items enabling you to perform operations like Remove (but not Move) on multiple items with a single mouse click. Selected items are marked with a red item index.  
Note that you can select a range by holding down the shift button while clicking on the item.

**Message Item Name [Ctrl+N]**

Enter the name to be used for the actual message item. Names can be used instead of index numbers to address the item.  
Note that the first item name found will be used so names should be unique.

**Code Page**

Select code page from the list to be used for the current message.  
Note that there are no code pages selectable if the message is set to Unicode.

**Set to Unicode**

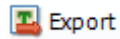
Set the current message to be stored in Unicode.

**Save Messages**

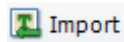
Save message data to a file.

**Load Messages**

Load message data from a file. This will replace all existing messages with the new data from the file.

**Export Message Item**

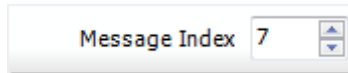
Export current message to a file.

**Import Message Item**

Import a message by adding the content of the file to a new message appended.

**Show Commands in Hex**

Show commands to be sent to the iLCD panel in Hex as well.

**Message Index [Ctrl+J]**

Select index of message to be shown.



## Settings



### **Enable Category**

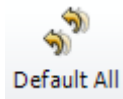
If a category like 'I/O Settings' is not enabled the appropriate data is not written to the iLCD Controller when writing the project. Values already existing in the iLCD Controller are used instead.

This can help you to not accidentally overwrite important settings in the iLCD Controller.



### **Default**

Set default values for the currently selected category.



### **Default All**

Set default values for all categories.



### **Save Settings**

Save setting data to a file.



### **Load Settings**

Load setting data from a file.

### Project Settings

Startup Values

Display Settings

Hardware Settings

I/O Settings

TCP/IP Settings

Keyboard Settings

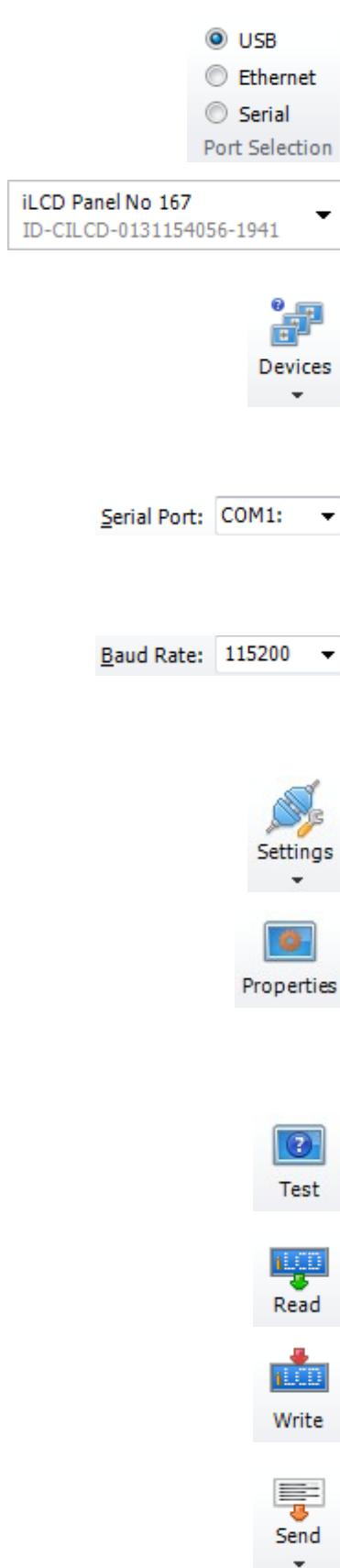
PC Control

Messages

### **All Settings Categories**

In order to make or change a setting in a certain category, click on the category. Making/changing settings can be enabled with the "Enable" button (see above) or by a click onto the checkbox.

## Device



The screenshot shows a software interface for configuring an iLCD panel. At the top, there is a 'Port Selection' section with three radio buttons: 'USB' (selected), 'Ethernet', and 'Serial'. Below this is a dropdown menu for 'iLCD Panel No' showing '167' and 'ID-CILCD-0131154056-1941'. A 'Devices' button with a question mark icon is below. Further down are two dropdown menus: 'Serial Port:' set to 'COM1:' and 'Baud Rate:' set to '115200'. Below these are four buttons: 'Settings' (with a screwdriver icon), 'Properties' (with a monitor icon), 'Test' (with a question mark icon), and 'Read' (with an iLCD icon and a green arrow). At the bottom are two more buttons: 'Write' (with an iLCD icon and a red arrow) and 'Send' (with a document icon and an orange arrow).

### **Select Communications Port**

Select USB, Ethernet or serial port.

### **Currently selected USB/Ethernet device**

Lists assigned device name and the serial number.  
Note: available if USB or Ethernet port is selected.

### **Enumerate Devices**

- Enumerate devices or
- Attach a name to a serial number

Note: available if USB or Ethernet port is selected.

### **Select Serial Port Name**

Select serial port - Enter the port name if not showing in the list.

Note: available if serial port is selected.

### **Select Baud Rate**

Select the baud rate matching the settings of the currently connected iLCD panel.

Note: available if serial port is selected.

### **Change Serial Port Settings**

Change the appropriate serial port settings.

Note: available if serial port is selected.

### **Get/Set Properties**

- Save and restore display settings
- View the iLCD panel's properties
- Update the iLCD Controller's firmware

### **Test Device**

Test the connected iLCD panel.  
The backlight of the display will blink.

### **Read Project**

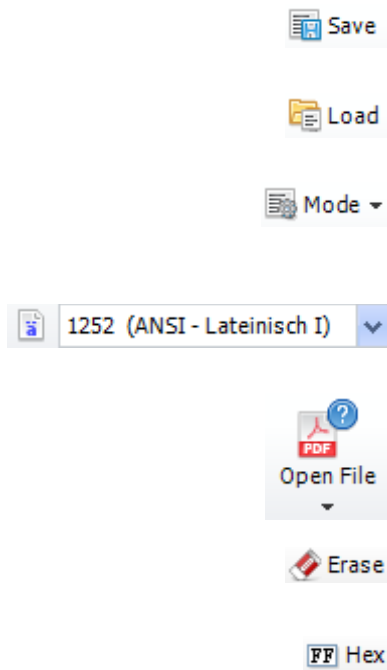
Read back the project data from the currently connected iLCD panel.

### **Write Current Project [Ctrl+W]**

Write the current project to the iLCD panel currently connected.

### **Send Commands [Ctrl+Enter]**

Send commands from the editor window to the iLCD panel.

**Save Commands**

Save commands from the editor window to a file.

**Load Commands**

Load commands to be sent to the iLCD panel from a file.

**Set White Space Mode**

Determine how to handle white space characters (spaces and carriage returns / linefeeds).

**Code Page**

Select code page from the list to be used for iLCD commands.

**Open Documentation [Shift+Ctrl+F1]**

Click the button to open the PDF file.

Click the option indicator to open/close the Control Sequences hint window.

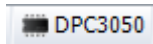
**Erase [Shift+Ctrl+BkSp]**

Erase commands to be sent to the iLCD panel.

**Show Commands in Hex**

Show commands to be sent to the iLCD panel in Hex as well.

## Status Bar



### **iLCD Controller**

Shows the iLCD Controller the project is designed for.



### **Flash Memory Size**

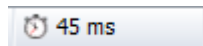
Shows the total amount of flash memory occupied for the project.

To see further details, click on the memory size.



### **Transferring Data**

Shows the progress of data transfer to or from the iLCD panel.



### **Execution Time**

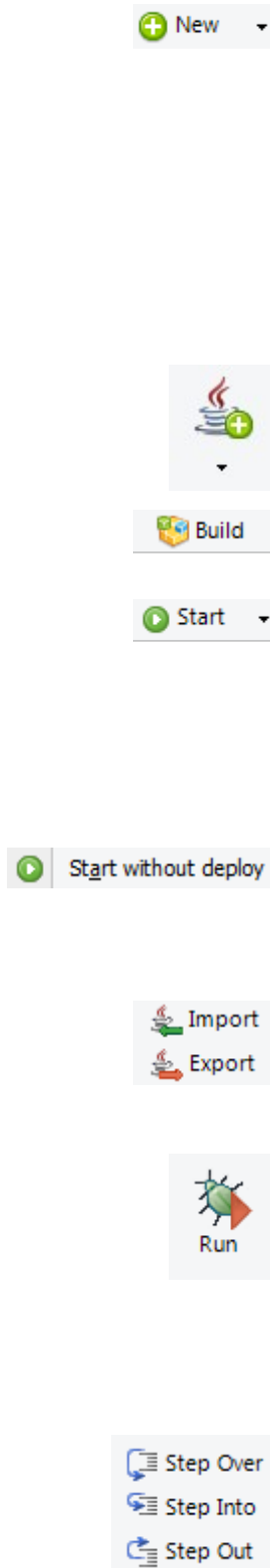
Time used for processing.



### **Connection State**

Shows the way how the iLCD panel ist connected to the computer.

## Java IDE



### **Create new Java App**

Create new Java App and add it to the current iLCD Project. Two file dialogs will open for creating an application class file and a main class file. Class templates will be inserted into these files.

Note that there has to be exactly one class (the Main Class) containing the main method with the following signature:

```
public static void main(String[] args)
```

This is the entry point of the Java program.

### **Add new Java File to App**

Open a file dialog and add the created file to the Java App. A class template will be inserted.

### **Build Java App [Ctrl+F7]**

Compile Java source code and create Java Binary.

### **Start Java Binary [Ctrl+F5]**

Execute Java Binary on the iLCD panel without starting the debugger. Depending on the iLCD Manager XE program options the iLCD project and/or the Java Binary will be automatically loaded to the iLCD panel. Note that in this case the Java Binary on the iLCD panel will be overwritten. Select "Start immediately" from the drop down menu to avoid this in any case.

### **Start Java Binary without Deploy**

Start Java Binary on the iLCD panel without downloading it (requires previous download). So the Java Binary that is already present will not be overwritten.

### **Export/Import Java App**

Export/Import Java App data to/from binary file (\*.jproj).

This might be useful for importing the Java App data into another iLCD project.

### **Start or continue Debugger [F5]**

Start the Debugger. Depending on the iLCD Manager XE program options the iLCD project and/or the Java Binary will be automatically loaded to the iLCD panel.

Note that the execution of the Java Binary continues immediately. Use "Step Into" to stop execution on the first line.

When the Debugger is already running this button is used to continue execution (e.g. when stopped at a breakpoint).

### **Debug Step Commands**

- Step over actual instruction. [F6]
- Step into function call at current line. [F7]
- Step out of actual function call. [F8]

Note that stepping goes by instruction and not by line, so it might happen that stepping to the next line requires multiple Step over executions (e.g. in for-loops).

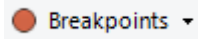


### **Add Watch**

Open dialog for adding a symbol to the watch list.

This can also be done via the right mouse click menu of the Java source code editor.

Note that it is also possible to inspect symbol values by hovering over the symbol in the source code. If the symbol is in the current scope a tooltip will be shown.

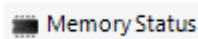


### **Breakpoints**

Add breakpoint to the current line.

Note that all breakpoints can be listed via the drop down menu entry "Open Breakpoint List" (also available via the right mouse click menu of the Java source code editor).

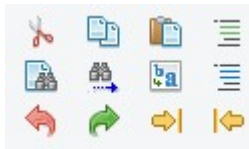
Note also that it's possible to specify a pass count and the Thread for each breakpoint via the breakpoint list.



### **Memory Status**

Get information about the Java Virtual Machine memory status.

Show free Heap size, Java Binary size, etc.



### **Edit Source Code**

Toolbar with various edit options such as:

- Comment source code region.
- Indent/Outdent source code.
- Open Find/Replace dialog.



### **Add new**

Adds a new file to the project.



### **Save Settings**

Save setting data to a file.



### **Load Settings**

Load setting data from a file.

## iLCD Command Set

This section describes the command set for iLCD Controllers DPC3050 and higher, which is a superset of the commands for the controllers up to DPC3020.

When using earlier or monochrome models, please refer to the command documentation for DPC10xx / DPC20xx / DPC3020 iLCD Controller: [iLCD Commands.pdf](#)

There is also a local *iLCD Commands.pdf* version available, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

The commands controllers up to DPC3020 can entirely be used for the DPC3050 and higher, newly added functions for current controllers are:

- Turn the LCD display on/off (refer to chapter [Turn Display On/Off](#))
- Viewports (refer to chapter [Viewport Related Commands](#))
- Text and graphics orientation (refer to chapter [Set Text/Graphic Orientation](#))

New functions for the iLCD Controllers DPC3080 and higher are:

- Multiple Screens (refer to chapter [Screen Memory Related Commands](#))
- MicroSD card support (refer to chapter [MicroSD Card Related Commands](#))
- Extended graphic commands for file handling (refer to chapters [Display Local Graphic](#) and [Load Animated Graphics](#))
- Flashing the application data by using the MicroSD card as source (refer to chapter [Write Application Data to Flash](#))

Furthermore, there are differences when addressing the USB port and the flash memory. For more information refer to <http://www.demmel.com/download/ilcd/iloader.zip>. Its source code handles all available iLCD Controllers automatically and can be adapted for your own needs.

---

## Command Overview

Find following sections in this chapter:

- [Command Structure](#)
- [Syntax used in iLCD Manager XE](#)
- [Terminal Mode](#)
- [ANSI Support](#)
- [Commands Sorted by Functionality](#)
- [Commands Sorted by Command Sequence](#)

For detailed descriptions of all available commands refer to chapter [Command Description](#).

---

## Command Structure

All commands must start with a special command introducer (written as `\i` in iLCD Manager XE), which has the value 0xAA, and the command characters followed by the data block. This allows the controller to resynchronize in case of data errors or hang-ups of the controlling application. If the data block contains a 0xAA character (e.g. a word value of decimal 170), it must be quoted by a preceding 0xAA character.

Mind this behavior when sending commands from any controlling application. In iLCD Manager XE's editor, quoting is done automatically and therefore can be omitted. Note that inserting 0xAA characters manually in this case leads to syntax errors.

All commands respond with an `[ACK]` (0x06) character after they have been processed. If the command is unknown or contains an invalid value, a `[NACK]` (0x15) is returned as soon as the invalid value is recognized and the controller returns to "sync mode", which means it waits for a new command introducer (`\i`). In this case, the source of the problem is saved as an error code and can be accessed via the command [Get Last Error Code](#). A complete list can be found in the section [Error Codes](#). If an unexpected command introducer is found, processing of the current command is cancelled, and the next character is tried to be interpreted as a command. Any characters after a successful interpreted command, which are not a command introducer, are simply ignored.

Special considerations regarding input buffer size have to be taken into account when communicating via serial port (see [Controlling the iLCD via Serial Port](#)).

Find sections describing the possible parameter types in this chapter:

- [Byte \(8-Bit\) Values](#)
  - [Boolean Bytes](#)
  - [Bit Mask Bytes](#)
  - [Signed Bytes](#)
- [Word \(16-Bit\) Values](#)
  - [Bit Mask Words](#)
  - [Signed Words](#)
- [Long \(32-Bit\) Values](#)
- [String Parameter](#)
  - [Unicode Strings](#)
  - [Graphic, Font, Macro and Message Names](#)
  - [MicroSD Paths and Filenames](#)
- [16-Bit Color Values](#)
- [24-Bit Color Values](#)

---

### **Byte (8-Bit) Values**

Some commands require a 8-bit byte describing the parameter.

In the chapter [Command Description](#), normal 8-bit values are always described as type "byte" in the corresponding parameter tables.

#### **Boolean Bytes**

Boolean parameters can only be set to the values 1 (true) and 0 (false).



In the parameter tables of chapter [Command Description](#), boolean bytes are always described as type "bool".

### Bit Mask Bytes

In bit mask bytes, every single bit can be interpreted as a boolean flag. Bits not defined in the parameter are ignored (refer to according tables in chapter [Command Description](#)).

In parameter tables, byte bit masks are always described as "bits".

#### Syntax in iLCD Manager XE:

```
\iCi\36
\iCi\d36
\iCi\x24
\iCi$
```

All above notations represent the same value (36 resp. 0x25). For certain byte values (e.g. touch field keys) it can be convenient to use the ASCII code of characters (ASCII 0x25 corresponds with '\$'). Refer to [Syntax used in iLCD Manager XE](#) for further information about these introducters.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like this:

```
[AA] [43] [69] [24]
```

### Signed Bytes

Signed bytes have a value range from decimal –128 to +127. If the value is positive, the hex value is the simple hex representation of the decimal value; if it is negative the calculation can be done as follows:

$$\text{hex value} = \text{hex}(256 - \text{abs}(\text{value}))$$

This means a value of 0x80 represents decimal –128 and a value 0xFF decimal –1. The decimal value 100 translates to 0x9C.

In parameter tables, signed bytes are always described as "signed byte".

#### Syntax in iLCD Manager XE:

```
\iAu\–4
\iAu\d–4
\iAu\xFC
```

All above notations represent the same value (–4 resp. 0xFC).

The sequence sent via the interface for this example would look like this:

```
[AA] [41] [75] [FC]
```

## **Word (16-Bit) Values**

Some commands require a 16-bit word describing the parameter. All these 16-bit values must be sent in the form of high byte first and low byte next. That means that to send a value such as decimal 345 (= 0x159) to the iLCD Controller, 0x01 followed by 0x59 has to be sent.

In the chapter [Command Description](#), 16-bit parameters are always described as type "word" in the corresponding parameter tables.

### **Syntax in iLCD Manager XE:**

```
\iG\D333
\iG\X14D
\iG\d1\d77
\iG\x1\x4D
```

All above notations represent the same value (333 resp. 0x14D). Note that word values may also be specified by a pair of bytes. Refer to [Syntax used in iLCD Manager XE](#) for further information about these introducters.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like this:

```
[AA] [47] [01] [4D]
```

## **Bit Mask Words**

In bit mask words, every single bit can be interpreted as a boolean flag. Bits not defined in the parameter are ignored (refer to according tables in chapter [Command Description](#)).

In parameter tables, word bit masks are always described as "wbits".

### **Syntax in iLCD Manager XE:**

```
\iILs\XA\D1280
```

This turns on outputs #1 and #3 (Hex 0xA = Bin 0000 0000 0000 1010) and sets outputs #8 and #10 (decimal 1280 = Hex 0x500 = Bin 0000 0101 0000 0000) to blink.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like this:

```
[AA] [49] [4C] [73] [00] [0A] [05] [00]
```

## **Signed Words**

Signed words have a value range from decimal -32768 to +32767. If the value is positive, the hex value is the simple hex representation of the decimal value; if it is negative the calculation can be done as follows:

$$\text{hex value} = \text{hex}(65536 - \text{abs}(\text{value}))$$

This means a value of 0x8000 represents decimal -32768 and a value 0xFFFF decimal -1. The decimal value 10000 translates to 0xD8F0.

Please note that signed words have to be sent in the order: high byte first, low byte next (same as for standard 16-bit values).

### Syntax in iLCD Manager XE:

```
\iCc\D-300
\iCc\XFED4
```

All above notations represent the same value (-300 resp. 0xFED4).

The sequence sent via the interface for this example would look like this:

```
[AA] [43] [63] [FE] [D4]
```

### Long (32-Bit) Values

A very few commands require a 32-bit (long) value describing the parameter. All these 32-bit values must be sent in the form of four bytes - high byte first and low byte last. That means that to send a value such as decimal 115200 (= 0x001C200), 0x00, 0x01, 0xC2 and 0x00 have to be sent in that order.

In the chapter [Command Description](#), 32-bit parameters are described as type "long" in the corresponding parameter tables.

### Syntax in iLCD Manager XE:

```
\iIB\1\L115200
\iIB\1\X1\XC200
\iIB\x1\x0\x1\xC2\x0
```

All above notations represent the same value (115.200 resp. 0x1C200). Note that long values may also be specified by a pair of words or 4 bytes. Refer to [Syntax used in iLCD Manager XE](#) for further information about these introducters.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like this:

```
[AA] [49] [42] [01] [00] [01] [C2] [00]
```

### String Parameter

Some commands require a string parameter. In the chapter [Command Description](#), string parameters are always described as type "string" in the corresponding parameter tables.

#### Note

- To terminate a string, a NULL byte (0x00) has to be placed.
- Strings may contain any ASCII codepage character between 0x01 and 0xFF.
- Characters not defined in the character table of a font are replaced with a space.

- Unquoted 0xAA bytes will interrupt string output and begin a new command (refer to [Syntax used in iLCD Manager XE](#)).

### Syntax in iLCD Manager XE:

```
\iDTHello World!\0
```

When entering codepage strings in iLCD Manager XE (using [Write Text](#) resp. [Get Text Extent](#)), the character codes are interpreted according to the codepage of the current macro, message or device terminal. Purple underlined characters indicate that they do not exist in this codepage.

As codepage strings are interpreted byte-wise, any [word](#) (2 bytes) or [long](#) (4 bytes) values will be interpreted as string characters. For example, `\X3` will be interpreted as the ASCII values for `'\'` (0x5C), `'X'` (0x58) and `'3'` (0x33) and `"\X3"` will be displayed.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like this (ASCII characters are interpreted in this case):

```
AAH D T H e l l o 20H W o r l d ! 00H
```

### Unicode Strings

From firmware version 4.0 onward, iLCDs support the display of Unicode fonts too. In these, every character is addressed by a 2-byte character code (0x0001 .. 0xFFFF) thus allowing for a wider range of available glyphs (symbols). They can be used to depict languages like Chinese, Japanese, Korean, Cyrillic, Hebrew and many more.

In the ASCII range (0x00 .. 0xFF), the Unicode character codes correspond with the ones of the codepage 1252 (ANSI - Latin I) including the regional characters for most Latin languages (English, Spanish, French, German,...) starting from 0x80. When selecting any other codepage, this second half of characters contains other region-specific glyphs and is then not corresponding with the Unicode values any more.

The following table shows a few examples for character codes in different codepages as well as in the Unicode character set, no character code means the symbol is not present in the respective codepage:

Codepage	A	ä	ø	交
1252 (ANSI - Latin I)	0x41	0xE4	-	-
1253 (ANSI - Greek)	0x41	-	0xE4	-
65001 (Unicode)	0x0041	0x00E4	0x03B4	0x4EA4

Therefore it is possible to output single-byte strings with a Unicode font as long as the "Basic Latin" language script is included when converting the font in iLCD Manager XE. In the same way, 2-byte (Unicode) strings can be displayed using a single-byte (codepaged) font. All characters not defined in the font will then be replaced with spaces, text output still continues working and responds with `[ACK]`.

In the chapter [Command Description](#), Unicode string parameters are always described as type "wstring" in the corresponding parameter tables.

**Note**

- To terminate a string, a NULL word (0x0000) has to be placed.
- Unquoted 0xAA bytes will interrupt string output and begin a new command (refer to [Syntax used in iLCD Manager XE](#)).

**Syntax in iLCD Manager XE:**

```
\iDUHello World!\0
```

When entering Unicode strings in iLCD Manager XE, all characters as well as decimal or hex values are interpreted as words (2 bytes). So a `\0` translates to 0x0000, `\r` to 0x000D and also `\5`, `\d5` or `\x5` will be filled to 0x0005.

Note that Unicode strings are not indicating missing character glyphs as codepage strings do. The sequence for the above example would look like this:

```
AAH D U 00H H 00H e 00H l 00H l 00H o 00H 20H 00H W 00H o 00H r 00H l 00H d 00H ! 00H 00H
```

**Graphic, Font, Macro and Message Names**

Commands using graphic, font, macro or message indices can alternatively be addressed by a unique name for the according element. In this case - instead of the expected index - 0xFFFE has to be sent, followed by the element's name as a 0-terminated ASCII string comprised of a maximum of 32 characters.

In iLCD Manager XE, the sequence `\m` inserts the name index (0xFFFE).

In the chapter [Command Description](#), a name parameter replacing an index is always described as type "by\_name" in the corresponding parameter tables.

**Syntax in iLCD Manager XE:**

```
\iAF\mTahoma 10pt\0
```

The sequence for this example would look like this:

```
AAH A F FFH FEH T a h o m a 20H 1 0 p t 00H
```

**MicroSD Paths and Filenames**

Paths and filenames on the on-board MicroSD card have to be named with DOS filenames in 8.3 format in order to be accessible.

When using filenames instead of indices, 0xFFFF followed by the complete path to the according file has to be sent.

In iLCD Manager XE, the sequence `\f` inserts the filename index (0xFFFF).

In the chapter [Command Description](#), filename parameters are described as type "file" in the corresponding parameter tables. A filename parameter replacing an index however is described as "by\_file".

## Syntax in iLCD Manager XE:

```
\iG\ffPath/Backgr~1.rii\0
```

The sequence for this example would look like this:

```
AAH G FFH FFH P a t h / B a c k g r ~ 1 . r i i 00H
```

## 16-Bit Color Values

The Color iLCD controller supports commands for setting colors such as background color or foreground color. Although the Color iLCD controller internally works with 16-bit color values, all commands except the read and write scan line commands use 24-bit color values (see at [24-Bit Color Values](#)) to allow future expansions. The [read](#) and [write](#) scan line commands use the 16-bit color values only to minimize the amount of data to be read and written, as one 320x240 color pixel screen needs  $320 \times 240 \times 2 = 153,600$  bytes to be read/written even in 16-bit color format.

In the chapter [Command Description](#), 16-bit color values are always described as type "color\_16bit" in the corresponding parameter tables.

The bit assignment of the 16-bit color values is R5G6B5, which make up one 16-bit word as follows:

*RRRRRGGGGGBBBBB* (most significant bit is leftmost)

The formula (C-notation) to get a 16-bit color value from the red/green/blue parts is as follows:

$$color\_16\_bit = (red \ll 11) + (green \ll 5) + blue$$

where *red* and *blue* has a range of 0..31 and *green* has a range of 0..63, the maximum value for *red/green/blue* refers to 100% color intensity. Shifting *red* left by 11, shifting *green* left by 5 and adding the shifted red and green value and blue together delivers the 16-bit color value.

Some color values shown in binary and hex representation explain the color values further:

Color	Red (Dec. /%)	Green (Dec. /%)	Blue (Dec. /%)	Binary Value	Hex Value
Pure red	31 / 100%	0 / 0%	0 / 0%	1111100000000000	F800
Pure green	0 / 0%	63 / 100%	0 / 0%	0000011111100000	07E0
Pure blue	0 / 0%	0 / 0%	31 / 100%	000000000011111	001F
Black	0 / 0%	0 / 0%	0 / 0%	0000000000000000	0000
White	31 / 100%	63 / 100%	31 / 100%	1111111111111111	FFFF
Yellow	31 / 100%	63 / 100%	0 / 0%	1111111111100000	FFE0
Magenta	31 / 100%	0 / 0%	31 / 100%	1111100000011111	F81F
Cyan	0 / 0%	63 / 100%	31 / 100%	0000011111111111	07FF

In the following description of these commands, the parameters of a 16-bit color value are always written as *xxx\_hb* and *xxx\_lb* where *xxx* describes the 16-bit color parameter.

This is the command for writing a (partial) scan line with *no\_of\_pixels* length made up of pixels *p1*, *p2*, etc. An example showing the hex parameters for setting a red, a green and a blue pixel starting from the current cursor position is as follows:

### Syntax in iLCD Manager XE:

```
\iDNW\D3\XF8\XE007\X1F00
\iDNW\d0\d3\x0\xF8\xE0\x7\x1F\x0
```

All above notations represent the same value (set 3 pixels: red, green, blue). Note, that only for 16-bit color values, the 2 bytes are sent low byte first. Refer to [Syntax used in iLCD Manager XE](#) for further information about these introducters.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like:

```
[AA] [44] [4E] [57] [00] [03] [00] [F8] [E0] [07] [1F] [00]
```

## 24-Bit Color Values

The Color iLCD controller supports commands for setting colors such as background color or foreground color. Although the Color iLCD controller internally works with [16-Bit Color Values](#), all commands except the read and write scan line commands use 24 bit color values to allow future expansions.

In the chapter [Command Description](#), 24-bit color values are always described as type "color" in the corresponding parameter tables.

The 24-bit color value is converted to a 16-bit color value by the iLCD controller internally by using the following formula:

$$color\_16\_bit = ((red \gg 3) \ll 11) + ((green \gg 2) \ll 5) + (blue \gg 3)$$

where *red*, *green* and *blue* are 8-bit values with a range between 0 and 255.

The color values must be set in a 24-bit format, where *color\_r* describes the 8-bit red part, *color\_g* the green and *color\_b* the blue part of the 24-bit color value.

Syntax for setting the background color:

```
\i A C B color_r color_g color_b
```

### Syntax in iLCD Manager XE:

```
\iACB\#00FF00
\iACB\x00\xFF\x00
\iACB\0\255\0
```

All above notations represent the same value (pure green). Refer to [Syntax used in iLCD Manager XE](#) for further information about these introducers.

The actual bytes sent via the interface can be verified in the "Hex" part of any editor panel. The sequence for the above example would look like this:

```
[AA] [41] [43] [42] [00] [FF] [00]
```

## Syntax used in iLCD Manager XE

iLCD Manager XE contains a terminal emulation allowing you to enter all possible commands used for iLCD Controllers in an easy-to-use way. Most simple terminal emulation programs do not allow you to enter non-ANSI characters like the command introducer (`\i` resp. `0xAA`) and so trying to use such programs is not advisable.

The iLCD Manager XE uses several sequences allowing you to enter any hex bytes or words very easily. If you need to lookup any of the command sequences, simply press the little arrow in the lower right corner of the "Documentation" group to display a list of these control sequences.

Seq	Hex-Value	Description
<code>\i</code>	AA	Command Introducer (0xAA)
<code>\...</code>	00 ... FF	Decimal <a href="#">Byte</a> (e.g. <code>\123</code> or <code>\-123</code> )
<code>\d...</code>	00 ... FF	Decimal <a href="#">Byte</a> (e.g. <code>\d123</code> or <code>\d-123</code> )
<code>\D...</code>	0000 ... FFFF	Decimal <a href="#">Word</a> (e.g. <code>\D12345</code> or <code>\D-1</code> )
<code>\L...</code>	0000 0000 ... FFFF FFFF	Decimal <a href="#">Long</a> (e.g. <code>\L1234567890</code> )
<code>\x...</code>	00 ... FF	Hex <a href="#">Byte</a> (e.g. <code>\x7B</code> )
<code>\X...</code>	0000 ... FFFF	Hex <a href="#">Word</a> (e.g. <code>\XAB12</code> )
<code>\#...</code>	00000 ... FFFFF	<a href="#">24-Bit Color Value</a> (e.g. <code>\#00FF00</code> or <code>\#CECECE</code> )
<code>\f</code>	FFFF	<a href="#">MicroSD Path and Filename</a> String Introducer (e.g. <code>\f/DIR/FILE.EXT\0</code> )
<code>\m</code>	FFFE	<a href="#">Graphic/Font/Macro/Message Name</a> String Introducer (e.g. <code>\mNAME\0</code> )
<code>\{...}</code>	-	Comment, characters between '{' and '}' are not sent
<code>\0</code>	00	NULL Character (NUL)
<code>\e</code>	1B	Escape Character (ESC)
<code>\b</code>	08	Back-Space Character (BS)
<code>\n</code>	0A	Line Feed (LF)
<code>\r</code>	0D	Carriage Return (CR)
<code>\s</code>	20	Space Character (SP)
<code>\t</code>	09	Tabulator Character (HS)



When reading the response from the left-hand side of the docking panel "Response from iLCD" in iLCD Manager XE, keep in mind that some hex values are interpreted and will be outputted differently. The actual value for the according byte can still be seen in the "Hex" part of this panel.

Seq	Hex-Value	Description
[ACK]	06	Acknowledge (command successfully processed)
[NACK]	15	Not acknowledge (command failed)
[BLACK]	04	Bootloader acknowledge (for internal use)
[EERR]	10	EEPROM write error (see <a href="#">Write EEPROM</a> )
[CSERR]	17	Checksum Error (when writing flash or EEPROM)
[OVR]	19	Input buffer overflow (on serial port, SPI or I <sup>2</sup> C)
[XON]	11	Input buffer high water (see <a href="#">Set XON/XOFF for Terminal Mode</a> )
[XOFF]	13	Input buffer low water (see <a href="#">Set XON/XOFF for Terminal Mode</a> )

### Example

When entering the following text in iLCD Manager XE:

```
\i!           { Reset All }
\iCK\D100\D50 { Set Cursor Position to decimal x = 100, y = 50 }
\iDTHello World!\0 { Write Text "Hello World!" at x = 100, y = 50 }
```

and pressing the "Send" button, the following Hex characters are sent:

Hex Bytes	Representation
AA 21	.!
AA 43 4B 00 64 00 32	.CK.d.2
AA 44 54 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00	.DTHello World!.

The hex output to be sent can also be monitored in iLCD Manager XE by clicking on the "Hex" button.

Please note that if you send the decimal word value `\D170` as a parameter in a command from any controlling application instead of iLCD Manager XE, this will cause the current command sequence to be terminated. This is because the least significant byte of `\D170` represents 0xAA and the controller will interpret it as such. The controller will then interpret the following bytes as the beginning of a new command with unexpected results. To prevent this, the value has to be 'quoted' by sending it twice. iLCD Manager XE adds these quoting characters automatically, refer to the Hex representation for verification.

So, for example, when setting the cursor to  $x = 170$ ,  $y = 50$ , following sequence has to be sent: AA (`\i`) 43 (`C`) 4B (`K`) 00 AA AA (`\D170`) 00 32 (`\D50`)

The 0xAA character is quoted in this way to tell the iLCD Controller not to start with a new command sequence but to use the AA character as a parameter instead (also refer to chapter [Command Structure](#)).

## Terminal Mode

The terminal mode allows the controlling application to send data to the iLCD module as if it were a standard ANSI controlled terminal. All keystrokes scanned by the iLCD module are reported as simple ASCII characters, so using the terminal mode enables the user of, for example, a Linux system to redirect the standard console to the iLCD module.

The iLCD modules can be switched to terminal mode via a command sequence (see [Go Terminal Mode](#)). They can also start in terminal mode when set on the "Settings" page of iLCD Manager XE. When in terminal mode, no commands can be sent to the iLCD until terminal mode is exited via the "End terminal mode" escape sequence (see [Private ANSI Extensions](#)). In terminal mode, keystrokes and touch events are not reported via the make and break sequences anymore; only the assigned key code is reported when the key or touch field is pressed. Refer to chapters [The Concept of iLCD's Touch Fields](#) and [Enable/Disable Keyboard Reporting](#) for detailed information.

If the controlling application cannot monitor the CTS pin (see [Controlling the iLCD via Serial Port](#)) the iLCD module can work with XON/XOFF control characters instead (see [Set XON/XOFF for Terminal Mode](#)). Please note that XON/XOFF mode is only active when running in terminal mode.

---

## ANSI Support

All text messages sent to the iLCD may contain ANSI control sequences, for cursor control, attribute settings and so on. For further information about what sequences exist and how they are interpreted refer to the chapters listed under [Standard ANSI Sequences](#) and [Private ANSI Extensions](#). Please note that setting ANSI mode to off causes the ANSI sequences to be interpreted as normal characters.

Using cursor control sequences from the standard ANSI sequences makes sense only if you have chosen a fixed pitch font as the iLCD controller always uses the maximum character width of the selected font (which is equal to the character width of any character on a fixed size font).

If you use a standard sequence such as `\e[4D` (go left 4 characters), and you have printed the text "Hill" via a proportional font, the resulting cursor position would be too far left as the "i" and "l" characters take much less space than the maximum character width used.

ANSI cursor control sequences will mainly be used on simple character output applications, which may use the iLCD's terminal mode. A typical application is showing the console output of a Linux system. In such cases there is no problem in using a fixed pitch font. More sophisticated applications can run the command mode or use the private ANSI extensions, which support many more possibilities.

Find following listings in this chapter:

- [Standard ANSI Sequences](#)
  - [Private ANSI Extensions](#)
-

## Standard ANSI Sequences

### Control Characters

Sequence	Hex-Code	Description
\r	0D	Carriage Return (CR) - Return to beginning of line
\n	0A	Line Feed (LF) - Go to next line
	0C	Form Feed (FF) - Same effect as LF
	0B	Vertical Tab (VT) - Same effect as LF
\b	08	Backspace (BS) - Go back one character
\t	09	Horizontal Tab (HT) - Go to next tabulator position
\e	1B	Escape (ESC) - Start of ANSI sequence

### Attribute Sequences

Sequence	Description
\e [ m	Set bold, underline and inverse mode off
\e [ 0 m	Set bold, underline and inverse mode off
\e [ 1 m	Set bold mode on
\e [ 4 m	Set underline mode on
\e [ 7 m	Set invert mode on

### Display Control Sequences

Sequence	Description
\e [ 2 J	Clear Display
\e [ K	Erase to end of line
\e [ 0 K	Erase to end of line

### Cursor Control Sequences

Sequence	Description
\e [ <n> A	Cursor up <n> rows
\e [ <n> B	Cursor down <n> rows
\e [ <n> C	Cursor right <n> columns
\e [ <n> D	Cursor left <n> columns
\e [ <x> ; <y> H	Go to <x> <y> position
\e [ <x> ; <y> f	Go to <x> <y> position

\e [ H	Go to home (set cursor to x=0 / y=0)
\e [ f	Go to home (set cursor to x=0 / y=0)
\e [ 6 n	Get cursor position as \e [ <x> ; <y> R

Please note, that <x> <y> is used in the opposite order to that in standard ANSI sequences.

### Save Restore Sequences

Sequence	Description
\e [ <n> s	Save cursor & attributes to memory position <n> (0 ... 7)
\e [ s	Save cursor & attributes to memory position 0
\e [ <n> u	Restore cursor & attributes from memory position <n> (0 ... 7)
\e [ u	Restore cursor & attributes from memory position 0

### See also:

[Private ANSI Extensions](#)

### Private ANSI Extensions

The iLCD's private ANSI extensions are not covered by an independent specification, so they will work on iLCD displays only.

Sequence	Description
\e { ! e N d	End terminal mode
\e { ! M	Get model info as [ACK] model_name [ACK]
\e { ! a	Get acknowledge as [ACK]
\e { <x> ; <y> H	Go to <x> <y> <u>pixel</u> position <b>(1-based)</b>
\e { <x> ; <y> f	Go to <x> <y> <u>pixel</u> position <b>(1-based)</b>
\e { H	Go to home
\e { f	Go to home
\e { 6 n	Get cursor <u>pixel</u> position as \e { <x> ; <y> R <b>(1-based)</b>
\e { <n> F	Select font number <n> <b>(1-based, 0 = startup font)</b>
\e { <n> G	Paint graphic number <n> <b>(1-based, 0 = startup graphic)</b>
\e { <n> s	Save cursor & attributes to memory position <n> (0 ... 7)
\e { s	Save cursor & attributes to memory position 0
\e { <n> u	Restore cursor & attributes from memory position <n> (0 ... 7)

<code>\e { u</code>	Restore cursor & attributes from memory position 0
---------------------	--

**See also:**

[Standard ANSI Sequences](#)

## Commands Sorted by Functionality

The depiction of commands in this list differs from the detailed descriptions in chapter [Command Description](#), where the syntax complies with the notation in iLCD Manager XE. Here, every character resp. word separated by spaces represents a byte sent to the iLCD controller (e.g. from the controlling application).

- the sequence `\i` describes the command introducer (0xAA)
- single characters are represented by their corresponding ASCII codes
- strings are depicted as `char1 char2 ... null`
- [Word \(16-Bit\) Values](#) ("word" in descriptions) are written as `xxx_hb xxx_lb` (meaning  $xxx\_hb * 256 + xxx\_lb$ )
- [Long \(32-Bit\) Values](#) ("long" in descriptions ) are written as `xxx_b3 xxx_b2 xxx_b1 xxx_b0` (meaning  $value\_b3 * 16,777,216 + value\_b2 * 65,536 + value\_b1 * 256 + value\_b0$ )

Section	Command Description	Command
General	<a href="#">No Operation</a>	<code>\i ' </code>
General	<a href="#">Reset All</a>	<code>\i !</code>
General	<a href="#">Reset All and Show Startup Graphic</a>	<code>\i \$</code>
General	<a href="#">Reboot Panel Controller</a>	<code>\i #</code>
General	<a href="#">Get Last Error Code</a>	<code>\i ? E</code>
General	<a href="#">Get Firmware Info</a>	<code>\i ? I</code>
General	<a href="#">Get Identification Info</a>	<code>\i ? M</code>
General	<a href="#">Get Firmware Version</a>	<code>\i ? V</code>
General	<a href="#">Get Serial Number</a>	<code>\i ? S</code>
General	<a href="#">Get iLCD Controller Name</a>	<code>\i ? C</code>
General	<a href="#">Get Hardware Revision</a>	<code>\i ? H</code>
General	<a href="#">Get Network Status</a>	<code>\i ? N</code>
General	<a href="#">Get Input Buffer Size</a>	<code>\i ? Q</code>
General	<a href="#">Get Project Info</a>	<code>\i ? P</code>
General	<a href="#">Get Touch Screen Type</a>	<code>\i ? T</code>
Viewports	<a href="#">Define Viewport</a>	<code>\i C V D viewport orientation width_hb</code>

		width_lb height_hb height_lb
Viewports	<a href="#">Select Viewport</a>	\i C V S viewport
Viewports	<a href="#">Copy Viewport</a>	\i C V C screen viewport copy_content pos_x_hb pos_x_lb pos_y_hb pos_y_lb
LCD-Control	<a href="#">Turn Display On/Off</a>	\i C D mode
LCD-Control	<a href="#">Set Screen Orientation</a>	\i C O orientation
LCD-Control	<a href="#">Set Text/Graphic Orientation</a>	\i C o orientation
LCD-Control	<a href="#">Enable/Disable ANSI</a>	\i C A on_off
LCD-Control	<a href="#">Set Column Address</a>	\i C C address_hb address_lb
LCD-Control	<a href="#">Increment/Decrement Column Address</a>	\i C c addr_inc_hb addr_inc_lb
LCD-Control	<a href="#">Set Row Address</a>	\i C R address_hb address_lb
LCD-Control	<a href="#">Increment/Decrement Row Address</a>	\i C r addr_inc_hb addr_inc_lb
LCD-Control	<a href="#">Set Cursor Position</a>	\i C K x_hb x_lb y_hb y_lb
LCD-Control	<a href="#">Set Relative Cursor Position</a>	\i C K x_inc_hb x_inc_lb y_inc_hb y_inc_lb
LCD-Control	<a href="#">Get Graphic Info</a>	\i C ? G graph_idx_hb graph_idx_lb
LCD-Control	<a href="#">Set Graphic Alignment</a>	\i C G A mode width_hb width_lb height_hb height_lb
LCD-Control	<a href="#">Get Cursor Position</a>	\i C ? K
LCD-Control	<a href="#">Get Text Extent</a>	\i C ? T char1 char2 ... null
LCD-Control	<a href="#">Get Unicode Text Extent</a>	\i C ? U char1_hb char1_lb char2_hb char2_lb ... null null
LCD-Control	<a href="#">Get Text Message Extent</a>	\i C ? t index_hb index_lb
LCD-Control	<a href="#">Set Text Alignment</a>	\i C T mode width_hb width_lb height_hb height_lb
LCD-Control	<a href="#">Set Line Style</a>	\i C L style
LCD-Control	<a href="#">Get Display Size</a>	\i C ? D
LCD-Control	<a href="#">Set TAB Spacing</a>	\i C S tab_spacing
LCD-Control	<a href="#">Set Auto-Linefeed</a>	\i C F on_off
LCD-Control	<a href="#">Set Wrap Mode</a>	\i C W horz_wrap vert_wrap
LCD-Control	<a href="#">Go Terminal Mode</a>	\i C G =
LCD-Control	<a href="#">Set XON/XOFF for Terminal Mode</a>	\i C X on_off
LCD-Control	<a href="#">Set Backlight Mode</a>	\i C B mode
LCD-Control	<a href="#">Get Backlight Mode</a>	\i C ? B

LCD-Control	<a href="#">Set Backlight Blink Frequency</a>	\i C b period
LCD-Control	<a href="#">Set Backlight Intensity</a>	\i C I intensity
LCD-Control	<a href="#">Set Backlight Intensity (High-Res)</a>	\i C i intensity
LCD-Control	<a href="#">Get Backlight Intensity</a>	\i C ? I
LCD-Control	<a href="#">Get Backlight Intensity (High-Res)</a>	\i C ? i
LCD-Control	<a href="#">Get Fixed LCD Contrast/Gamma</a>	\i C ? X
LCD-Control	<a href="#">Set LCD Contrast</a>	\i C N value
LCD-Control	<a href="#">Get LCD Contrast</a>	\i C ? N
LCD-Control	<a href="#">Set LCD Gamma Value</a>	\i C M value
LCD-Control	<a href="#">Get LCD Gamma Value</a>	\i C ? M
LCD-Control	<a href="#">Set Graphics Scaling</a>	\i C E G factor
LCD-Control	<a href="#">Set Fonts Scaling</a>	\i C E F factor
LCD-Control	<a href="#">Set Column Coordinates Scaling</a>	\i C E C mult div
LCD-Control	<a href="#">Set Row Coordinates Scaling</a>	\i C E R mult div
LCD-Attributes	<a href="#">Set Font</a>	\i A F number_hb number_lb
LCD-Attributes	<a href="#">Set Font Spacing</a>	\i A S x_spacing y_spacing
LCD-Attributes	<a href="#">Set Symbol Font</a>	\i A Y on_off
LCD-Attributes	<a href="#">Set Bold Mode</a>	\i A B on_off
LCD-Attributes	<a href="#">Set Underline Mode</a>	\i A U on_off
LCD-Attributes	<a href="#">Set Underline Position</a>	\i A u position
LCD-Attributes	<a href="#">Set Inverse Mode</a>	\i A I on_off
LCD-Attributes	<a href="#">Set Transparent Mode On/Off</a>	\i A T on_off
LCD-Attributes	<a href="#">Set Foreground Color</a>	\i A C F red green blue
LCD-Attributes	<a href="#">Set Background Color</a>	\i A C B red green blue
LCD-Attributes	<a href="#">Set Border Color</a>	\i A C R red green blue
LCD-Attributes	<a href="#">Set Border Shadow Color</a>	\i A C S red green blue
LCD-Attributes	<a href="#">Set Shadow Offset</a>	\i A H S byte::x_offset byte::y_offset
LCD-Attributes	<a href="#">Set Rectangle Corner Radius</a>	\i A H R word::radius_hb radius_lb
LCD-Attributes	<a href="#">Set Line Thickness</a>	\i A L T thickness
LCD-Attributes	<a href="#">Set Line Caps Style</a>	\i A L C style

LCD-Attributes	<a href="#">Set Line Ending Mode</a>	\i A L M mode
LCD-Attributes	<a href="#">Set Alpha</a>	\i A A A alpha
LCD-Attributes	<a href="#">Set Filling Color</a>	\i A P C red green blue
LCD-Attributes	<a href="#">Set Filling Gradient</a>	\i A P G mode ramp_hb ramp_lb from_color_red from_color_green from_color_blue to_color_red to_color_green to_color_blue
LCD-Attributes	<a href="#">Set Filling Tile</a>	\i A P T graphic_index_hb graphic_index_lb
LCD-Attributes	<a href="#">Set Adjustment for Graphics</a>	\i A A G mode
LCD-Attributes	<a href="#">Set Brightness Adjustment</a>	\i A A B brightness_hb brightness_lb
LCD-Attributes	<a href="#">Set Contrast Adjustment</a>	\i A A C contrast_hb contrast_lb
LCD-Attributes	<a href="#">Set Hue Adjustment</a>	\i A A H hue_hb hue_lb
LCD-Attributes	<a href="#">Set Saturation Adjustment</a>	\i A A S saturation_hb saturation_lb
LCD-Draw	<a href="#">Erase Display</a>	\i D E
LCD-Draw	<a href="#">Erase Display Area</a>	\i D e width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Invert Display</a>	\i D I
LCD-Draw	<a href="#">Invert Display Area</a>	\i D i width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Write Text</a>	\i D T char1 char2 ... null
LCD-Draw	<a href="#">Write Unicode Text</a>	\i D U char1_hb char1_lb char2_hb char2_lb ... null null
LCD-Draw	<a href="#">Write Text Message</a>	\i D t index_hb index_lb
LCD-Draw	<a href="#">Scroll Up</a>	\i D S U scroll_y_hb scroll_y_lb
LCD-Draw	<a href="#">Scroll Down</a>	\i D S D scroll_y_hb scroll_y_lb
LCD-Draw	<a href="#">Scroll Left</a>	\i D S L scroll_x_hb scroll_x_lb
LCD-Draw	<a href="#">Scroll Right</a>	\i D S R scroll_x_hb scroll_x_lb
LCD-Draw	<a href="#">Write Scan Line</a>	\i D N W no_of_pixels_hb no_of_pixels_lb p0_lb p0_hb p1_lb p1_hb ...
LCD-Draw	<a href="#">Read Scan Line</a>	\i D N R no_of_pixels_hb no_of_pixels_lb
LCD-Draw	<a href="#">Write Scan Line and Advance</a>	\i D N A no_of_pixels_hb no_of_pixels_lb p0_lb p0_hb p1_lb p1_hb ...
LCD-Draw	<a href="#">Write 1D/2D Run-Length Encoded Scan Line</a>	\i D N w prev_line_offset no_of_pixels_hb no_of_pixels_lb no_of_rle_bytes_hb no_of_rle_bytes_lb b0 b1 ...



LCD-Draw	<a href="#">Read 1D/2D Run-Length Encoded Scan Line</a>	\i D N r prev_line_offset no_of_pixels_hb no_of_pixels_lb
LCD-Draw	<a href="#">Set/Clear Pixel</a>	\i D P on_off
LCD-Draw	<a href="#">Set/Clear Pixel at X/Y</a>	\i D p x_pos_hb x_pos_lb y_pos_hb y_pos_lb on_off
LCD-Draw	<a href="#">Draw Dot</a>	\i D D red green blue
LCD-Draw	<a href="#">Draw Dot at X/Y</a>	\i D d x_pos_hb x_pos_lb y_pos_hb y_pos_lb red green blue
LCD-Draw	<a href="#">Draw Line</a>	\i D L end_x_hb end_x_lb end_y_hb end_y_lb
LCD-Draw	<a href="#">Draw Rectangle</a>	\i D R mode width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Draw Circle</a>	\i D C radius_hb radius_lb
LCD-Draw	<a href="#">Draw Styled Circle</a>	\i D c mode radius_hb radius_lb
LCD-Draw	<a href="#">Draw Ellipse</a>	\i D Y bits::mode word::vertex_a_hb word::vertex_a_lb word::vertex_b_hb word::vertex_b_lb
LCD-Draw	<a href="#">Adjust Display</a>	\i D A mode
LCD-Draw	<a href="#">Adjust Display Area</a>	\i D a mode width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Fill Display</a>	\i D F
LCD-Draw	<a href="#">Fill Display Area</a>	\i D f width_hb width_lb height_hb height_lb
LCD-Graphics	<a href="#">Display Local Graphic</a>	\i G graph_idx_hb graph_idx_lb
LCD-Graphics	<a href="#">Display Graphic Area</a>	\i g A x_hb x_lb y_hb y_lb width_hb width_lb height_hb height_lb graph_idx_hb graph_idx_lb
LCD-Graphics	<a href="#">Load Animated Graphics</a>	\i g L anim_loc index_hb index_lb
LCD-Graphics	<a href="#">Set Animation Coordinates to X/Y</a>	\i g K anim_loc pos_x_hb pos_x_lb pos_y_hb pos_y_lb
LCD-Graphics	<a href="#">Set Animation Coordinates to Cursor Position</a>	\i g k anim_loc
LCD-Graphics	<a href="#">Start or Restart Animation</a>	\i g S anim_loc
LCD-Graphics	<a href="#">Stop Animation and Set Frame Number</a>	\i g F anim_loc frame_hb frame_lb
LCD-Graphics	<a href="#">Stop (Break) Animation</a>	\i g B anim_loc
LCD-Graphics	<a href="#">Set Animation Repetitions</a>	\i g R anim_loc repeat_hb repeat_lb
LCD-Graphics	<a href="#">Erase Animation Image Area</a>	\i g E anim_loc
LCD-Graphics	<a href="#">Erase Animation Frame Area</a>	\i g e anim_loc

LCD-Graphics	<a href="#">Suspend Animation Engine</a>	\i g s
LCD-Graphics	<a href="#">Resume Animation Engine</a>	\i g r
LCD-Graphics	<a href="#">Move Animation To Frame</a>	\i g M anim_loc frame_idx_hb frame_idx_lb backwards
LCD-Graphics	<a href="#">Set Animation Background Color</a>	\i g b C anim_loc red green blue
LCD-Graphics	<a href="#">Set Animation Background Graphic</a>	\i g b G anim_loc bg_offset_x_hb bg_offset_x_lb bg_offset_y_hb bg_offset_y_lb graphic_idx_hb graphic_idx_lb
LCD-Graphics	<a href="#">Set Animation Background Frame</a>	\i g b F anim_loc frame_hb frame_lb
LCD-Graphics	<a href="#">Set Animation Background Screen</a>	\i g b S anim_loc bg_offset_x_hb bg_offset_x_lb bg_offset_y_hb bg_offset_y_lb screen
LCD-Graphics	<a href="#">Remove Animation Background</a>	\i g b N anim_loc
LCD-Screen Memory	<a href="#">Get # of Screens</a>	\i M S ?
LCD-Screen Memory	<a href="#">Set View Screen</a>	\i M V index
LCD-Screen Memory	<a href="#">Get View Screen Parameters</a>	\i M ? V
LCD-Screen Memory	<a href="#">Set Draw Screen</a>	\i M D index
LCD-Screen Memory	<a href="#">Get Draw Screen Parameters</a>	\i M ? D
LCD-Screen Memory	<a href="#">Copy Screen Area</a>	\i M A width_hb width_lb height_hb height_lb index x_hb x_lb y_hb y_lb
LCD-Screen Memory	<a href="#">Copy Screen To</a>	\i M S S index
LCD-Screen Memory	<a href="#">Copy Screen From</a>	\i M S C index
LCD-Screen Memory	<a href="#">Paint Screen From</a>	\i M S P index
LCD-Screen Memory	<a href="#">Invert Screen</a>	\i M S I index
LCD-Screen Memory	<a href="#">Scroll Up Screen</a>	\i M S U index scroll_y_hb scroll_y_lb
LCD-Screen Memory	<a href="#">Scroll Down Screen</a>	\i M S D index scroll_y_hb scroll_y_lb
LCD-Screen Memory	<a href="#">Scroll Left Screen</a>	\i M S L index scroll_x_hb scroll_x_lb
LCD-Screen	<a href="#">Scroll Right Screen</a>	\i M S R index scroll_x_hb scroll_x_lb

Memory		
LCD-Screen Memory	<a href="#">Set Height of Screen</a>	\i M S H index height_hb height_lb
LCD-Screen Memory	<a href="#">Set Width of Screen</a>	\i M S W index width_hb width_lb
LCD-Cursor Memory	<a href="#">Save Cursor &amp; Attributes to Memory</a>	\i M C S index
LCD-Cursor Memory	<a href="#">Restore Cursor &amp; Attributes from Memory</a>	\i M C C index
Macros	<a href="#">Execute Macro</a>	\i O E index_hb index_lb
Macros	<a href="#">Execute Protected Macro</a>	\i O e index_hb index_lb
Macros	<a href="#">Jump to Macro</a>	\i O J index_hb index_lb
Macros	<a href="#">Delay Macro Execution</a>	\i O D delay_hb delay_lb
Macros	<a href="#">Set Macro Execution Speed</a>	\i O S speed_hb speed_lb
Macros	<a href="#">Allow Keyboard/Touch Events to be Processed</a>	\i O K
Macros	<a href="#">Set Macro Timer</a>	\i O T time
Touch Screen	<a href="#">Calibrate Touch Screen</a>	\i T C
Touch Screen	<a href="#">Calibrate Touch Screen and Report</a>	\i T c
Touch Screen	<a href="#">Verify Touch Screen Calibration</a>	\i T V
Touch Screen	<a href="#">Set Touch Field Width</a>	\i T W width_hb width_lb
Touch Screen	<a href="#">Set Touch Field Height</a>	\i T H height_hb height_lb
Touch Screen	<a href="#">Set Touch Field Make Macro</a>	\i T M macro_idx_hb macro_idx_lb
Touch Screen	<a href="#">Set Touch Field Break Macro</a>	\i T B macro_idx_hb macro_idx_lb
Touch Screen	<a href="#">Enable/Disable Automatic Touch Macro Executing</a>	\i T E A on_off
Touch Screen	<a href="#">Set Touch Field Text Message</a>	\i T T text_message_idx_hb text_message_idx_lb
Touch Screen	<a href="#">Create/Define Touch Field</a>	\i T A field_idx key
Touch Screen	<a href="#">Remove Touch Field</a>	\i T R field_idx
Touch Screen	<a href="#">Global Enable/Disable Touch Fields</a>	\i T G on_off
Touch Screen	<a href="#">Set Current Touch Field Index</a>	\i T I field_idx
Touch Screen	<a href="#">Execute Touch Make Macro</a>	\i T E M field_idx
Touch Screen	<a href="#">Execute Touch Break</a>	\i T E B field_idx

	<a href="#">Macro</a>	
Touch Screen	<a href="#">Draw Touch Field Text Message</a>	\i T D field_idx
Touch Screen	<a href="#">Enable/Disable Touch Field Reporting</a>	\i T P on_off
Touch Screen	<a href="#">Enable/Disable Reporting Touch-Coordinates</a>	\i T K on_off
Touch Screen	<a href="#">Enable/Disable Reporting Movements</a>	\i T O on_off
Touch Screen	<a href="#">Retrieve Last Touch Screen Event</a>	\i T ?
Touch Screen	<a href="#">Set Number of Touch Fingers</a>	\i T S N fingers
Touch Screen	<a href="#">Set Threshold for Movement Reporting</a>	\i T S M threshold
Touch Screen	<a href="#">Set Cursor to Touch Field</a>	\i T E C field_idx
Touch Screen	<a href="#">Enable/Disable Touch Field Queue</a>	\i T Q on_off
Touch Screen	<a href="#">Start Demonstration Mode</a>	\i T v mode timeout_hb timeout_lb macro_index_hb macro_index_lb
Input/Output	<a href="#">Set Output</a>	\i I L S out_no mode
Input/Output	<a href="#">Set Multiple Outputs</a>	\i I L s out_mask_hb out_mask_lb blink_mask_hb blink_mask_lb
Input/Output	<a href="#">Set Output Blink Frequency</a>	\i I L F period
Input/Output	<a href="#">Set Relays On/Off/PWM</a>	\i I R relay_no mode
Input/Output	<a href="#">Relays One Shot</a>	\i I r relay_no mode time_hb time_lb
Input/Output	<a href="#">Enable/Disable Rotary Encoder Reporting</a>	\i I E on_off
Input/Output	<a href="#">Enable/Disable Keyboard</a>	\i I K E on_off
Input/Output	<a href="#">Enable/Disable Keyboard Reporting</a>	\i I K R on_off
Input/Output	<a href="#">Get Keyboard State</a>	\i I K ?
Input/Output	<a href="#">Set Baud Rate</a>	\i I B port_no baud_b3 baud_b2 baud_b1 baud_b0
Input/Output	<a href="#">Get Current Communication-Port</a>	\i I ? C
Input/Output	<a href="#">Disable Communication-Ports</a>	\i I C D port_mask timeout_hb timeout_lb
Input/Output	<a href="#">Get Enabled Communication-Ports</a>	\i I C ?
Input/Output	<a href="#">Get Inputs State</a>	\i I ? I

Input/Output	<a href="#">Get ADC Value</a>	\i I ? A port
SD Card	<a href="#">Request Disk Status</a>	\i S ?
SD Card	<a href="#">Make Directory</a>	\i S M dirname_char1 dirname_char2 ... null
SD Card	<a href="#">Get File Status</a>	\i S G filename_char1 filename_char2 ... null
SD Card	<a href="#">Open Directory and Read First Entry</a>	\i S F dirname_char1 dirname_char2 ... null
SD Card	<a href="#">Read Next Entry</a>	\i S N
SD Card	<a href="#">Delete File</a>	\i S K filename_char1 filename_char2 ... null
SD Card	<a href="#">Open File</a>	\i S O handle mode filename_char1 filename_char2 ... null
SD Card	<a href="#">Close File</a>	\i S C handle
SD Card	<a href="#">Tell Position in File</a>	\i S P handle
SD Card	<a href="#">Set Position in File</a>	\i S S handle position
SD Card	<a href="#">Read File</a>	\i S R handle length
SD Card	<a href="#">Read String from File</a>	\i S r handle endchar
SD Card	<a href="#">Write File</a>	\i S W handle length data_b0 data_b1 ...
SD Card	<a href="#">Write String to File</a>	\i S w handle char1 char2 ... null
SD Card	<a href="#">Truncate File</a>	\i S T handle
SD Card	<a href="#">Write Application Data to Flash</a>	\i S A flags filename_char1 filename_char2 ... null
SD Card	<a href="#">Unmount MicroSD Card</a>	\i S U
Time/Date	<a href="#">Set Time</a>	\i I T hour minute second
Time/Date	<a href="#">Get Time</a>	\i I ? T
Time/Date	<a href="#">Set Date</a>	\i I D year month day weekday
Time/Date	<a href="#">Get Date</a>	\i I ? D
PWM	<a href="#">Set PWM #0</a>	\i I P 00h freq_b3 freq_b2 freq_b1 freq_b0 duty_cycle_hb duty_cycle_lb
PWM	<a href="#">Set PWM #1</a>	\i I P 01h duty_cycle_hb duty_cycle_lb
EEPROM	<a href="#">Get EEPROM Size</a>	\i E ?
EEPROM	<a href="#">Erase EEPROM</a>	\i E E =
EEPROM	<a href="#">Read EEPROM</a>	\i E R index_hb index_lb
EEPROM	<a href="#">Write EEPROM</a>	\i E W index_hb index_lb
EEPROM	<a href="#">Set PCAP Configuration to Factory Default</a>	\i E P D

Power/Watch Dog	<a href="#">Set Watchdog Interval</a>	\i P W interval_hb interval_lb
Power/Watch Dog	<a href="#">Feed Watchdog</a>	\i P w
Power/Watch Dog	<a href="#">Shutdown (Power Off)</a>	\i P U =
Power/Watch Dog	<a href="#">Hard Shutdown (Long Power Off)</a>	\i P u =
Power/Watch Dog	<a href="#">Cancel Shutdown</a>	\i P U C
Power/Watch Dog	<a href="#">Get Power State</a>	\i P ?
Power/Watch Dog	<a href="#">Reset Motherboard</a>	\i P ! =
Power/Watch Dog	<a href="#">Set Smart Power-Off Mode</a>	\i P S on_off
Power/Watch Dog	<a href="#">Set Power-Off Notification On/Off</a>	\i P N on_off
Extra	<a href="#">Set Message Offset</a>	\i X O T offset_hb offset_lb
Extra	<a href="#">Set Graphic Offset</a>	\i X O G offset_hb offset_lb
Extra	<a href="#">Set Macro Offset</a>	\i X O O offset_hb offset_lb
Extra	<a href="#">Set Font Offset</a>	\i X O F offset_hb offset_lb
Extra	<a href="#">Set Message Name Prefix</a>	\i X P T char1 char2 ... null
Extra	<a href="#">Set Graphic Name Prefix</a>	\i X P G char1 char2 ... null
Extra	<a href="#">Set Macro Name Prefix</a>	\i X P O char1 char2 ... null
Extra	<a href="#">Set Font Name Prefix</a>	\i X P F char1 char2 ... null
Extra	<a href="#">Set Message Name Suffix</a>	\i X S T char1 char2 ... null
Extra	<a href="#">Set Graphic Name Suffix</a>	\i X S G char1 char2 ... null
Extra	<a href="#">Set Macro Name Suffix</a>	\i X S O char1 char2 ... null
Extra	<a href="#">Set Font Name Suffix</a>	\i X S F char1 char2 ... null

## Commands Sorted by Command Sequence

The depiction of commands in this list differs from the detailed descriptions in chapter [Command Description](#), where the syntax complies with the notation in iLCD Manager XE. Here, every character resp. word between two spaces represents a byte sent to the iLCD controller (e.g. from the controlling application).

- the sequence `\i` describes the command introducer (0xAA)
- single characters are represented by their corresponding ASCII codes
- strings are depicted as `char1 char2 ... null`

- [Word \(16-Bit\) Values](#) ("word" in descriptions) are written as  $xxx\_hb \ xxx\_lb$  (meaning  $xxx\_hb * 256 + xxx\_lb$ )
- [Long \(32-Bit\) Values](#) ("long" in descriptions ) are written as  $xxx\_b3 \ xxx\_b2 \ xxx\_b1 \ xxx\_b0$  (meaning  $value\_b3 * 16,777,216 + value\_b2 * 65,536 + value\_b1 * 256 + value\_b0$ )

Section	Command Description	Command
General	<a href="#">No Operation</a>	\i '
General	<a href="#">Reset All</a>	\i !
General	<a href="#">Reboot Panel Controller</a>	\i #
General	<a href="#">Reset All and Show Startup Graphic</a>	\i \$
General	<a href="#">Get iLCD Controller Name</a>	\i ? C
General	<a href="#">Get Last Error Code</a>	\i ? E
General	<a href="#">Get Hardware Revision</a>	\i ? H
General	<a href="#">Get Firmware Info</a>	\i ? I
General	<a href="#">Get Identification Info</a>	\i ? M
General	<a href="#">Get Network Status</a>	\i ? N
General	<a href="#">Get Project Info</a>	\i ? P
General	<a href="#">Get Input Buffer Size</a>	\i ? Q
General	<a href="#">Get Serial Number</a>	\i ? S
General	<a href="#">Get Touch Screen Type</a>	\i ? TM
General	<a href="#">Get Firmware Version</a>	\i ? V
LCD-Attributes	<a href="#">Set Alpha</a>	\i A A A alpha
LCD-Attributes	<a href="#">Set Brightness Adjustment</a>	\i A A B brightness_hb brightness_lb
LCD-Attributes	<a href="#">Set Contrast Adjustment</a>	\i A A C contrast_hb contrast_lb
LCD-Attributes	<a href="#">Set Adjustment for Graphics</a>	\i A A G mode
LCD-Attributes	<a href="#">Set Hue Adjustment</a>	\i A A H hue_hb hue_lb
LCD-Attributes	<a href="#">Set Saturation Adjustment</a>	\i A A S saturation_hb saturation_lb
LCD-Attributes	<a href="#">Set Bold Mode</a>	\i A B on_off
LCD-Attributes	<a href="#">Set Foreground Color</a>	\i A C F red green blue
LCD-Attributes	<a href="#">Set Background Color</a>	\i A C B red green blue
LCD-Attributes	<a href="#">Set Border Color</a>	\i A C R red green blue
LCD-Attributes	<a href="#">Set Border Shadow Color</a>	\i A C S red green blue
LCD-Attributes	<a href="#">Set Font</a>	\i A F number_hb number_lb
LCD-Attributes	<a href="#">Set Rectangle Corner Radius</a>	\i A H R word::radius_hb radius_lb

LCD-Attributes	<a href="#">Set Shadow Offset</a>	\i A H S byte::x_offset byte::y_offset
LCD-Attributes	<a href="#">Set Inverse Mode</a>	\i A I on_off
LCD-Attributes	<a href="#">Set Line Caps Style</a>	\i A L C style
LCD-Attributes	<a href="#">Set Line Ending Mode</a>	\i A L M mode
LCD-Attributes	<a href="#">Set Line Thickness</a>	\i A L T thickness
LCD-Attributes	<a href="#">Set Filling Color</a>	\i A P C red green blue
LCD-Attributes	<a href="#">Set Filling Gradient</a>	\i A P G mode ramp_hb ramp_lb from_color_red from_color_green from_color_blue to_color_red to_color_green to_color_blue
LCD-Attributes	<a href="#">Set Filling Tile</a>	\i A P T graphic_index_hb graphic_index_lb
LCD-Attributes	<a href="#">Set Font Spacing</a>	\i A S x_spacing y_spacing
LCD-Attributes	<a href="#">Set Transparent Mode On/Off</a>	\i A T on_off
LCD-Attributes	<a href="#">Set Underline Mode</a>	\i A U on_off
LCD-Attributes	<a href="#">Set Underline Position</a>	\i A u position
LCD-Attributes	<a href="#">Set Symbol Font</a>	\i A Y on_off
LCD-Control	<a href="#">Get Backlight Mode</a>	\i C ? B
LCD-Control	<a href="#">Get Display Size</a>	\i C ? D
LCD-Control	<a href="#">Get Graphic Info</a>	\i C ? G graph_idx_hb graph_idx_lb
LCD-Control	<a href="#">Get Backlight Intensity</a>	\i C ? I
LCD-Control	<a href="#">Get Backlight Intensity (High-Res)</a>	\i C ? i
LCD-Control	<a href="#">Get Cursor Position</a>	\i C ? K
LCD-Control	<a href="#">Get LCD Gamma Value</a>	\i C ? M
LCD-Control	<a href="#">Get LCD Contrast</a>	\i C ? N
LCD-Control	<a href="#">Get Text Extent</a>	\i C ? T char1 char2 ... null
LCD-Draw	<a href="#">Get Unicode Text Extent</a>	\i C ? U char1_hb char1_lb char2_hb char2_lb ... null null
LCD-Control	<a href="#">Get Text Message Extent</a>	\i C ? t index_hb index_lb
LCD-Control	<a href="#">Get Fixed LCD Contrast/Gamma</a>	\i C ? X
LCD-Control	<a href="#">Enable/Disable ANSI</a>	\i C A on_off
LCD-Control	<a href="#">Set Backlight Blink Frequency</a>	\i C b period
LCD-Control	<a href="#">Set Backlight Mode</a>	\i C B mode
LCD-Control	<a href="#">Increment/Decrement Column Address</a>	\i C c addr_inc_hb addr_inc_lb



LCD-Control	<a href="#">Set Column Address</a>	\i C C address_hb address_lb
LCD-Control	<a href="#">Turn Display On/Off</a>	\i C D mode
LCD-Control	<a href="#">Set Column Coordinates Scaling</a>	\i C E C mult div
LCD-Control	<a href="#">Set Fonts Scaling</a>	\i C E F factor
LCD-Control	<a href="#">Set Graphics Scaling</a>	\i C E G factor
LCD-Control	<a href="#">Set Row Coordinates Scaling</a>	\i C E R mult div
LCD-Control	<a href="#">Set Auto-Linefeed</a>	\i C F on_off
LCD-Control	<a href="#">Go Terminal Mode</a>	\i C G =
LCD-Control	<a href="#">Set Graphic Alignment</a>	\i C G A mode width_hb width_lb height_hb height_lb
LCD-Control	<a href="#">Set Backlight Intensity</a>	\i C I intensity
LCD-Control	<a href="#">Set Backlight Intensity (High-Res)</a>	\i C i intensity
LCD-Control	<a href="#">Set Cursor Position</a>	\i C K x_hb x_lb y_hb y_lb
LCD-Control	<a href="#">Set Relative Cursor Position</a>	\i C K x_inc_hb x_inc_lb y_inc_hb y_inc_lb
LCD-Control	<a href="#">Set Line Style</a>	\i C L style
LCD-Control	<a href="#">Set LCD Gamma Value</a>	\i C M value
LCD-Control	<a href="#">Set LCD Contrast</a>	\i C N value
LCD-Control	<a href="#">Set Screen Orientation</a>	\i C O orientation
LCD-Control	<a href="#">Set Text/Graphic Orientation</a>	\i C o orientation
LCD-Control	<a href="#">Increment/Decrement Row Address</a>	\i C r addr_inc_hb addr_inc_lb
LCD-Control	<a href="#">Set Row Address</a>	\i C R address_hb address_lb
LCD-Control	<a href="#">Set TAB Spacing</a>	\i C S tab_spacing
LCD-Control	<a href="#">Set Text Alignment</a>	\i C T mode width_hb width_lb height_hb height_lb
Viewports	<a href="#">Copy Viewport</a>	\i C V C screen viewport copy_content pos_x_hb pos_x_lb pos_y_hb pos_y_lb
Viewports	<a href="#">Define Viewport</a>	\i C V D viewport orientation width_hb width_lb height_hb height_lb
Viewports	<a href="#">Select Viewport</a>	\i C V S viewport
LCD-Control	<a href="#">Set Wrap Mode</a>	\i C W horz_wrap vert_wrap
LCD-Control	<a href="#">Set XON/XOFF for Terminal Mode</a>	\i C X on_off
LCD-Draw	<a href="#">Adjust Display</a>	\i D A mode

LCD-Draw	<a href="#">Adjust Display Area</a>	\i D a mode width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Draw Circle</a>	\i D C radius_hb radius_lb
LCD-Draw	<a href="#">Draw Styled Circle</a>	\i D c mode radius_hb radius_lb
LCD-Draw	<a href="#">Draw Dot</a>	\i D D red green blue
LCD-Draw	<a href="#">Draw Dot at X/Y</a>	\i D d x_pos_hb x_pos_lb y_pos_hb y_pos_lb red green blue
LCD-Draw	<a href="#">Erase Display</a>	\i D E
LCD-Draw	<a href="#">Erase Display Area</a>	\i D e width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Fill Display</a>	\i D F
LCD-Draw	<a href="#">Fill Display Area</a>	\i D f width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Invert Display</a>	\i D I
LCD-Draw	<a href="#">Invert Display Area</a>	\i D i width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Draw Line</a>	\i D L end_x_hb end_x_lb end_y_hb end_y_lb
LCD-Draw	<a href="#">Write Scan Line and Advance</a>	\i D N A no_of_pixels_hb no_of_pixels_lb p0_lb p0_hb p1_lb p1_hb ...
LCD-Draw	<a href="#">Read Scan Line</a>	\i D N R no_of_pixels_hb no_of_pixels_lb
LCD-Draw	<a href="#">Read 1D/2D Run-Length Encoded Scan Line</a>	\i D N r prev_line_offset no_of_pixels_hb no_of_pixels_lb
LCD-Draw	<a href="#">Write Scan Line</a>	\i D N W no_of_pixels_hb no_of_pixels_lb p0_lb p0_hb p1_lb p1_hb ...
LCD-Draw	<a href="#">Write 1D/2D Run-Length Encoded Scan Line</a>	\i D N w prev_line_offset no_of_pixels_hb no_of_pixels_lb no_of_rle_bytes_hb no_of_rle_bytes_lb b0 b1 ...
LCD-Draw	<a href="#">Set/Clear Pixel</a>	\i D P on_off
LCD-Draw	<a href="#">Set/Clear Pixel at X/Y</a>	\i D p x_pos_hb x_pos_lb y_pos_hb y_pos_lb on_off
LCD-Draw	<a href="#">Draw Rectangle</a>	\i D R mode width_hb width_lb height_hb height_lb
LCD-Draw	<a href="#">Scroll Down</a>	\i D S D scroll_y_hb scroll_y_lb
LCD-Draw	<a href="#">Scroll Left</a>	\i D S L scroll_x_hb scroll_x_lb
LCD-Draw	<a href="#">Scroll Right</a>	\i D S R scroll_x_hb scroll_x_lb
LCD-Draw	<a href="#">Scroll Up</a>	\i D S U scroll_y_hb scroll_y_lb
LCD-Draw	<a href="#">Write Text</a>	\i D T char1 char2 ... null

LCD-Draw	<a href="#">Write Text Message</a>	\i D t index_hb index_lb
LCD-Draw	<a href="#">Write Unicode Text</a>	\i D U char1_hb char1_lb char2_hb char2_lb ... null null
LCD-Draw	<a href="#">Draw Ellipse</a>	\i D Y bits::mode word::vertex_a_hb word::vertex_a_lb word::vertex_b_hb word::vertex_b_lb
EEPROM	<a href="#">Get EEPROM Size</a>	\i E ?
EEPROM	<a href="#">Erase EEPROM</a>	\i E E =
EEPROM	<a href="#">Set PCAP Configuration to Factory Default</a>	\i E P D
EEPROM	<a href="#">Read EEPROM</a>	\i E R index_hb index_lb
EEPROM	<a href="#">Write EEPROM</a>	\i E W index_hb index_lb
LCD-Graphics	<a href="#">Display Graphic Area</a>	\i g A x_hb x_lb y_hb y_lb width_hb width_lb height_hb height_lb graph_idx_hb graph_idx_lb
LCD-Graphics	<a href="#">Stop (Break) Animation</a>	\i g B anim_loc
LCD-Graphics	<a href="#">Set Animation Background Color</a>	\i g b C anim_loc red green blue
LCD-Graphics	<a href="#">Set Animation Background Frame</a>	\i g b F anim_loc frame_hb frame_lb
LCD-Graphics	<a href="#">Set Animation Background Graphic</a>	\i g b G anim_loc bg_offset_x_hb bg_offset_x_lb bg_offset_y_hb bg_offset_y_lb graphic_idx_hb graphic_idx_lb
LCD-Graphics	<a href="#">Remove Animation Background</a>	\i g b N anim_loc
LCD-Graphics	<a href="#">Set Animation Background Screen</a>	\i g b S anim_loc bg_offset_x_hb bg_offset_x_lb bg_offset_y_hb bg_offset_y_lb screen
LCD-Graphics	<a href="#">Erase Animation Image Area</a>	\i g E anim_loc
LCD-Graphics	<a href="#">Erase Animation Frame Area</a>	\i g e anim_loc
LCD-Graphics	<a href="#">Stop Animation and Set Frame Number</a>	\i g F anim_loc frame_hb frame_lb
LCD-Graphics	<a href="#">Display Local Graphic</a>	\i g G graph_idx_hb graph_idx_lb
LCD-Graphics	<a href="#">Set Animation Coordinates to Cursor Position</a>	\i g k anim_loc
LCD-Graphics	<a href="#">Set Animation Coordinates to X/Y</a>	\i g K anim_loc pos_x_hb pos_x_lb pos_y_hb pos_y_lb
LCD-Graphics	<a href="#">Load Animated Graphics</a>	\i g L anim_loc index_hb index_lb
LCD-Graphics	<a href="#">Move Animation To Frame</a>	\i g M anim_loc frame_idx_hb frame_idx_lb backwards

LCD-Graphics	<a href="#">Resume Animation Engine</a>	\i g r
LCD-Graphics	<a href="#">Set Animation Repetitions</a>	\i g R anim_loc repeat_hb repeat_lb
LCD-Graphics	<a href="#">Suspend Animation Engine</a>	\i g s
LCD-Graphics	<a href="#">Start or Restart Animation</a>	\i g S anim_loc
Input/Output	<a href="#">Get ADC Value</a>	\i I ? A port
Input/Output	<a href="#">Get Current Communication-Port</a>	\i I ? C
Time/Date	<a href="#">Get Date</a>	\i I ? D
Input/Output	<a href="#">Get Inputs State</a>	\i I ? I
Time/Date	<a href="#">Get Time</a>	\i I ? T
Input/Output	<a href="#">Set Baud Rate</a>	\i I B port_no baud_b3 baud_b2 baud_b1 baud_b0
Input/Output	<a href="#">Get Enabled Communication-Ports</a>	\i I C ?
Input/Output	<a href="#">Disable Communication-Ports</a>	\i I C D port_mask timeout_hb timeout_lb
Time/Date	<a href="#">Set Date</a>	\i I D year month day weekday
Input/Output	<a href="#">Enable/Disable Rotary Encoder Reporting</a>	\i I E on_off
Input/Output	<a href="#">Get Keyboard State</a>	\i I K ?
Input/Output	<a href="#">Enable/Disable Keyboard</a>	\i I K E on_off
Input/Output	<a href="#">Enable/Disable Keyboard Reporting</a>	\i I K R on_off
Input/Output	<a href="#">Set Output Blink Frequency</a>	\i I L F period
Input/Output	<a href="#">Set Multiple Outputs</a>	\i I L s out_mask_hb out_mask_lb blink_mask_hb blink_mask_lb
Input/Output	<a href="#">Set Output</a>	\i I L S out_no mode
PWM	<a href="#">Set PWM #0</a>	\i I P 00h freq_b3 freq_b2 freq_b1 freq_b0 duty_cycle_hb duty_cycle_lb
PWM	<a href="#">Set PWM #1</a>	\i I P 01h duty_cycle_hb duty_cycle_lb
Input/Output	<a href="#">Set Relays On/Off/PWM</a>	\i I R relay_no mode
Input/Output	<a href="#">Relays One Shot</a>	\i I r relay_no mode time_hb time_lb
Time/Date	<a href="#">Set Time</a>	\i I T hour minute second
LCD-Screen Memory	<a href="#">Get Draw Screen Parameters</a>	\i M ? D
LCD-Screen Memory	<a href="#">Get View Screen Parameters</a>	\i M ? V
LCD-Screen Memory	<a href="#">Copy Screen Area</a>	\i M A width_hb width_lb height_hb height_lb index x_hb x_lb y_hb y_lb

LCD-Cursor Memory	<a href="#">Restore Cursor &amp; Attributes from Memory</a>	\i M C C index
LCD-Cursor Memory	<a href="#">Save Cursor &amp; Attributes to Memory</a>	\i M C S index
LCD-Screen Memory	<a href="#">Set Draw Screen</a>	\i M D index
LCD-Screen Memory	<a href="#">Get # of Screens</a>	\i M S ?
LCD-Screen Memory	<a href="#">Copy Screen From</a>	\i M S C index
LCD-Screen Memory	<a href="#">Scroll Down Screen</a>	\i M S D index scroll_y_hb scroll_y_lb
LCD-Screen Memory	<a href="#">Set Height of Screen</a>	\i M S H index height_hb height_lb
LCD-Screen Memory	<a href="#">Invert Screen</a>	\i M S I index
LCD-Screen Memory	<a href="#">Scroll Left Screen</a>	\i M S L index scroll_x_hb scroll_x_lb
LCD-Screen Memory	<a href="#">Paint Screen From</a>	\i M S P index
LCD-Screen Memory	<a href="#">Scroll Right Screen</a>	\i M S R index scroll_x_hb scroll_x_lb
LCD-Screen Memory	<a href="#">Copy Screen To</a>	\i M S S index
LCD-Screen Memory	<a href="#">Scroll Up Screen</a>	\i M S U index scroll_y_hb scroll_y_lb
LCD-Screen Memory	<a href="#">Set Width of Screen</a>	\i M S W index width_hb width_lb
LCD-Screen Memory	<a href="#">Set View Screen</a>	\i M V index
Macros	<a href="#">Delay Macro Execution</a>	\i O D delay_hb delay_lb
Macros	<a href="#">Execute Macro</a>	\i O E index_hb index_lb
Macros	<a href="#">Execute Protected Macro</a>	\i O e index_hb index_lb
Macros	<a href="#">Jump to Macro</a>	\i O J index_hb index_lb
Macros	<a href="#">Allow Keyboard/Touch Events to be Processed</a>	\i O K
Macros	<a href="#">Set Macro Execution Speed</a>	\i O S speed_hb speed_lb
Macros	<a href="#">Set Macro Timer</a>	\i O T time
Power/Watch Dog	<a href="#">Reset Motherboard</a>	\i P ! =
Power/Watch Dog	<a href="#">Get Power State</a>	\i P ?

Power/Watch Dog	<a href="#">Set Power-Off Notification On/Off</a>	\i P N on_off
Power/Watch Dog	<a href="#">Set Smart Power-Off Mode</a>	\i P S on_off
Power/Watch Dog	<a href="#">Shutdown (Power Off)</a>	\i P U =
Power/Watch Dog	<a href="#">Hard Shutdown (Long Power Off)</a>	\i P u =
Power/Watch Dog	<a href="#">Cancel Shutdown</a>	\i P U C
Power/Watch Dog	<a href="#">Feed Watchdog</a>	\i P w
Power/Watch Dog	<a href="#">Set Watchdog Interval</a>	\i P W interval_hb interval_lb
SD Card	<a href="#">Request Disk Status</a>	\i S ?
SD Card	<a href="#">Write Application Data to Flash</a>	\i S A flags filename_char1 filename_char2 ... null
SD Card	<a href="#">Close File</a>	\i S C handle
SD Card	<a href="#">Open Directory and Read First Entry</a>	\i S F dirname_char1 dirname_char2 ... null
SD Card	<a href="#">Get File Status</a>	\i S G filename_char1 filename_char2 ... null
SD Card	<a href="#">Delete File</a>	\i S K filename_char1 filename_char2 ... null
SD Card	<a href="#">Make Directory</a>	\i S M dirname_char1 dirname_char2 ... null
SD Card	<a href="#">Read Next Entry</a>	\i S N
SD Card	<a href="#">Open File</a>	\i S O handle mode filename_char1 filename_char2 ... null
SD Card	<a href="#">Tell Position in File</a>	\i S P handle
SD Card	<a href="#">Read String from File</a>	\i S r handle endchar
SD Card	<a href="#">Read File</a>	\i S R handle length
SD Card	<a href="#">Set Position in File</a>	\i S S handle position
SD Card	<a href="#">Truncate File</a>	\i S T handle
SD Card	<a href="#">Unmount MicroSD Card</a>	\i S U
SD Card	<a href="#">Write String to File</a>	\i S w handle char1 char2 ... null
SD Card	<a href="#">Write File</a>	\i S W handle length data_b0 data_b1 ...
Touch Screen	<a href="#">Retrieve Last Touch Screen Event</a>	\i T ?
Touch Screen	<a href="#">Create/Define Touch Field</a>	\i T A field_idx key

Touch Screen	<a href="#">Set Touch Field Break Macro</a>	\i T B macro_idx_hb macro_idx_lb
Touch Screen	<a href="#">Calibrate Touch Screen</a>	\i T C
Touch Screen	<a href="#">Calibrate Touch Screen and Report</a>	\i T c
Touch Screen	<a href="#">Draw Touch Field Text Message</a>	\i T D field_idx
Touch Screen	<a href="#">Enable/Disable Automatic Touch Macro Executing</a>	\i T E A on_off
Touch Screen	<a href="#">Execute Touch Break Macro</a>	\i T E B field_idx
Touch Screen	<a href="#">Set Cursor to Touch Field</a>	\i T E C field_idx
Touch Screen	<a href="#">Execute Touch Make Macro</a>	\i T E M field_idx
Touch Screen	<a href="#">Global Enable/Disable Touch Fields</a>	\i T G on_off
Touch Screen	<a href="#">Set Touch Field Height</a>	\i T H height_hb height_lb
Touch Screen	<a href="#">Set Current Touch Field Index</a>	\i T I field_idx
Touch Screen	<a href="#">Enable/Disable Reporting Touch-Coordinates</a>	\i T K on_off
Touch Screen	<a href="#">Set Touch Field Make Macro</a>	\i T M macro_idx_hb macro_idx_lb
Touch Screen	<a href="#">Enable/Disable Reporting Movements</a>	\i T O on_off
Touch Screen	<a href="#">Enable/Disable Touch Field Reporting</a>	\i T P on_off
Touch Screen	<a href="#">Enable/Disable Touch Field Queue</a>	\i T Q on_off
Touch Screen	<a href="#">Remove Touch Field</a>	\i T R field_idx
Touch Screen	<a href="#">Set Threshold for Movement Reporting</a>	\i T S M threshold
Touch Screen	<a href="#">Set Number of Touch Fingers</a>	\i T S N fingers
Touch Screen	<a href="#">Set Touch Field Text Message</a>	\i T T text_message_idx_hb text_message_idx_lb
Touch Screen	<a href="#">Verify Touch Screen Calibration</a>	\i T V
Touch Screen	<a href="#">Start Demonstration Mode</a>	\i T v mode timeout_hb timeout_lb macro_index_hb macro_index_lb
Touch Screen	<a href="#">Set Touch Field Width</a>	\i T W width_hb width_lb
Extra	<a href="#">Set Font Offset</a>	\i X O F offset_hb offset_lb

Extra	<a href="#">Set Graphic Offset</a>	\i X O G offset_hb offset_lb
Extra	<a href="#">Set Macro Offset</a>	\i X O O offset_hb offset_lb
Extra	<a href="#">Set Message Offset</a>	\i X O T offset_hb offset_lb
Extra	<a href="#">Set Font Name Prefix</a>	\i X P F char1 char2 ... null
Extra	<a href="#">Set Graphic Name Prefix</a>	\i X P G char1 char2 ... null
Extra	<a href="#">Set Macro Name Prefix</a>	\i X P O char1 char2 ... null
Extra	<a href="#">Set Message Name Prefix</a>	\i X P T char1 char2 ... null
Extra	<a href="#">Set Font Name Suffix</a>	\i X S F char1 char2 ... null
Extra	<a href="#">Set Graphic Name Suffix</a>	\i X S G char1 char2 ... null
Extra	<a href="#">Set Macro Name Suffix</a>	\i X S O char1 char2 ... null
Extra	<a href="#">Set Message Name Suffix</a>	\i X S T char1 char2 ... null

## Command Description

In this chapter, all commands are described in detail. Find following sections in this chapter:

- [General Commands](#)
- [Viewport Related Commands](#)
- [LCD Control Commands](#)
- [LCD Attribute Commands](#)
- [LCD Draw Commands](#)
- [LCD Graphics Commands](#)
- [Screen Memory Related Commands](#)
- [Cursor Memory Related Commands](#)
- [Macro Related Commands](#)
- [Touch Screen Related Commands](#)
- [Input/Output Related Commands](#)
- [MicroSD Card Related Commands](#)
- [Time/Date Related Commands](#)
- [Pulse Width Modulation \(PWM\) Related Commands](#)
- [EEPROM Related Commands](#)
- [Power/Watchdog Related Commands](#)
- [Extra Commands](#)

## General Commands

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [No Operation](#)
- [Reset All](#)
- [Reset All and Show Startup Graphic](#)
- [Reboot Panel Controller](#)



- [Get Last Error Code](#)
  - [Get Firmware Info](#)
  - [Get Identification Info](#)
  - [Get Firmware Version](#)
  - [Get Serial Number](#)
  - [Get iLCD Controller Name](#)
  - [Get Hardware Revision](#)
  - [Get Network Status](#)
  - [Get Project Info](#)
  - [Get Touch Screen Type](#)
  - [Get PCAP Controller information](#)
  - [Is Reporting Touch-Coordinates Enabled](#)
  - [Can Run Java](#)
- 

## **No Operation**

`\i`

Sending this sequence does not cause any action, but terminates any incomplete command (as the command introducer (`\i`) is used to synchronize the state machine again) and sends an `[ACK]` character.

### **Note**

- This command may be used to check the presence of an LCD panel on the selected serial port.

**Response:** `[ACK]`

**See also:**

---

## **Reset All**

`\i !`

Clears entire screen area and sets default values for all attributes:

- Cursor is set to 0, 0
- The screen orientation is set to the default value set by iLCD Manager XE.
- All touch fields are removed, report touch field coordinates and movements is disabled
- Text alignment is reset
- Bold, inverse and underline attributes are set to off
- Color values are set to default
- Font is set to startup font
- Underline position is set to 0
- All animations are stopped
- Animation engine is started
- LCD contrast is set to the corresponding EEPROM value
- LCD Gamma value is set to the corresponding EEPROM value

- Backlight intensity is set to the corresponding EEPROM value
- The backlight state is set to the corresponding EEPROM value
- Watchdog is disabled
- Smart power-off is switched off
- Power-off notification set to default
- Power/Watchdog related pins are released
- Shutdown is cancelled
- ANSI mode is set to default
- Horizontal and vertical wrapping mode is set to default
- Auto-Linefeed is set to default
- TAB spacing is set to default
- Backlight blink frequency is set to default
- XON/XOFF mode is set to default
- Keyboard is enabled/disabled according to default value
- Keyboard reporting is turned on/off according to default value
- Relay outputs are set to default
- Output blink frequencies are set to default
- All outputs are reset to their startup states
- All open files on the MicroSD card are closed
- The main screen is selected as draw and view screen
- All viewports are removed
- All scale factors are set to 1
- All line attributes are set to default
- The Macro Timer is set 0

Only if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear" (this setting is ignored in firmware versions < 3.02):

- All offsets are set to 0
- All prefixes and suffixes are set to empty strings

**Response:** [ACK]

**See also:**

[Reset All and Show Startup Graphic](#)

---

## **Reset All and Show Startup Graphic**

```
\i $
```

This command calls the [Reset All](#) command and then displays the startup graphic on the position defined via iLCD Manager XE.

**Response:** [ACK]

**See also:**

[Reset All](#)

---

## Reboot Panel Controller

```
\i #
```

The panel controller is rebooted which gives the same effect as a hard reset.

### Note

- After the controller is rebooted it sends the startup message (see [Startup](#)) to the default communications port, if sending the startup message is activated on the "Settings" page of iLCD Manager XE.
- The baud rate for serial ports will be reset to the default value set on the "Settings" page of iLCD Manager XE. The `[ACK]` response is sent with the current baud rate, the startup message with the default baud rate.

**Response:** `[ACK]`

### See also:

[Hard Shutdown \(Long Power Off\)](#)  
[Set Baud Rate](#)

## Get Last Error Code

```
\i ? E
```

Returns the error code of the last executed command.

### Note

- When a command is successfully executed, the error code is set to 0x0. Therefore, retrieving the error codes only makes sense immediately after a command returning `[NACK]`.
- Refer to [Error Codes](#) for a complete list of possible errors.

**Response:** `[ACK]`  
`string::error`  
`[ACK]`

Parameter	Type	Range	Description
error	<a href="#">word</a>	0x0 ... 0x4BA	error code

### Example

```
\iAF\D24  

\i?E
```

## Response Example

```
[NACK] [ACK] [03] [F1] [ACK]
```

This example returns `[NACK]` as a response to the [Set Font](#) command and the error code 1009 (0x03F1) which means "Font not found" (see [Error Codes](#)).

### See also:

[Error Codes](#)

## Get Firmware Info

```
\i ? I
```

Returns the firmware info string.

```
Response: [ACK]
          string::info
          [ACK]
```

Parameter	Type	Description
info	<a href="#">string</a>	firmware information

## Response Example

```
[ACK]iLCD Firmware V2.20 (c) by demmel products 2003-2011[ACK]
```

The iLCD panel is running firmware version 2.20.

### Note

### See also:

[Get Firmware Version](#)  
[Get Identification Info](#)  
[Get iLCD Controller Name](#)

## Get Identification Info

```
\i ? M
```

Returns the identification information.

```
Response: [ACK]
          string::info
          [ACK]
```

Parameter	Type	Description
info	<a href="#">string</a>	identification information

### Response Example

```
[ACK]iLCD Firmware[ACK]
```

The identification information is: iLCD Firmware.

### Note

#### See also:

[Get Firmware Version](#)

[Get Firmware Info](#)

[Get iLCD Controller Name](#)

## Get Firmware Version

```
\i ? v
```

Returns the firmware version as major and minor version separated by a dot.

**Response**     [ACK]  
:               string::version  
                  [ACK]

Parameter	Type	Description
version	<a href="#">string</a>	firmware version

### Response Example

```
[ACK]2.20[ACK]
```

The firmware version is 2.20.

### Note

#### See also:

[Get Firmware Info](#)

[Get Identification Info](#)

[Get iLCD Controller Name](#)

## Get Serial Number

```
\i ? S
```

Returns a text string containing the unique serial number of the iLCD module.

```
[ACK]
Response: string::serial
[ACK]
```

Parameter	Type	Description
serial	<a href="#">string</a>	serial number

### Response Example

```
[ACK] DH-XADC-00131154001-0566 [ACK]
```

The serial number of the iLCD module is: DH-XADC-00131154001-0566.

### Note

#### See also:

[Get Firmware Info](#)

---

## Get iLCD Controller Name

```
\i ? C
```

Returns the name of the iLCD Controller of the module as text string.

```
[ACK]
Response: string::name
[ACK]
```

Parameter	Type	Description
name	<a href="#">string</a>	controller name

### Response Example

```
[ACK] DPC3080 [ACK]
```

The identification information is: iLCD Firmware.

**Note****See also:**

[Get Firmware Info](#)  
[Get Serial Number](#)

---

**Get Hardware Revision**

[\i ? H](#)

Returns a text string containing the hardware revision of the iLCD panel.

**Response** [ACK]  
 : string::revision  
 ACK]

Parameter	Type	Description
revision	<a href="#">string</a>	hardware revision

**Response Example**

[ACK] 2.0 [ACK]

The hardware revision of the iLCD panel is 2.0.

**Note****See also:**

[Get Firmware Info](#)

---

**Get Network Status**

[\i ? N](#)

Returns a structure of bytes containing the status of the Ethernet interface.

**Response:** [ACK] structure [ACK]

Bytes	Description
0	structure version
1	structure length in bytes

2	TCP/IP enabled (0x00 = off, 0x01 = on)
3 .. 6	device IP address
7 .. 10	subnet mask
11 .. 14	standard gateway
15 .. 18	IP of DHCP server
19 .. 22	access IP address (0.0.0.0 for unrestricted access)
23 .. 26	IP currently connected via TCP (0.0.0.0 if no connection)
27 .. 30	IP of last HTTP access (0.0.0.0 if no connection)
31	password mode (refer to table below)

Mod e	Description
0	no password
1	plain text
2	CRC32
3	MD5

### Response Example

```
[ACK]
[01]
[20]
[01]
[C0] [A8] [0A] [0D]
[FF] [FF] [FF] [00]
[C0] [A8] [0A] [01]
[C0] [A8] [0A] [01]
[00] [00] [00] [00]
[C0] [A8] [0A] [0A]
[C0] [A8] [0A] [0A]
[03]
[ACK]
```

### Note

This response translates to:

- structure version = 1
- structure length = 32 bytes
- TCP/IP on
- device IP address = 192.168.10.13
- subnet mask = 255.255.255.0
- standard gateway = 192.168.10.1
- IP of DHCP server = 192.168.10.1
- access IP address = 0.0.0.0 (every IP can access)
- IP currently connected via TCP = 192.168.10.10



- IP of last HTTP access = 192.168.10.10
- password encrypted with MD5 hash

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[Get Firmware Info](#)

---

## **Get Input Buffer Size**

`\i ? Q`

Returns the size of the serial port's input buffer.

**Response:** [ACK] word::size  
[ACK]

Parameter	Type	Range	Description
size	<a href="#">word</a>	128 ... 1024	input buffer size in bytes

### **Response Example**

`[ACK] [04] [00] [ACK]`

This response means that the input buffer for the serial port has a size of 1024 (0x400) bytes.

**See also:**

[Controlling the iLCD via Serial Port](#)  
[Set XON/XOFF for Terminal Mode](#)

---

## **Get Project Info**

`\i ? P`

Returns a structure of ASCII characters containing information about the iLCD Manager XE project currently stored in the flash memory.

### **Note**

- The project modification date and time are updated when a modified project is either saved, exported or written to flash. When an unchanged project is written to flash, the date and time of the last save is used.

**Response:** [ACK] info [ACK]

Bytes	Description
0 .. 23	project filename (without extension)
24 .. 31	project modification date (ASCII string 'MM-DD-YY')
38 .. 45	project modification time (ASCII string 'HH:MM:SS')
47 .. 78	project description (can be edited on the "Settings" page of iLCD Manager XE)

### Response Example

```
[ACK]
ilcd project[00][00]..[00]
09-04-12
[00][00][00][00][00][00]
11:17:11
[00]
My Project 2.34[00][00]..[00]
[ACK]
```

This response means that:

- the project's filename is "ilcd project.lcdp-flash"
- it was written to flash on September 4th, 2012 at 11:17:11
- the description string contains "My Project 2.34"

### Note

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Get Firmware Info](#)

[Get Graphic Info](#)

## Get Touch Screen Type

`\i ? T M`

Returns the type of the currently mounted touch screen.

**Response:** [ACK] type [ACK]

Paramete	Type	Description
----------	------	-------------

r		
type	<a href="#">byte</a>	type of touch screen

where *type* can adopt to one of the following values:

Mode	Description
0	no touch screen mounted
1	resistive touch screen mounted
2	capacitive touch screen mounted

### Response Example

```
[ACK] [02] [ACK]
```

The used iLCD has a capacitive touch screen mounted.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Touch Screen Related Commands](#)

## Get PCAP Controller Information

```
\i ? T P
```

Returns information regarding the PCAP Controller of the module as text string.

```
[ACK]
Response: string::information
[ACK]
```

Parameter	Type	Description
name	<a href="#">string</a>	controller information

### Response Example

```
[ACK]Manuf.: FocalTech, ChipID: 85, FW: 3, FW-lib.: 48.3[ACK]
```

The command returns manufacturer-specific information. Please refer to the manufacturer for further details.

**Note****See also:**

[Get Firmware Info](#)  
[Get Serial Number](#)

---

**Is Reporting Touch-Coordinates Enabled**

```
\i ? T K
```

Returns true if reporting of touch coordinates is enabled (refer to [Enable/Disable Reporting Touch-Coordinates](#)).

**Response:** [ACK] on\_off [ACK]

Parameter	Type	Description
on_off	<a href="#">bool</a>	reporting is on (1) or off (0)

**Response Example**

```
[ACK] [01] [ACK]
```

When touch reports are sent, the touch coordinates will be appended to the report (refer to [Touch Field Press/Release + Coordinate of Event](#)).

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Touch Screen Related Commands](#)  
[Enable/Disable Reporting Touch-Coordinates](#)  
[Touch Field Press/Release + Coordinate of Event](#)

---

**Can Run Java**

```
\i ? J
```

Checks if Java is supported by the iLCD panel.

**Response:** [ACK] status [ACK]

Parameter	Type	Description
-----------	------	-------------

status	<a href="#">bool</a>	Java supported (1) or not (0)
--------	----------------------	-------------------------------

**See also:**

[Java](#)  
[Start Java Binary](#)  
[Set Java Main Method Arguments](#)

**Is Simulator Running**

```
\i ? m
```

Checks if Simulator is currently running.

**Response:** [ACK] status [ACK]

Parameter	Type	Description
status	<a href="#">bool</a>	Simulator is running (1) or not (0)

**Viewport Related Commands****Concept of Viewports**

Viewports are user-defined screen areas which have their own coordinates, width, height and orientation. Additional to the main viewport 0 (entire main screen), you may define up to 8 viewports (1-8) in total for all screens in a project. They may overlap each other, but consider that they don't have independent memory areas. Therefore, a viewport may overwrite the content of an overlapping other one without saving the underlying area.

All commands (except the [Screen Memory Related Commands](#)) executed within a viewport refer to the viewport's size and cursor position. So, for example, [Get Display Size](#) returns the height and width of the current viewport, not the entire screen. Text wrapping and scrolling as well as cropping of graphics is also done in the selected viewport's area.

All defined viewports have their own set of attributes (see [LCD Attribute Commands](#)). So when switching from viewport 1 to viewport 2 and outputting text in both of them, it shows up on the cursor position and with the colors and attributes of the respective viewport.

**Note**

- The commands [Reset All](#) and [Reset All and Show Startup Graphic](#) delete all defined viewports

**Example**

```
\i! { Reset All }
```

```

\iCK\D30\D130          { Set Cursor Position to 30/130 }
\iCVD\1\0\D65\D50     { Define Viewport 1 with size of
65x50 }

\iCK\D120\D130        { Set Cursor Position to 120/130 }
\iCVD\2\1\D65\D70     { Define Viewport 2 with size of
65x70 }

\iCK\D280\D130        { Set Cursor Position to 280/130 }
\iCVD\3\3\D78\D72     { Define Viewport 3 with size of
78x72 }

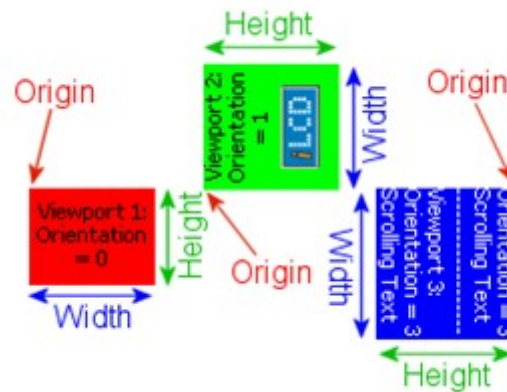
\iCVS\1               { Select Viewport 1 }
\iACB\#FF0000         { Set Background Color to red }
\iDE                  { Fill viewport with background
color }
\iCT\x83\D0\D0        { Center text in viewport }
\iDTVviewport 1: Orientation = 0\0 { Draw Text }

\iCVS\2               { Select Viewport 2 }
\iACB\#00FF00         { Set Background Color to green }
\iDE                  { Fill viewport with background
color }
\iCT\x81\D0\D0        { Center horizontally in viewport }
\iDTVviewport 2: Orientation = 1\0 { Draw Text }

{ ! Cursor Position now relative to Viewport Origin and Orientation: }
\iCK\D11\D40          { Set Cursor Position to X=11 /
Y=40 }
\iG\D14               { Display Local Graphic #14 }

\iCVS\3               { Select Viewport 3 }
\iACB\#0000FF         { Set Background Color to blue }
\iACF\#FFFFFF         { Set Foreground Color to white }
\iDE                  { Fill viewport with background
color }
\iDTVviewport 3:\r    { Draw Text }
Orientation = 3\r
Scrolling Text\r
-----\r
Viewport 3:\r
Orientation = 3\r
Scrolling Text\0

```



Not supported by: DPC3020, DPC2060, DPC10xx

Find following commands in this chapter as well as in the category "LCD Control..." when using the parameter completion feature of iLCD Manager XE:

- [Define Viewport](#)
- [Select Viewport](#)
- [Copy Viewport](#)

## Define Viewport

```
\i C V D byte::viewport byte::orientation word::width word::height
```

Parameter	Type	Range	Description
viewport	<a href="#">byte</a>	1 ... 8	index of the viewport
orientation	<a href="#">byte</a>	0 ... 3	orientation of the viewport
width	<a href="#">word</a>	0 ... display width	width of the viewport
height	<a href="#">word</a>	0 ... display height	height of the viewport

Defines a viewport with origin at the current cursor position.

### Note

- You may define and use up to 8 viewports.
- A viewport is always defined on the currently active draw screen (see [Set Draw Screen](#)).
- The maximum values for *width* and *height* depend on the LCD-Screen used. If one of these values exceeds the screen dimension the viewport will not be defined and a `[NACK]` will be reported.
- The viewport's orientation is interpreted relative to the screen orientation (see [Set Screen Orientation](#)).
- Any other orientation than 0 will rotate the viewport around the cursor position and accordingly swap *width* and *height* (see [Concept of Viewports](#)).

- All defined Viewports are automatically removed on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCVD\1\0\D30\D15
```

This example defines a viewport with index 1, the default orientation and a size of 30x15 pixels.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Concept of Viewports](#)  
[Get Cursor Position](#)  
[Set Screen Orientation](#)  
[Error Codes](#)

## Select Viewport

```
\i C V S byte::viewport
```

Parameter	Type	Range	Description
viewport	<a href="#">byte</a>	0 ... 8	index of the viewport

Selects a specified viewport via its index *viewport*.

### Note

- A *viewport* of 0 represents the main viewport (the entire screen).
- Any other custom viewport can only be selected if it was previously defined with the [Define Viewport](#) command.
- After selecting a viewport, the cursor position and all attributes (see [LCD Attribute Commands](#)) are restored to the state when this viewport was lastly active.
- If *viewport* is other than 0 (main viewport for the active draw screen), the screen the viewport was defined on is activated as the draw screen.

**Response:** [ACK]

### Example

```
\iCVS\1
```

This example selects the viewport defined with index 1.

Not supported by: DPC3020, DPC2060, DPC10xx



**See also:**

[Concept of Viewports](#)  
[Define Viewport](#)

**Copy Viewport**

```
\i C V C byte::screen byte::viewport bool::copy_content word::pos_x  
word::pos_y
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the target screen (M = main screen)
viewport	<a href="#">byte</a>	1 ... 8	index of the new viewport
copy_content	<a href="#">bool</a>	0 ... 1	copy display content (1) or just the parameters (0)
pos_x	<a href="#">word</a>	0 ... display width - 1	x coordinate for the new viewport's origin
pos_y	<a href="#">word</a>	0 ... display height - 1	y coordinate for the new viewport's origin

Creates a new viewport with *viewport* as index at the specified position (*pos\_x* and *pos\_y*) of the selected *screen* and assigns all attributes of the currently selected viewport to it.

**Note**

- Creating a new viewport with this command will also update the target screen's cursor position to the specified origin coordinates.
- The new viewport will inherit the dimensions (width and height) of the source viewport in any case.
- Depending on the boolean value *copy\_content* the new viewport will have the same content (every pixel) as the currently active viewport. If the currently active viewport contains a running animation, only a screenshot of the currently shown frame is copied.
- *screen* can also be set to the currently active draw screen, allowing to copy a viewport to a different position on the same screen. Even if the target screen was never selected as draw screen before, it is possible to select it as view screen after copying a viewport to it.

**Response:** [ACK]

**Example**

```
\iCVC\0\2\1\D45\D60
```

On screen #0, a new viewport with an index of 2 and the same dimensions as the previously selected viewport is created. It has the same content as the selected viewport, its origin is at the coordinates x = 45 and y = 60. The cursor position of screen #0 is also set to this position.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Concept of Viewports](#)

---

## LCD Control Commands

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Turn Display On/Off](#)
  - [Set Screen Orientation](#)
  - [Set Text/Graphic Orientation](#)
  - [Enable/Disable ANSI](#)
  - [Set Column Address](#)
  - [Increment/Decrement Column Address](#)
  - [Set Row Address](#)
  - [Increment/Decrement Row Address](#)
  - [Set Cursor Position](#)
  - [Set Relative Cursor Position](#)
  - [Get Cursor Position](#)
  - [Get Graphic Info](#)
  - [Set Graphic Alignment](#)
  - [Get Text Extent](#)
  - [Get Unicode Text Extent](#)
  - [Get Text Message Extent](#)
  - [Set Text Alignment](#)
  - [Set Line Style](#)
  - [Get Display Size](#)
  - [Set TAB Spacing](#)
  - [Set Auto-Linefeed](#)
  - [Set Wrap Mode](#)
  - [Go Terminal Mode](#)
  - [Set XON/XOFF for Terminal Mode](#)
  - [Set Backlight Mode](#)
  - [Get Backlight Mode](#)
  - [Set Backlight Blink Frequency](#)
  - [Set Backlight Intensity](#)
  - [Set Backlight Intensity \(High-Res\)](#)
  - [Get Backlight Intensity](#)
  - [Get Backlight Intensity \(High-Res\)](#)
  - [Get Fixed LCD Contrast/Gamma](#)
  - [Set LCD Contrast](#)
  - [Get LCD Contrast](#)
  - [Set LCD Gamma Value](#)
  - [Get LCD Gamma Value](#)
  - [Set Graphics Scaling](#)
  - [Set Fonts Scaling](#)
  - [Set Column Coordinates Scaling](#)
  - [Set Row Coordinates Scaling](#)
-

## Turn Display On/Off

`\i C D byte::mode`

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	0 ... 2	display mode

Turns the display on or off. The following modes can be set:

Mode	Description
0	turn off the display and its power supply
1	turn on the display (initialization delay of approximately 300ms)
2	turn on the display and backlight immediately

### Note

- While the display is turned off, the iLCD still processes all commands that are sent. The results show on the screen as soon as it is turned on again.
- When using *mode* 2, the display shows a blank screen for approximately 300ms. To avoid this white screen, set the backlight intensity to 0 (refer to [Set Backlight Intensity](#)) before turning on the display or use mode 1 instead.

**Response:** [ACK]

**See also:**

[Set Backlight Intensity](#)

## Set Screen Orientation

`\i C O byte::orientation`

Parameter	Type	Range	Description
orientation	<a href="#">byte</a>	0 ... 3	screen orientation

Sets the screen orientation to one of the following modes:

Orientation	Description
0	landscape mode (0°)
1	portrait mode (90°)
2	landscape mode upside down (180°)

3	portrait mode upside down (270°)
---	----------------------------------

**Note**

- This command causes a [Reset All](#) to be carried out without setting the default orientation.
- The default value for *orientation* is 0, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**See also:**

[Set Text/Graphic Orientation](#)

**Set Text/Graphic Orientation**

`\i C o byte::orientation`

Parameter	Type	Range	Description
orientation	<a href="#">byte</a>	0 ... 3	screen orientation

Sets the orientation of subsequent text and graphic drawing to one of the following modes:

Orientation	Description
0	landscape mode (0°)
1	portrait mode (90°)
2	landscape mode upside down (180°)
3	portrait mode upside down (270°)

**Note**

- This orientation is interpreted relative to the screen orientation and, if used, viewport orientation (see [Set Screen Orientation](#) and [Define Viewport](#)).
- The default value for *orientation* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When using [Set Text Alignment](#), width and height of the alignment area are interpreted according to the selected text orientation (refer to Example below).

**Response:** [ACK]

**Example**

```
\i!           { Reset All }
\iCL\x55     { Set Line Style to dotted }
```

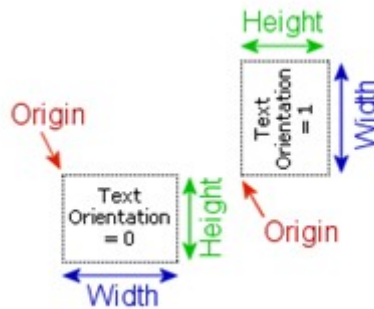
```

\iCK\D50\D130      { Set Cursor Position to Origin X=50 /
Y=130 }
\iCT\x83\D65\D50  { Set Text Alignment to Width=65 /
Height=50 }
\iDR\x0\D65\D50   { Draw Rectangle with width=65 /
height=50 }
\iDTText Orientation = 0\0 { Draw Text "Text Orientation = 0" }

\iCK\D150\D130    { Set Cursor Position to Origin X=150 /
Y=130 }
\iCo\1            { Set Text Orientation to 90° }
\iCT\x83\D65\D50  { Set Text Alignment to Width=65 /
Height=50 }
\iDTText Orientation = 1\0 { Draw Text "Text Orientation = 1" }

{ ! Draw Rectangle is not affected by Text Orientation,
  thus the point of origin is different and width are swapped }
\iCK\D150\D66     { Set Cursor Position to X=150 / Y=66 }
\iDR\x0\D50\D65   { Draw Rectangle with width=50 /
height=65 }

```



### See also:

[Set Screen Orientation](#)  
[Concept of Viewports](#)  
[Set Text Alignment](#)  
[Set Cursor Position](#)  
[Draw Rectangle](#)

### Enable/Disable ANSI

`\i C A bool::on_off`

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	ANSI on (1) or off (0)

Enables or disables ANSI mode.

### Note

- When ANSI support is disabled (*on\_off* = 0), ANSI sequences like carriage returns or escape sequences will not have any effect (see [ANSI Support](#)).
- The default value for *on\_off* is 1, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCA\1
```

Enables the use of ANSI sequences.

**See also:**

[ANSI Support](#)

## Set Column Address

```
\i C C word::address
```

Parameter	Type	Range	Description
address	<a href="#">word</a>	0 ... display width - 1	value for x coordinate

Sets the horizontal cursor position (column address).

### Note

- Using this command changes the x value of the cursor position but not the y value (see [Set Cursor Position](#)).
- Using values greater than the available columns causes a [NACK] to be sent.
- The default value for *address* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCC\D25
```

This example sets the x value of the cursor position to 25. The y value is not affected.

**See also:**

[Set Cursor Position](#)  
[Set Row Address](#)

**Increment/Decrement Column Address**

```
\i C c word::xInc
```

Parameter	Type	Range	Description
xInc	<a href="#">signed word</a>	minus display width - 1 ... display width - 1	amount to increment or decrement the column address

Increments (positive value) or decrements (negative value) the column address (horizontal cursor position) in pixel units relative to the current cursor position.

**Note**

- Using an invalid number for incrementing or decrementing, resulting in setting the cursor beyond the left or right margin of the display, causes a *[NACK]* to be sent.

**Response:** [ACK]

**Example**

```
\iCc\D-40
```

This example will move the cursor position 40 pixels to the left.

**See also:**

[Set Cursor Position](#)  
[Set Relative Cursor Position](#)  
[Set Column Address](#)  
[Set Row Address](#)  
[Increment/Decrement Row Address](#)

**Set Row Address**

```
\i C R word::address
```

Parameter	Type	Range	Description
address	<a href="#">word</a>	0 ... display height - 1	value for an y coordinate

Sets the vertical cursor position (row address).

### Note

- Using this command changes the y value of the cursor position but not the x value (see [Set Cursor Position](#)).
- Using values greater than the available rows causes a `[NACK]` to be sent.
- The default value for `address` is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCR\D50
```

This example sets the y value of the cursor position to 50. The x value is not affected.

### See also:

[Set Cursor Position](#)  
[Set Column Address](#)

## Increment/Decrement Row Address

```
\i C r word::yInc
```

Parameter	Type	Range	Description
yInc	<a href="#">signed word</a>	minus display height - 1 ... display height - 1	amount to increment or decrement the row address

Increments (positive value) or decrements (negative value) the row address (vertical cursor position) in pixel units relative to the current cursor position.

### Note

- Using an invalid number for incrementing or decrementing, resulting in setting the cursor beyond the top or bottom margin of the display, causes a `[NACK]` to be sent.

**Response:** [ACK]

### Example

```
\iCr\D40
```

This example will move the cursor position 40 pixels down.



**See also:**

[Set Cursor Position](#)  
[Set Relative Cursor Position](#)  
[Set Row Address](#)  
[Set Column Address](#)  
[Increment/Decrement Column Address](#)

**Set Cursor Position**

```
\i C K word::pos_x word::pos_y
```

Parameter	Type	Range	Description
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal cursor position
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical cursor position

Sets the cursor to the specified position according to *pos\_x* and *pos\_y* coordinates.

**Note**

- The cursor position is used in conjunction with most drawing commands (refer to "See also:" below).
- This command sets the column address to *pos\_x* and the row address to *pos\_y*.
- The default value for *pos\_x* and *pos\_y* is 0. They will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iCK\D0\D0
```

Sets the cursor to the top-left corner.

**See also:**

[Set Relative Cursor Position](#)  
[Get Cursor Position](#)  
[Set Column Address](#)  
[Set Row Address](#)  
[Get Cursor Position](#)  
[Set/Clear Pixel](#)  
[Display Local Graphic](#)  
[Load Animated Graphics](#)  
[Create/Define Touch Field](#)  
[Draw Dot](#)  
[Draw Line](#)  
[Draw Rectangle](#)  
[Draw Circle](#)

[Draw Styled Circle](#)  
[Draw Ellipse](#)  
[Write Text](#)  
[Save Cursor & Attributes to Memory](#)  
[Define Viewport](#)  
[Erase Display Area](#)  
[Write Scan Line](#)

## Set Relative Cursor Position

```
\i C k word::x_inc word::y_inc
```

Parameter	Type	Range	Description
x_inc	<a href="#">signed word</a>	minus display width - 1 ... display width - 1	amount to increment or decrement the horizontal cursor position
y_inc	<a href="#">signed word</a>	minus display height - 1 ... display height - 1	amount to increment or decrement the vertical cursor position

Moves the cursor relative to the current position by the values *x\_inc* and *y\_inc*.

### Note

- If the resulting cursor position would be beyond the margins of the display, a *[NACK]* is returned and the cursor is not modified.
- The cursor position is used in conjunction with most drawing commands (refer to "See also:" below).
- The default cursor position is 0/0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCk\D-10\D20
```

Moves the cursor 10 pixel to the left and 20 pixel down.

### See also:

[Set Cursor Position](#)  
[Get Cursor Position](#)  
[Set Column Address](#)  
[Set Row Address](#)  
[Get Cursor Position](#)  
[Set/Clear Pixel](#)  
[Display Local Graphic](#)  
[Load Animated Graphics](#)  
[Create/Define Touch Field](#)  
[Draw Dot](#)  
[Draw Line](#)

[Draw Rectangle](#)  
[Draw Circle](#)  
[Draw Styled Circle](#)  
[Draw Ellipse](#)  
[Write Text](#)  
[Save Cursor & Attributes to Memory](#)  
[Define Viewport](#)  
[Erase Display Area](#)  
[Write Scan Line](#)

---

## **Get Cursor Position**

```
\i C ? K
```

Returns the current position of the cursor.

**Response:** [ACK] word::pos\_x word::pos\_y  
[ACK]

Parameter	Type	Range	Description
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal cursor position
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical cursor position

### **Note**

### **Response Example**

```
[ACK] [01] [F4] [00] [25] [ACK]
```

### **Note**

This response means that the current cursor position is at x = 500 (0x1F4), y = 37 (0x25).

### **See also:**

[Set Cursor Position](#)

---

## **Get Graphic Info**

### **by index:**

```
\i C ? G word::graphic_index
```

Parameter	Type	Range	Description
graphic_index	<a href="#">word</a>	0 ... max. graphic index	index of the graphic

**by name:**

```
\i C ? G by_name::graphic_name
```

Parameter	Type	Range	Description
graphic_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated graphic name

**by filename:**

```
\iC ? G file::graphic_filename
```

Parameter	Type	Range	Description
graphic_filename	<a href="#">by_file</a>	DOS filename (8.3 format)	name and path of the graphics file

Returns a structure of bytes containing information about the specified graphic.

**Response:** [ACK] info [ACK]

Bytes	Description
0 ... 1	width of the graphic
2 ... 3	height of the graphic
4	graphic properties (refer to table below)
5 ... 6	number of frames (0 if not animated)
7 ... 8	graphic index
9 ... 40	graphic name (filled up with 0)

byte 4 - graphic properties:

Bit	Description
0	animated
3 .. 4	color depth (refer to table below)
5	transparent
7	disabled

bits 3 and 4 - color depth:

Value	Description
0	monochrome
1	8 bit
2	16 bit
3	24 bit

**Note**

- Additional to the [by\\_name](#) and [by\\_file](#) indices, the *graphic\_index* range from 0xFFE0 to 0xFFFF is reserved for internal use!

**Example**

```
\iC?G\D5
\iC?G\mGRAPHIC\0
\iC?G\fdIR/FILE.RII\0
```

These commands will read the information structure of the graphic with index 5 and graphic "GRAPHIC" from the on-board flash as well as graphic "FILE.RII" from the SD card's "DIR" folder.

**Response Example**

```
[ACK]
[01] [90]
[01] [2C]
[31]
[00] [0B]
[00] [05]
[00] [05]
anim[00] [00] .. [00]
[ACK]
```

**Note**

This response means that the requested graphic:

- has a width of 400 pixel
- has a height of 300 pixel
- is animated, has a color depth of 16bit and contains transparency
- has 11 frames
- has graphic index #5
- has graphic name "anim"

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Display Local Graphic](#)

[Load Animated Graphics](#)

**Set Graphic Alignment**

```
\i C G A bits::mode word::width word::height
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

mode	<a href="#">bits</a>	Bits 0 ... 3 and 7	alignment properties
width	<a href="#">word</a>	0 ... display width (0 = full display)	width of the alignment area
height	<a href="#">word</a>	0 ... display height (0 = full display)	height of the alignment area

Define an area in which the next graphic will be aligned when subsequently sending the command [Display Local Graphic](#) or [Display Graphic Area](#) as well as [Load Animated Graphics](#), [Set Animation Coordinates to X/Y](#) or [Set Animation Coordinates to Cursor Position](#).

Bit	Description
0	center graphic horizontally
1	center graphic vertically
2	right justify graphic
3	bottom justify graphic
7	turn alignment on

### Note

- After executing an drawing command, the alignment is automatically cleared. For consecutive aligning of graphics, this command has to be repeated!
- If a static graphic does not fit into the specified area, it will be cropped accordingly. When trying to align an animated graphic which doesn't fit into the area, the drawing command will return a `[NACK]`. Alignment will still be cleared in this case.
- Bit 7 of *mode* must always be set to enable the alignment settings. If only this bit is set, the next graphic is top/left justified and cropped in the specified area.
- If the *width* and/or *height* parameter is set to 0, the controller automatically uses the maximum area available (starting from the current cursor position when the drawing command is issued).
- If the *width* and/or *height* parameter would exceed the resulting right/bottom margin when the actual drawing command is executed, the controller automatically adjusts the *width* and/or *height* accordingly to the margins.
- Some bits of *mode* do not make sense when used together, so there are some logical precedent rules (centering always overrules justifying).
- The default value for *mode* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCGA\x83\D100\D200
```

This will set graphic alignment to center the next drawn graphic or loaded animation horizontally and vertically in an area of 100 pixels width and 200 pixels height.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Display Local Graphic](#)  
[Load Animated Graphics](#)  
[Display Graphic Area](#)  
[Set Animation Coordinates to X/Y](#)  
[Set Animation Coordinates to Cursor Position](#)

**Get Text Extent**

```
\i C ? T string::text
```

Parameter	Type	Range	Description
text	<a href="#">string</a>	ASCII chars (0x01 .. 0xFF)	text string to measure

Returns the extent of a text string when drawn with the current font.

**Note**

- To terminate a string, a NULL character (`\0`) has to be placed.
- Any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled.
- Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display.
- If you use a symbol font (refer to [Set Symbol Font](#)) no horizontal or vertical spaces between the characters will be displayed, and this command calculates its values accordingly.
- If text alignment is on (see [Set Text Alignment](#)), the reported *height* and *width* values correspond to the setting of the alignment. However, the current cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).

**Response:** [ACK] word::height  
word::width [ACK]

Parameter	Type	Range	Description
height	<a href="#">word</a>	0 ... display width - 1	height of the text string
width	<a href="#">word</a>	0 ... display height - 1	width of the text string

**Note****Example**

```
\iC?THello World!\0
```

## Response Example

```
[ACK] [00] [51] [00] [10] [ACK]
```

This response means that the text "Hello World!" will occupy a space of width = 81 (0x51) pixels, and height = 16 (0x10) pixels with the selected font.

### Note

#### See also:

[Write Text](#)  
[Write Unicode Text](#)  
[Get Unicode Text Extent](#)  
[Get Text Message Extent](#)

## Get Unicode Text Extent

```
\i C ? U string::unicode_text
```

Parameter	Type	Range	Description
unicode_text	<a href="#">wstring</a>	Unicode chars (0x0001 .. 0xFFFF)	unicode text to measure

Returns the extent of a unicode text string according to the current font.

### Note

- To terminate a string, a NULL character (`\0`) has to be placed.
- Any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled.
- Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display.
- If you use a symbol font (refer to [Set Symbol Font](#)) no horizontal or vertical spaces between the characters will be displayed, and the this command calculates its values accordingly.
- If text alignment is on (see [Set Text Alignment](#)), the reported *height* and *width* values correspond to the setting of the alignment. However, the current cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).

**Response:** [ACK] word::height  
word::width [ACK]

Parameter	Type	Range	Description
height	<a href="#">word</a>	0 ... display width - 1	width of the text string
width	<a href="#">word</a>	0 ... display height - 1	height of the text string



## Example

```
\iC?UHello World!\0
```

## Response Example

```
[ACK] [00] [51] [00] [10] [ACK]
```

This response means that the text "Hello World!" will occupy a space of width = 81 (0x51) pixels, and height = 16 (0x10) pixels with the selected font.

Not supported by: DPC3020, DPC2060, DPC10xx

## See also:

[Write Unicode Text](#)

[Write Text](#)

[Get Text Extent](#)

[Get Text Message Extent](#)

## Get Text Message Extent

### by index:

```
\i C ? t word::message_index
```

Parameter	Type	Range	Description
message_index	<a href="#">byte</a>	0 ... max. message_index	index of the message stored in the iLCD

### by name:

```
\i C ? t by_name::message_name
```

Parameter	Type	Range	Description
message_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated message name

Returns the extent of a text message (stored in the iLCD via iLCD Manager XE) when drawn with the current font.

## Note

- If the message with the corresponding *message\_index* or *message\_name* is not available, a *[NACK]* is returned.
- When addressing by index, the message offset (see [Set Message Offset](#)) is taken into account.
- When addressing by name, the message prefix (refer to [Set Message Name Prefix](#)) and suffix (refer to [Set Message Name Suffix](#)) are taken into account.

- Any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled.
- Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display.
- If you use a symbol font (refer to [Set Symbol Font](#)) no horizontal or vertical spaces between the characters will be displayed, and the this command calculates its values accordingly.
- If text alignment is on (see [Set Text Alignment](#)), the reported *height* and *width* values correspond to the setting of the alignment. However, the current cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).

**Response:** [ACK] word::height  
word::width [ACK]

Parameter	Type	Range	Description
height	<a href="#">word</a>	0 ... display width - 1	width of the text string
width	<a href="#">word</a>	0 ... display height - 1	height of the text

### Example

```
\iC?t\D1
```

### Response Example

```
[ACK] [00] [51] [00] [10] [ACK]
```

This response means that the text message stored with index 1 will occupy a space of width = 81 (0x51) pixels, and height = 16 (0x10) pixels with the selected font.

### Note

#### See also:

[Get Text Extent](#)

## Set Text Alignment

```
\i C T bits::mode word::width word::height
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 0 ... 7	alignment properties
width	<a href="#">word</a>	0 ... display width (0 = full display)	width of the alignment area
height	<a href="#">word</a>	0 ... display height (0 = full display)	height of the alignment area

This powerful command helps you to align text on the screen automatically. The next invocation of command [Write Text](#), [Write Text Message](#) or [Get Text Extent](#) will align, word-wrap and crop text corresponding to the *mode* set by this command in the specified area. A maximum of 1024 characters and 48 text lines can be aligned. The *mode* consists of the following bits, which can be binary ORed together:

Bit	Description
0	center text horizontally
1	center text vertically
2	right justify text
3	bottom justify text
4	do not word wrap text
5	add horizontal space for border
6	add vertical space for border
7	turn alignment on

#### Note

- After executing an output-command like for example [Write Text Message](#), the alignment is automatically cleared. For consecutive aligning of text strings, the set text alignment command has to be repeated!
- Bit 7 must always be set to enable the alignment settings. Clearing this bit allows an accidentally set alignment to be disabled before the text output.
- If alignment is turned on and none of the bits 0~3 are set, following text will be top/left justified (just as without any alignment), but also wrapped and cropped in the specified alignment area.
- Some bits do not make sense when used together, so there are some logical precedent rules (centering always overrules justifying).
- If the text does not fit into the given area, it will be truncated. If word wrap cannot be done due to words, which are longer than the available space, or word wrapping is switched off, the rest of the word will be continued on the next line.
- If the *width* and/or *height* parameter would exceed the resulting right/bottom margin when the actual output-command (for example [Write Text](#)) is executed, the controller automatically adjusts the *width* and/or *height* accordingly to the margins.
- If the *width* and/or *height* parameter is set to 0, the controller automatically uses the maximum area available (starting from the current cursor position when an actual output-command is issued).
- Even ANSI sequences (such as setting the font, etc.) can be used within the text to be aligned, although using cursor control ANSI commands within text alignments does not make sense and may produce unwanted results.
- Carriage returns and linefeeds can be used to force a new line regardless of the actual horizontal space already used for the current line. Entering a CR/LF pair causes one new line, entering e.g. two consecutive CRs causes two new lines.
- The default value for *mode* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

## Example

```
\iCT\x83\D200\D100
```

This will set text alignment to center subsequently outputted text horizontally and vertically in an area of 200 pixels width and 100 pixels height.

### See also:

[Write Text](#)  
[Write Text Message](#)  
[Get Text Extent](#)  
[Set Line Style](#)  
[Set Fonts Scaling](#)  
[ANSI Support](#)

## Set Line Style

```
\i C L byte::style
```

or

```
\i A L S byte::style
```

Parameter	Type	Range	Description
style	<a href="#">byte</a>	0x01 ... 0xFF	style of the line

This command allows a definition for a line style used by [Draw Line](#) and [Draw Rectangle](#). The parameter *style* represents a bit mask where a 1 represents a pixel to be written and a 0 a pixel to be omitted. The following table shows some examples for line styles:

Hex	Binary	Description
FF	1111 1111	solid line
55	1010 1010	dotted line
F0	1111 0000	dashed line (4 pixel black, 4 pixel white)
33	0011 0011	dashed line (2 pixel black, 2 pixel white)
27	0010 0111	dash-dotted line

### Note

- The default value for *style* is 0xFF (solid line). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- The least significant bit of style describes the first pixel painted when drawing a line from left to right.
- A style of 0x00 results in an invisible line, a *[NACK]* is returned in this case.

- When line thickness is greater than 1 (refer to [Set Line Thickness](#)), every bit represents a dot with a diameter of line thickness. The line ending mode (refer to [Set Line Ending Mode](#)) is then taken into account as well.
- The current line style is used when a rectangle is painted allowing the drawing of various styles of rectangles as well. When using a line style other than 0xFF, the two rectangular line caps styles lead to the same result (refer to [Set Line Caps Style](#)).

**Response:** [ACK]

### Example

```
\iCL\x33
\iALS\x33
```

These two commands are equivalent and will set the line style to a dashed line with 2 pixels black and 2 pixels white.

### See also:

[Set Line Thickness](#)  
[Set Line Ending Mode](#)  
[Set Line Caps Style](#)  
[Draw Line](#)  
[Draw Rectangle](#)

## Get Display Size

```
\i C ? D
```

Returns the size of the display.

**Response:** [ACK] word::width  
word::height [ACK]

Parameter	Type	Range	Description
width	<a href="#">word</a>	display width	display width
height	<a href="#">word</a>	display height	display height

### Note

### Response Example

```
[ACK] [03] [20] [01] [E0] [ACK]
```

This response means that the display has a width of 800 (0x320) pixels and a height of 480 (0x1E0) pixels.

**Note****Set TAB Spacing**

```
\i C S byte::tab_spacing
```

Parameter	Type	Range	Description
tab_spacing	<a href="#">byte</a>	1 ... 16	tabulator spacing

Sets the tabulator spacing. The next available multiple of *tab\_spacing* is the next character position where the cursor is placed when a TAB character is outputted.

**Note**

- The font width of the currently selected font is multiplied by *tab\_spacing* for the cursor position calculation.
- When using a proportional font, the font width is not identical with the character width of single characters.
- The default value for *tab\_spacing* is 8, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iCS\5
```

Sets the tabulator spacing to font width times 5.

**See also:**

[Set Text Alignment](#)

[Set Auto-Linefeed](#)

[Set Wrap Mode](#)

**Set Auto-Linefeed**

```
\i C F bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	sets auto-linefeed on (1) or off (0)

Enables/Disables auto-linefeed.

**Note**

- When auto-linefeed is on, receiving a carriage return (0x0D or \r) causes the cursor to be set to the left margin and a new line (0x0A or \n) to be added automatically.
- The default value for *on\_off* is 1, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iCF\1
```

Activates auto-linefeed.

**See also:**

[Set Text Alignment](#)  
[Set TAB Spacing](#)  
[Set Wrap Mode](#)

**Set Wrap Mode**

```
\i C W bool::horz_wrap bool::vert_wrap
```

Parameter	Type	Range	Description
horz_wrap	<a href="#">bool</a>	0 ... 1	sets horizontal wrap on (1) or off (0)
vert_wrap	<a href="#">bool</a>	0 ... 1	sets vertical wrap on (1) or off (0)

Enables/Disables character wrapping for horizontal and/or vertical character output.

**Note**

- When horizontal wrapping is on, characters outputted are automatically placed on a new line if there is no more space on the current line.
- When vertical wrap mode is on, the screen scrolls up when there is not enough space for the next line to be outputted to the LCD.
- The startup values for *horz\_wrap* and *vert\_wrap* are set via iLCD Manager XE.
- The default value for *horz\_wrap* and *vert\_wrap* is 1, but can be modified on the "Settings" page of iLCD Manager XE. They will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iCW\1\0
```

Activates horizontal wrapping but not vertical wrapping

**See also:**

[Set Text Alignment](#)

[Set Auto-Linefeed](#)

[Set TAB Spacing](#)

## Go Terminal Mode

```
\i C G =
```

This command allows the iLCD to leave the command mode and to enter the so-called terminal mode where the module acts like a standard ANSI terminal. After sending the `[ACK]` character, the only way to end the terminal mode is by sending the special escape sequence "End terminal mode" (see [Private ANSI Extensions](#)) from a terminal software. When the terminal mode is active, keystrokes are not reported via the make and break key anymore, but only the key code is reported. See [The Concept of iLCD's Touch Fields](#) and [Enable/Disable Keyboard Reporting](#) for detailed information.

**Note**

- Terminal mode is not supported in iLCD Manager XE. Sending alphanumeric characters will work, but e.g. curly brackets are interpreted as comments.
- Issuing this command automatically enables the ANSI mode (see [Enable/Disable ANSI](#)) and horizontal and vertical wrap mode (see [Set Wrap Mode](#)).
- The terminal mode can be forced at startup when set via iLCD Manager XE. Ending a terminal mode set at startup is done via the same escape sequence as described above.

**Response:** `[ACK]`

**See also:**

[ANSI Support](#)

## Set XON/XOFF for Terminal Mode

```
\i C X bool::on_off
```

Parameter	Type	Range	Description
<code>on_off</code>	<a href="#">bool</a>	0 ... 1	XON/XOFF mode on or off

When the iLCD is set to the terminal mode, the XON/OFF mode becomes relevant. If XON/XOFF is on (`on_off = 1`) the iLCD module sends a `[XOFF]` character (0x13) when the input buffer gets filled with 1/2 ("high water") of the total size and a `[XON]` character (0x11) when the buffer is emptied to 1/3 ("low water") of the total size.



**Note**

- The total size of the input buffer is 128 characters for all iLCD Controllers up to DPC3050 and 1024 characters for DPC3080 and higher. The command [Get Input Buffer Size](#) can be used to request the size of the input buffer.
- This is a standard mechanism used for terminals which do not have the ability to use hardware handshake signals (see some general information about this issue in the chapters [Command Structure](#) and [Terminal Mode](#)).
- XON/XOFF mode is not active when the module is in standard command mode.
- The default value for *on\_off* is 0, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Set Backlight Mode**

```
\i C B byte::mode
```

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	0 ... 3	mode for the backlight

Sets the backlight to the following options for *mode*:

Mode	Description
0	backlight off
1	backlight on
2	blink backlight
3	fade-out backlight (when previously on or blinking)

**Note**

- The default value for *mode* is retrieved from the EEPROM emulation at location 2. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM Related Commands](#).

**Response:** [ACK]

**Example**

```
\iCB\2
```

This example causes the backlight to blink.

**See also:**

[Get Backlight Mode](#)  
[Set Backlight Blink Frequency](#)  
[Set Backlight Intensity](#)  
[EEPROM Related Commands](#)

---

**Get Backlight Mode**

```
\i C ? B
```

Returns the current mode for the backlight.

**Response:** [ACK] byte::mode  
[ACK]

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	0 ... 3	backlight mode

Mode	Description
0	backlight off
1	backlight on
2	blink backlight
3	fade-out backlight

**Response Example**

```
[ACK] [01] [ACK]
```

This response means backlight is on.

**See also:**

[Set Backlight Mode](#)  
[Set Backlight Intensity](#)

---

**Set Backlight Blink Frequency**

```
\i C b byte::period
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

period	byte	1 ... 255	interval of state changes in units of 10ms
--------	------	-----------	--

Sets the blinking frequency for the backlight by defining the interval of state changes.

### Note

- The default value for *period* is 20, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When the backlight is in blink mode (see [Set Backlight Mode](#)) when the command is issued, the blink frequency changes immediately.

**Response:** [ACK]

### Example

```
\iCb\25
```

This value of 25 (0x19) gives a frequency of 2 Hertz (250ms on and 250ms off).

### See also:

[Set Backlight Mode](#)  
[Set Backlight Intensity](#)

## Set Backlight Intensity

```
\i C I byte::intensity
```

Parameter	Type	Range	Description
intensity	byte	0 ... 15	intensity of the backlight (max. intensity = 15)

Sets the backlight of the display.

### Note

- The default value for *intensity* is retrieved from the EEPROM emulation at location 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM Related Commands](#).
- This command does not switch of the backlight! That can be done with [Set Backlight Mode](#).

**Response:** [ACK]

## Example

```
\iCI\0
```

Sets the backlight intensity to its lowest level.

### See also:

[Set Backlight Mode](#)

[Set Backlight Intensity \(High-Res\)](#)

[Get Backlight Intensity \(High-Res\)](#)

## Set Backlight Intensity (High-Res)

```
\i C i byte::intensity
```

Parameter	Type	Range	Description
<code>intensity</code>	<a href="#">byte</a>	0 ... 255	intensity of the backlight (max. intensity = 255)

Sets the backlight of the display in high-res steps from 0 to 255.

### Note

- This command is similar to [Set Backlight Intensity](#) but provides a finer adjustment for the intensity (high-res).
- The default value for `intensity` is retrieved from the EEPROM emulation at location 1 in the coarse range (adjustable by the [Set Backlight Intensity](#) command). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM Related Commands](#).
- Although an `intensity` of 0 can darken the display completely, to turn the display off the command [Set Backlight Mode](#) should be used!

**Response:** [ACK]

## Example

```
\iCi\125
```

Sets the backlight intensity to a moderate level.

### See also:

[Set Backlight Mode](#)

[Set Backlight Intensity](#)

[Get Backlight Intensity](#)

## Get Backlight Intensity

```
\i C ? I
```

Delivers the current setting of the backlight intensity (even if the backlight is turned off).

**Response:** [ACK] byte::intensity  
[ACK]

Parameter	Type	Range	Description
intensity	byte	0 ... 15	numerical value for the intensity

### Response Example

```
[ACK] [0F] [ACK]
```

This response means that the backlight is set to its maximum brightness (15).

### See also:

[Set Backlight Intensity](#)

[Set Backlight Mode](#)

[Get Backlight Intensity \(High-Res\)](#)

## Get Backlight Intensity (High-Res)

```
\i C ? i
```

Delivers the current setting of the backlight intensity in high-res steps from 0 to 255 (even if the backlight is turned off).

**Response:** [ACK] byte::intensity  
[ACK]

Parameter	Type	Range	Description
intensity	byte	0 ... 255	numerical value for the intensity

### Response Example

```
[ACK] [FF] [ACK]
```

This response means that the backlight is set to its maximum brightness.

**See also:**[Set Backlight Intensity \(High-Res\)](#)[Set Backlight Mode](#)[Get Backlight Intensity](#)**Get Fixed LCD Contrast/Gamma**`\i C ? X`

Returns whether the iLCD panel has a fixed contrast/gamma setting or not.

**Note**

- Most iLCD panels do not need to have the contrast and gamma value set, as they have the optimum values already set by the TFT panel manufacturer. In this case (*fixed* = 1) although the commands for reading and setting the values can be carried out, there will be no visual effect!

**Response:** [ACK] bool::fixed  
[ACK]

Parameter	Type	Range	Description
fixed	<a href="#">bool</a>	0 ... 1	is the contrast/gamma fixed or not

**Response Example**`[ACK] [01] [ACK]`

This response means that the iLCD panel has a fixed contrast/gamma setting.

**See also:**[Get LCD Contrast](#)[Get LCD Gamma Value](#)**Set LCD Contrast**`\i C N byte::value`

Parameter	Type	Range	Description
value	<a href="#">byte</a>	0 ... 255	contrast value of the display (255 = maximum)

Sets the current contrast value of the LCD display.

**Note**

- The default value for *value* is retrieved from the EEPROM emulation at location 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM Related Commands](#).
- Not all LCD panels carry out changes to the contrast setting. Please refer to [Get Fixed LCD Contrast/Gamma](#).

**Response:** [ACK]

**Example**

```
\iCN\255
```

Sets the contrast of the display to its maximum.

**See also:**

[Get LCD Contrast](#)

[Get Fixed LCD Contrast/Gamma](#)

**Get LCD Contrast**

```
\i C ? N
```

Returns the current value set for the contrast of the display.

**Response:** [ACK] byte::value  
[ACK]

Parameter	Type	Range	Description
value	<a href="#">byte</a>	0 ... 255	numerical value for the display contrast (255 = maximum)

**Response Example**

```
[ACK] [FF] [ACK]
```

This response means that the contrast of the display is set to the maximum.

**See also:**

[Get LCD Gamma Value](#)

[Get Fixed LCD Contrast/Gamma](#)

[Set LCD Contrast](#)

## Set LCD Gamma Value

```
\i C M byte::value
```

Parameter	Type	Range	Description
value	<a href="#">byte</a>	0 ... 255	display gamma value

Sets the gamma value of the LCD panel.

### Note

- The default value for *value* is retrieved from the EEPROM emulation at location 3. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM Related Commands](#).
- Not all LCD panels carry out changes to the gamma setting. Please refer to [Get Fixed LCD Contrast/Gamma](#).

**Response:** [ACK]

### Example

```
\iCM\255
```

The gamma will be set to the maximum.

### See also:

[Get LCD Gamma Value](#)  
[Get Fixed LCD Contrast/Gamma](#)

## Get LCD Gamma Value

```
\i C ? M
```

Returns the current value set for the gamma of the display.

**Response:** [ACK] byte::value  
 [ACK]

Parameter	Type	Range	Description
-----------	------	-------	-------------



value	<a href="#">byte</a>	0 ... 255	display gamma value (255 = maximum)
-------	----------------------	-----------	-------------------------------------

### Response Example

```
[ACK] [FF] [ACK]
```

This response means that the gamma of the display is set to the maximum.

### See also:

[Get LCD Contrast](#)  
[Get Fixed LCD Contrast/Gamma](#)  
[Set LCD Gamma Value](#)

## Set Graphics Scaling

```
\i C E G byte::factor
```

Parameter	Type	Range	Description
factor	<a href="#">byte</a>	1 ... 16	factor for scaling graphics

Scales subsequent graphics pixel-wise according to *factor* in horizontal and vertical direction.

### Note

- This functionality can be used to "zoom" any graphic, e.g. when adapting images used on a 320x240 display to a 640x480 (VGA) iLCD.
- The scale factor is valid for all screens and viewports. Be aware, that all running animations are rescaled in the next frame update as soon as the scale factor is changed.
- If the scaled graphic exceeds the screen or viewport dimensions, scaled pixels that can't be completely drawn are omitted. This can lead to gaps between the cropped graphic and the according margin.
- The default value for *factor* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCEG\2
```

Graphics will be scaled by a factor of 2.

### See also:

[Set Fonts Scaling](#)  
[Set Column Coordinates Scaling](#)  
[Set Row Coordinates Scaling](#)

## Set Fonts Scaling

```
\i C E F byte::factor
```

Parameter	Type	Range	Description
factor	<a href="#">byte</a>	1 ... 16	factor for scaling fonts

Scales subsequently drawn text pixel-wise according to *factor* in horizontal and vertical direction.

### Note

- The factor set by this command is valid for all screens and viewports.
- The default font spacing is automatically scaled by this factor. When setting the spacing via [Set Font Spacing](#), the coordinate scale factors are taken into account (refer to [Set Row Coordinates Scaling](#) and [Set Column Coordinates Scaling](#)).
- The default value for *factor* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response** [ACK]

### Example

```
\iCEF\2
```

Text will be scaled by a factor of 2 after this command.

### See also:

[Set Graphics Scaling](#)  
[Set Column Coordinates Scaling](#)  
[Set Row Coordinates Scaling](#)

## Set Column Coordinates Scaling

```
\i C E C word::mult word::div
```

Parameter	Type	Range	Description
mult	<a href="#">word</a>	1 ... 4096	multiplier factor for horizontal positions
div	<a href="#">word</a>	1 ... 4096	divisor factor for horizontal positions

Scales column coordinates (horizontal positions) by a factor according to *mult* and *div*.

### Note

- Rounding errors may occur due to non-integer factors.
- This scaling is valid for all screens and viewports.
- This function can be used to migrate existing applications to another iLCD with a different display resolution.
- The parameter *radius* of the command [Draw Circle](#) is scaled with this factor.
- The display size is reported unscaled, i.e. as if both *mult* and *div* were 1 (refer to [Get Display Size](#)).
- The scan line commands are disabled as long as any coordinates scale factor is other than 1 (see [Write Scan Line](#) and [Read Scan Line](#)).
- The default value for *mult* and *div* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCEC\D1\D2
```

This command scales subsequent horizontal positions by a factor of 0.5 ( $mult/div = 1/2$ ).

### See also:

[Get Display Size](#)  
[Set Fonts Scaling](#)  
[Set Graphics Scaling](#)  
[Set Row Coordinates Scaling](#)

## Set Row Coordinates Scaling

```
\i C E R word::mult word::div
```

Parameter	Type	Range	Description
mult	<a href="#">word</a>	1 ... 4096	multiplier for vertical positions
div	<a href="#">word</a>	1 ... 4096	divisor for vertical positions

Scales row coordinates (vertical positions) by a factor according to *mult* and *div*.

### Note

- Rounding errors may occur due to non-integer factors.
- This scaling is valid for all screens and viewports.
- This function can be used to migrate existing applications to another iLCD with a different display resolution.

- The display size may be reported as unscaled, i.e. as if both *mult* and *div* were 1 (refer to [Get Display Size](#)).
- The scan line commands are disabled as long as any coordinates scale factor is other than 1 (see [Write Scan Line](#) and [Read Scan Line](#)).
- The parameter *radius* of the command [Draw Circle](#) is scaled with the column coordinates scale factor (see [Set Column Coordinates Scaling](#)).
- The default value for *mult* and *div* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iCER\D3\D2
```

This command scales subsequent vertical positions by a factor of 1.5 ( $mult/div = 3/2$ ).

### See also:

[Get Display Size](#)  
[Set Fonts Scaling](#)  
[Set Graphics Scaling](#)  
[Set Column Coordinates Scaling](#)

---

## LCD Attribute Commands

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Set Font](#)
- [Set Font Spacing](#)
- [Set Symbol Font](#)
- [Set Bold Mode](#)
- [Set Underline Mode](#)
- [Set Underline Position](#)
- [Set Inverse Mode](#)
- [Set Transparent Mode On/Off](#)
- [Set Foreground Color](#)
- [Set Background Color](#)
- [Set Border Color](#)
- [Set Border Shadow Color](#)
- [Set Shadow Offset](#)
- [Set Rectangle Corner Radius](#)
- [Set Line Style](#)
- [Set Line Thickness](#)
- [Set Line Caps Style](#)
- [Set Line Ending Mode](#)
- [Set Alpha](#)
- [Set Filling Color](#)
- [Set Filling Gradient](#)
- [Set Filling Tile](#)
- [Set Adjustment for Graphics](#)

- [Set Brightness Adjustment](#)
- [Set Contrast Adjustment](#)
- [Set Hue Adjustment](#)
- [Set Saturation Adjustment](#)

---

## Set Font

### by index:

```
\i A F word::font_index
```

Parameter	Type	Range	Description
font_index	<a href="#">word</a>	0 ... max. available font	index of the font

### by name:

```
\i A F by_name::font_name
```

Parameter	Type	Range	Description
font_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated font name

Sets a font for the subsequent text outputs.

### Note

- The default value for *font\_index* is 0, but can be modified on the "Fonts" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- The font with *font\_index* of 0 is always available (even if the flash memory is blank).
- When addressing by index, the font offset (see [Set Font Offset](#)) is taken into account.
- When addressing by name, the font prefix (refer to [Set Font Name Prefix](#)) and suffix (refer to [Set Font Name Suffix](#)) are taken into account.
- Setting a font also resets the underline position to 0 and sets the font spacing (see [Set Font Spacing](#)) and symbol font (see [Set Symbol Font](#)) mode to the way defined via iLCD Manager XE.
- There is also a private ANSI extension which allows setting the font via an escape sequence (see [Private ANSI Extensions](#)).

**Response:** [ACK]

### Example

```
\iAF\D3
```

All subsequently outputted texts are drawn with the font with index 3.

**See also:**

[Set Underline Position](#)  
[Set Font Spacing](#)  
[Set Symbol Font](#)  
[Set Font Offset](#)  
[Set Font Name Suffix](#)  
[Set Font Name Prefix](#)  
[Write Text](#)  
[Private ANSI Extensions](#)

**Set Font Spacing**

```
\i A S byte::x_spacing byte::y_spacing
```

Parameter	Type	Range	Description
<code>x_spacing</code>	<a href="#">byte</a>	0 ... 15	horizontal spacing (number of blank pixels between consecutive characters)
<code>y_spacing</code>	<a href="#">byte</a>	0 ... 15	vertical spacing (number of blank pixels between consecutive lines)

This command allows to overwrite the default font spacing defined via iLCD Manager XE for the currently selected font.

**Note**

- The default value for `x_spacing` and `y_spacing` is defined for every font on the "Fonts" page of iLCD Manager XE. They will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- The default font spacing is automatically scaled by font scale factor (refer to [Set Fonts Scaling](#)). When setting the spacing via this command, the coordinate scale factors are taken into account (refer to [Set Row Coordinates Scaling](#) for `x_spacing` and [Set Column Coordinates Scaling](#) for `y_spacing`).
- If the current font is set to symbol mode (see [Set Symbol Font](#)) the setting of `x_spacing` and `y_spacing` is ignored and no blank space is used between consecutive characters and lines.

**Response** [ACK]

**Example**

```
\iAS\15\10
```

This will overwrite the default font spacing to 15 pixels horizontally and to 10 pixels vertically.

**See also:**

[Set Font](#)  
[Set Symbol Font](#)

[Write Text](#)  
[Set Fonts Scaling](#)  
[Set Row Coordinates Scaling](#)  
[Set Column Coordinates Scaling](#)

## Set Symbol Font

```
\i A Y bool::on_off
```

Parameter	Type	Range	Description
on_off	bool	0 ... 1	determines whether the currently selected font is a symbol font or not

Sets the current font to a symbol font, meaning that there will be no blank pixels between consecutive characters and lines. This is especially useful when using fonts which contain border characters.

### Note

- Selecting a font which was defined as a symbol font via iLCD Manager XE does not require the issue of this command.
- If *on\_off* is 0, symbol mode is turned off and the default font spacing is used or when previously overwritten via the [Set Font Spacing](#), the last selected font spacing is used.

**Response** [ACK]

### Example

```
\iAY\1
```

Causes every subsequent text to be outputted with no blank pixels similar to a symbol font, until the command is repeated and the parameter *on\_off* is set to 0 again.

### See also:

[Set Font](#)  
[Write Text](#)

## Set Bold Mode

```
\i A B bool::on_off
```

Parameter	Type	Range	Description
on_off	bool	0 ... 1	text output is bold (1) or not (0)

Activates or deactivates bold mode for subsequent text outputs. This will add an additional pixel to the characters to be printed.

### Note

- When bold mode is activated, all characters occupy one more pixel on the display in horizontal direction.
- When using an anti-aliased font, bold mode is ignored.
- When using the appropriate alignment functions, this is taken into account automatically, but when running the terminal mode and using cursor control ANSI sequences, this may produce unwanted results (see [Terminal Mode](#))
- There is also an ANSI escape sequence (see [Syntax used in iLCD Manager XE](#)) to enable and disable bold mode.
- The default value for *on\_off* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response** [ACK]

### Example

```
\iAB\1
```

Causes every subsequent text to appear bold, until the command is repeated and the parameter *on\_off* is set to 0 again.

### See also:

[Set Font](#)  
[Write Text](#)

## Set Underline Mode

```
\i A U bool::on_off
```

Parameter	Type	Range	Description
<i>on_off</i>	<a href="#">bool</a>	0 ... 1	text is outputted underlined (1) or not (0)

Activates or deactivates an underline feature for any subsequent text outputs.

### Note

- By default the line is drawn at the lowest pixel position, which is within the character area. The vertical position of the underlining can be set via the [Set Underline Position](#) command.
- The default value for *on\_off* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]



## Example

```
\iAU\1
```

Causes every subsequent text to be underlined, until the command is repeated and the parameter *on\_off* is set to 0 again.

### See also:

[Set Font](#)  
[Set Underline Position](#)  
[Write Text](#)

---

## Set Underline Position

```
\i A u byte::position
```

Parameter	Type	Range	Description
position	<a href="#">signed byte</a>	-32 ... 31	underline position

Sets the vertical position for the underline.

### Note

- Underline Position is only relevant when Underline Mode is activated (see [Set Underline Mode](#)).
- Setting a font (see [Set Font](#)) causes *position* automatically to be reset to 0.
- The default value for *position* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- If the current vertical font spacing is 1 (see [Set Font Spacing](#)), any value greater than 0 for *position* causes the underline not drawn anymore as it would overwrite parts of next line.
- Using a negative value for *position* will cause the line to be drawn within the character area (thus crossing out instead of underlining the character).
- If *position* would cause overwriting parts of the previous line (depending on the font height), underlining is not carried out anymore.
- Setting a font (see [Set Font](#)) causes *position* automatically to be reset to 0.

**Response:** [ACK]

## Example

```
\iAu\2
```

This example sets the underline position to 2 pixels below the text.

**See also:**

[Set Font](#)  
[Set Font Spacing](#)  
[Write Text](#)

---

**Set Inverse Mode**

```
\i A I bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether inverse mode is on (1) or off (0)

Sets inverse mode on or off. The inverse mode swaps foreground and background colors.

**Note**

- All drawing, text and monochrome graphic commands take the inverse mode into account (for example [Write Text](#), [Draw Line](#), [Erase Display Area](#) etc.)
- The default value for *on\_off* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iAI\1
```

The foreground and background colors are swapped.

**See also:**

[Set Background Color](#)  
[Set Foreground Color](#)  
[Write Text](#)  
[Draw Line](#)  
[Erase Display Area](#)

---

**Set Transparent Mode On/Off**

```
\i A T bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether transparent mode is on (1) or off (0)

Activates or deactivates transparent mode for text outputs.

### Note

- Normally, drawing text will fill the background of the character with the current background color. With transparent mode on, it is possible to output text over an existing graphic.
- This command will not affect graphics with transparencies, these are always displayed transparent.
- The default value for *on\_off* is 0 (transparent mode off). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iAT\1
```

From now on, text output will be transparent.

**See also:**

[Set Font](#)

[Write Text](#)

## Set Foreground Color

```
\i A C F color::

```

Parameter	Type	Range	Description
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value for the foreground color

Sets the current foreground color all characters and lines are drawn with.

### Note

- The default value for *color\_value* is 0x000000 (black). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- Monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.

**Response:** [ACK]

### Example

```
\iACF\#FF0000
```

This example sets the foreground color to red.

Not supported by: DPC2060, DPC10xx

**See also:**

[24-Bit Color Values](#)  
[Set Background Color](#)  
[Set Inverse Mode](#)

## **Set Background Color**

```
\i A C B color::color_value
```

Parameter	Type	Range	Description
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value for the background color

Sets the current background color.

**Note**

- The default value for *color\_value* is 0xFFFFFFFF (white). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- If the transparent mode (see [Set Transparent Mode On/Off](#)) is off, the background of all text characters is drawn with the current background color.
- Using the [Erase Display](#) or the [Erase Display Area](#) command erases the display/area with the current background color.

**Response:** [ACK]

**Example**

```
\iACB\#0000FF
```

This example will set the background color to blue.

Not supported by: DPC2060, DPC10xx

**See also:**

[24-Bit Color Values](#)  
[Set Foreground Color](#)  
[Set Inverse Mode](#)  
[Set Transparent Mode On/Off](#)

## Set Border Color

```
\i A C R color::color_value
```

Parameter	Type	Range	Description
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value for the borders

Sets the color for borders.

### Note

- The default value for *color\_value* is 0x000000 (black). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- Drawing Rectangles with the [Draw Rectangle](#) command will make use of this border color.
- The shadows of borders have their own value and can be set via [Set Border Shadow Color](#).

**Response:** [ACK]

### Example

```
\iACR\#FFFF00
```

This example will set the border color to yellow.

Not supported by: DPC2060, DPC10xx

### See also:

[24-Bit Color Values](#)  
[Set Border Shadow Color](#)  
[Set Background Color](#)  
[Draw Rectangle](#)

## Set Border Shadow Color

```
\i A C S color::color_value
```

Parameter	Type	Range	Description
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value for the shadow of borders

Sets the shadow color for rectangle borders.

### Note

- The default value for *color\_value* is 0x7B7D7B (grey). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- Drawing Rectangles with the [Draw Rectangle](#) command will make use of this border shadow color when shadow is activated.

**Response:** [ACK]

### Example

```
\iACS\#333335
```

This example will set border shadow color to a dark grey.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[24-Bit Color Values](#)  
[Set Border Color](#)  
[Set Background Color](#)  
[Draw Rectangle](#)

## Set Shadow Offset

```
\i A H S byte::x_offset byte::y_offset
```

Parameter	Type	Range	Description
x_offset	<a href="#">signed byte</a>	-100 ... 100	horizontal shadow offset
y_offset	<a href="#">signed byte</a>	-100 ... 100	vertical shadow offset

Sets the shadow offset for the drop shadows of subsequent shape drawing (refer to [Draw Rectangle](#), [Draw Styled Circle](#), [Draw Ellipse](#)).

### Note

- If the origin of a shadow shape has negative coordinates, the drop shadow is disabled in the corresponding shape drawing function.
- The default values for *x\_offset* and *y\_offset* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- If any offset is 0, the shadow of shapes will be drawn with the historic default: half of the line thickness (refer to [Set Line Thickness](#)) if the frame is drawn and 1 pixel if not.

**Response:** [ACK]

### Example

```
\iAHS\20\ -10
```

All subsequent shapes are drawn with a drop shadow 20 pixels to the right (positive x) and 10 pixels above (negative y) the original shape.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Draw Rectangle](#)  
[Draw Styled Circle](#)  
[Draw Ellipse](#)  
[Set Border Shadow Color](#)

---

**Set Rectangle Corner Radius**

```
\i A H R word::radius
```

Parameter	Type	Range	Description
radius	<a href="#">word</a>	1 ... max[width, height]	radius of rectangle corners

Sets the corner radius for subsequently drawn rectangles.

**Note**

- The rectangle corner radius is only taken into account when the rounded corners flag (bit 0 of the *mode* parameter in [Draw Rectangle](#)) is set.
- For *radius*, the column coordinates scale factor is used (see [Set Column Coordinates Scaling](#)).
- The default value for *radius* is 5. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iAHR\D25
```

All subsequently rectangle drawing will be done with a corner radius of 25 pixels if the rounded corners flag is set.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Draw Rectangle](#)  
[Set Shadow Offset](#)  
[Set Line Thickness](#)

---

**Set Line Style**

```
\i C L byte::style
```

or

```
\i ALS byte::style
```

Parameter	Type	Range	Description
style	<a href="#">byte</a>	0x01 ... 0xFF	style of the line

This command allows a definition for a line style used by [Draw Line](#) and [Draw Rectangle](#). The parameter *style* represents a bit mask where a 1 represents a pixel to be written and a 0 a pixel to be omitted. The following table shows some examples for line styles:

Hex	Binary	Description
FF	1111 1111	solid line
55	1010 1010	dotted line
F0	1111 0000	dashed line (4 pixel black, 4 pixel white)
33	0011 0011	dashed line (2 pixel black, 2 pixel white)
27	0010 0111	dash-dotted line

#### Note

- The default value for *style* is 0xFF (solid line). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- The least significant bit of style describes the first pixel painted when drawing a line from left to right.
- A style of 0x00 results in an invisible line, a *[NACK]* is returned in this case.
- When line thickness is greater than 1 (refer to [Set Line Thickness](#)), every bit represents a dot with a diameter of line thickness. The line ending mode (refer to [Set Line Ending Mode](#)) is then taken into account as well.
- The current line style is used when a rectangle is painted allowing the drawing of various styles of rectangles as well. When using a line style other than 0xFF, the two rectangular line caps styles lead to the same result (refer to [Set Line Caps Style](#)).

**Response:** [ACK]

#### Example

```
\iCL\x33
\iALS\x33
```

These two commands are equivalent and will set the line style to a dashed line with 2 pixels black and 2 pixels white.

#### See also:

[Set Line Thickness](#)  
[Set Line Ending Mode](#)  
[Set Line Caps Style](#)  
[Draw Line](#)  
[Draw Rectangle](#)



## Set Line Thickness

```
\i A L T byte::thickness
```

Parameter	Type	Range	Description
thickness	<a href="#">byte</a>	1 ... 64	thickness of a line

Sets the current line thickness.

### Note

- The default value for *thickness* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iALT\5
```

This example will set the line thickness to 5.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:




[Draw Line](#)  
[Draw Rectangle](#)  
[Draw Styled Circle](#)  
[Draw Ellipse](#)  
[Set Line Caps Style](#)  
[Set Line Ending Mode](#)

## Set Line Caps Style

```
\i A L C byte::style
```

Parameter	Type	Range	Description
style	<a href="#">byte</a>	0 ... 2	style of the line caps

Sets the *style* of the line caps according to the following styles (the red dot represents the specified coordinates for the line ending):

Style	Description
0	
1	
2	

### Note

- The default value for *style* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iALC\1
```

After issuing this command, all subsequently drawn lines will have rectangular line caps and will start and end precisely at the specified coordinates (no overlapping).

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Draw Rectangle](#)  
[Set Line Thickness](#)  
[Set Line Ending Mode](#)













## Set Line Ending Mode

```
\i A L M byte::mode
```

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	0 ... 2	mode of the line endings

Sets the line ending mode according to the following values for *mode* (the red dot represents the specified coordinates for the line ending resp. rectangle corner):

Mode	Single Line	Connected	Description

	Lines				
0					line is drawn up to the last complete dash or dot
1					line is draw exactly to the specified end point
2					line is drawn to the last multiple of line thickness

### Note

- This value is only taken into account when drawing dashed or dotted lines (see [Set Line Style](#)).
- The default value for *mode* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iALM\2
```

This example will set the line ending mode to *mode* 1 (e.g. for drawing dashed rectangles without unexpectedly cropped dashes).

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Draw Rectangle](#)  
[Set Line Caps Style](#)  
[Set Line Thickness](#)  
[Set Line Style](#)

## Set Alpha

```
\i A A A byte::alpha
```

Parameter	Type	Range	Description
alpha	<a href="#">byte</a>	0 ... 255	opacity value

Sets an opacity value used for subsequent graphic drawing and filling. If set to 0, no drawing will be done, an *alpha* of 255 means full opacity.

**Note**

- The set alpha value is taken into account by all graphic ([Display Local Graphic](#), [Display Graphic Area](#)) and filling ([Fill Display](#), [Fill Display Area](#)) commands as well as shape drawing when filled ([Draw Rectangle](#), [Draw Styled Circle](#)).
- The default value for *alpha* is 255. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iAAA\127
```

All future drawing and filling will be done with half opacity.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Display Local Graphic](#)  
[Display Graphic Area](#)  
[Fill Display](#)  
[Fill Display Area](#)  
[Draw Rectangle](#)  
[Draw Styled Circle](#)

**Set Filling Color**

```
\i A P C color::color_value
```

Parameter	Type	Range	Description
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value for solid color filling

Sets the active fill pattern to the solid color *color\_value*. All subsequent filling will be done with this color.

**Note**

- The default fill pattern is a solid color with a *color\_value* of 0x000000 (black). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iAPC\#00FF00
```

All future filling will be done with the solid color green.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:





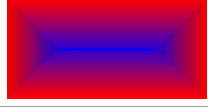
[Fill Display](#)  
[Fill Display Area](#)  
[Set Filling Gradient](#)  
[Set Filling Tile](#)  
[Draw Rectangle](#)  
[Draw Styled Circle](#)

## Set Filling Gradient

```
\i A P G byte::mode word::ramp color::from_color color::to_color
```

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	0 ... 4	gradient mode (see below)
ramp	<a href="#">word</a>	0 ... max[width, height]	pixels until <i>to_color</i> is reached (0 = maximum)
from_color	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value at beginning of gradient
to_color	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value at end of gradient

Sets the active fill pattern to a gradient going from *from\_color* to *to\_color*. All subsequent filling will be done with a gradient dependent on *mode*, which can be:

Mode	from red to blue	Description
0		horizontal gradient (ramp from left)
1		vertical gradient (ramp from top)
2		horizontal symmetrical gradient (ramp from left and right)
3		vertical symmetrical gradient (ramp from top and bottom)
4		buttonized (ramp from all directions)

**Note**

- If *ramp* is set to 0 or the area to fill is smaller than *ramp* in the according direction, the gradient fades over the complete area disregarding the set *ramp* value.
- In any other case, the gradient starts from *from\_color* and gradually changes to *to\_color* for *ramp* pixels. The rest of the according area is filled with the solid *to\_color*.
- The *ramp* is scaled by the corresponding coordinate scale factor (refer to [Set Row Coordinates Scaling](#) for horizontal and [Set Column Coordinates Scaling](#) for vertical gradients). For buttonized gradients, the smaller of those two scale factor is taken into account.
- The default fill pattern is a solid color (see [Set Filling Color](#)) with a *color\_value* of 0x000000 (black). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iAPG\4\D20\#FFFFFF\#00FF00
```

All future filling will be done with a gradient that fades from white to blue for 20 pixels in all directions. Filling an area with this pattern will give it a three-dimensional "buttonized" look.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Fill Display](#)  
[Fill Display Area](#)  
[Set Filling Color](#)  
[Set Filling Tile](#)  
[Draw Rectangle](#)  
[Draw Styled Circle](#)

**Set Filling Tile****by index:**

```
\i A P T word::graphic_index
```

Parameter	Type	Range	Description
graphic_index	<a href="#">word</a>	0 ... max. graphic index	index of tile graphic

**by name:**

```
\i A P T by_name::graphic_name
```

Parameter	Type	Range	Description
graphic_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated tile graphic name

**by filename:**

```
\i A P T by_file::graphic_filename
```

Parameter	Type	Range	Description
graphic_filename	<a href="#">by_file</a>	DOS filename (8.3 format)	name and path of the graphics file

Sets the active fill pattern to tiling using the graphic specified by *graphic\_index*, *graphic\_name* or *graphic\_filename* as tiles. All subsequent filling will be done with this tile graphic.

**Note**

- If no graphic with *graphic\_index*, *graphic\_name* or *graphic\_filename* is available, `[NACK]` is returned.
- When addressing by index, the graphic offset (see [Set Graphic Offset](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Set Graphic Name Prefix](#)) and suffix (refer to [Set Graphic Name Suffix](#)) are taken into account.
- Display and area filling can be done with scaled tiles (refer to [Set Graphics Scaling](#)). Scaling will be ignored when drawing filled shapes (e.g. [Draw Rectangle](#)) though.
- When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)) at the time filled drawing is done.
- The text/graphic orientation is ignored when filling with tiles (refer to [Set Text/Graphic Orientation](#)).

**Response:** [ACK]

**Example**

```
\iAPT\D1
\iAPT\mGRAPHIC\0
\iAPT\fdIR/FILE.RII\0
```

Sets the graphic with index 1, graphic "GRAPHIC" from the on-board flash and graphic "FILE.RII" from the SD card's "DIR" folder as the fill pattern. Only the last successfully set tile graphic is used for future filling.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Fill Display](#)  
[Fill Display Area](#)  
[Set Filling Color](#)  
[Set Filling Gradient](#)  
[Set Adjustment for Graphics](#)  
[Draw Rectangle](#)  
[Draw Styled Circle](#)

## Set Adjustment for Graphics

```
\i A A G bits::mode
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 0 ... 4	flags for possible modifications

Sets the adjustments used for all graphics drawn subsequently. The value for any selected modification has to be set before the actual drawing command is sent. Only the modifications set in the *mode* bitmask will be carried out:

Bit	Description
0	brightness (see <a href="#">Set Brightness Adjustment</a> )
1	contrast (see <a href="#">Set Contrast Adjustment</a> )
2	hue (see <a href="#">Set Hue Adjustment</a> )
3	saturation (see <a href="#">Set Saturation Adjustment</a> )
4	invert

### Note

- All adjustment modifications can be combined and carried out simultaneously with this command. The modifications are processed in the following order: invert - hue/saturation - brightness/contrast.
- After setting graphic adjustment, tile filling will also be adjusted accordingly (refer to [Set Filling Tile](#)).
- To draw unadjusted graphics again, send this command with *mode* 0.

**Response:** [ACK]

### Example

```
\iAAG\x10
```

All future graphic drawing will have the color values inverted.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Display Local Graphic](#)  
[Display Graphic Area](#)  
[Fill Display](#)  
[Fill Display Area](#)  
[Draw Rectangle](#)  
[Draw Styled Circle](#)



## Set Brightness Adjustment

```
\i A A B word::value
```

Parameter	Type	Range	Description
value	<a href="#">signed word</a>	-255 ... 255	value for brightness adjustment

Sets the *value* for brightness adjustment used by the commands [Adjust Display](#) and [Adjust Display Area](#) as well as graphic drawing commands when [Set Adjustment for Graphics](#) is used.

### Note

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iAAB\D-50
```

Sets the value for brightness adjustments to -50.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Adjust Display](#)  
[Adjust Display Area](#)  
[Set Adjustment for Graphics](#)  
[Set Contrast Adjustment](#)  
[Set Hue Adjustment](#)  
[Set Saturation Adjustment](#)

## Set Contrast Adjustment

```
\i A A C word::value
```

Parameter	Type	Range	Description
value	<a href="#">signed word</a>	-255 ... 255	value for contrast adjustment

Sets the *value* for contrast adjustment used by the commands [Adjust Display](#) and [Adjust Display Area](#) as well as graphic drawing commands when [Set Adjustment for Graphics](#) is used.

### Note

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iAAC\D100
```

Sets the value for contrast adjustments to 100.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Adjust Display](#)  
[Adjust Display Area](#)  
[Set Adjustment for Graphics](#)  
[Set Brightness Adjustment](#)  
[Set Hue Adjustment](#)  
[Set Saturation Adjustment](#)

## Set Hue Adjustment

```
\i A A H word::value
```

Parameter	Type	Range	Description
value	<a href="#">signed word</a>	-255 ... 255	value for hue adjustment

Sets the *value* for hue adjustment used by the commands [Adjust Display](#) and [Adjust Display Area](#) as well as graphic drawing commands when [Set Adjustment for Graphics](#) is used.

### Note

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iAAH\D-100
```

Sets the value for hue adjustments to -100.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Adjust Display](#)  
[Adjust Display Area](#)

[Set Adjustment for Graphics](#)  
[Set Brightness Adjustment](#)  
[Set Contrast Adjustment](#)  
[Set Saturation Adjustment](#)

---

## Set Saturation Adjustment

```
\i A A S word::value
```

Parameter	Type	Range	Description
value	<a href="#">signed word</a>	-255 ... 255	value for saturation adjustment

Sets the *value* for saturation adjustment used by the commands [Adjust Display](#) and [Adjust Display Area](#) as well as graphic drawing commands when [Set Adjustment for Graphics](#) is used.

### Note

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iAAS\D50
```

Sets the value for saturation adjustments to 50.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Adjust Display](#)  
[Adjust Display Area](#)  
[Set Adjustment for Graphics](#)  
[Set Brightness Adjustment](#)  
[Set Contrast Adjustment](#)  
[Set Hue Adjustment](#)

---

## LCD Draw Commands

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Erase Display](#)

- [Erase Display Area](#)
- [Invert Display](#)
- [Invert Display Area](#)
- [Write Text](#)
- [Write Unicode Text](#)
- [Write Text Message](#)
- [Scroll Up](#)
- [Scroll Down](#)
- [Scroll Left](#)
- [Scroll Right](#)
- [Write Scan Line](#)
- [Read Scan Line](#)
- [Write Scan Line and Advance](#)
- [Write 1D/2D Run-Length Encoded Scan Line](#)
- [Read 1D/2D Run-Length Encoded Scan Line](#)
- [Set/Clear Pixel](#)
- [Set/Clear Pixel at X/Y](#)
- [Draw Dot](#)
- [Draw Dot at X/Y](#)
- [Draw Line](#)
- [Draw Rectangle](#)
- [Draw Circle](#)
- [Draw Styled Circle](#)
- [Draw Ellipse](#)
- [Adjust Display](#)
- [Adjust Display Area](#)
- [Fill Display](#)
- [Fill Display Area](#)

---

## **Erase Display**

```
\i D E
```

Clears the entire active viewport to the current background color. Afterwards the cursor position is set to 0/0.

### **Note**

- If the selected viewport is 0 (the entire screen), all animations on this draw screen are stopped, even if they are drawn in another viewport.
- If inverse mode is active, the viewport is cleared to the foreground color instead.

**Response:** [ACK]

### **Example**

```
\iDE
```

Clears the display to the background color.

**See also:**

[Set Inverse Mode](#)  
[Set Background Color](#)  
[Erase Display Area](#)

**Erase Display Area**

```
\i D e word::width word::height
```

Parameter	Type	Range	Description
width	<a href="#">word</a>	0 ... display width (0 = full width)	horizontal value for the area to be erased
height	<a href="#">word</a>	0 ... display height (0 = full height)	vertical value for the area to be erased

Clears an area of the screen specified by the parameters *width* and *height* beginning from the current cursor position.

**Note**

- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without notifying an error.
- Depending on the current setting of the inverse mode, the area is either cleared to the background or to the foreground color.

**Response:** [ACK]

**Example**

```
\iDe\D400\D0
```

Clears the left half of a display with 800x480 pixels, provided the cursor position was 0/0.

**See also:**

[Erase Display](#)  
[Get Cursor Position](#)  
[Set Cursor Position](#)  
[Set Inverse Mode](#)  
[Set Background Color](#)

**Invert Display**

```
\i D I
```

Inverts all pixels on the currently active screen.

**Note**

- Using this command does not turn the inverse mode on or off.

**Response:** [ACK]

**Example**

```
\iDI
```

Black pixels are now white, red pixels are cyan, etc.

**See also:**

[Set Inverse Mode](#)

**Invert Display Area**

```
\i D i word::width word::height
```

Parameter	Type	Range	Description
width	<a href="#">word</a>	0 ... display width (0 = full width)	horizontal value for the area to be inverted
height	<a href="#">word</a>	0 ... display height (0 = full height)	vertical value for the area to be inverted

Inverts an area of the currently active screen specified by the parameters *width* and *height* beginning from the current cursor position.

**Note**

- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without notifying an error.
- Using this command does not turn the inverse mode on or off.

**Response:** [ACK]

**Example**

```
\iDi\D100\D240
```

Inverts all colors in an area of 100x240 pixels.

**See also:**

[Invert Display](#)  
[Set Cursor Position](#)  
[Set Inverse Mode](#)

## Write Text

```
\i D T string::text_string
```

Parameter	Type	Range	Description
text_string	<a href="#">string</a>	ASCII chars (0x01 .. 0xFF)	text to be output

Writes a *text\_string* to the current cursor position.

### Note

- To terminate a string, a NULL character (`\0`) has to be placed.
- Characters not defined in the character table of a font are replaced with a space.
- While writing text, the cursor position is constantly updated to the end of the last character.
- If the cursor position exceeds the right or bottom margin during text output and the corresponding wrap mode is not set, the following characters are ignored (refer to [Set Wrap Mode](#)). The cursor will then remain at the end of the last complete character.
- To get the horizontal and vertical space occupied by the outputted text use the [Get Text Extent](#) command.
- The text can contain any ANSI sequence (see [ANSI Support](#)) to e.g. control the cursor position (on a character or graphic based position) when the ANSI mode is on.

**Response:** [ACK]

### Example

```
\iDTHello World!\0
```

Writes "Hello World!" to the screen at the current cursor position and with the currently selected font and attributes.

### See also:

[Write Unicode Text](#)  
[Write Text Message](#)  
[Set Cursor Position](#)  
[Set Font](#)  
[Get Text Extent](#)  
[ANSI Support](#)  
[LCD Attribute Commands](#)

## Write Unicode Text

```
\i D U string::unicode_text
```

Parameter	Type	Range	Description
unicode_text	<a href="#">wstring</a>	Unicode chars (0x0001 .. 0xFFFF)	unicode text to be output

Writes a *unicode\_text* string to the current cursor position.

#### Note

- To terminate a string, a 2-byte NULL character (0x0000 resp. `\0`) has to be placed.
- Characters not defined in the character table of a font are replaced with a space.
- If the column address exceeds the right or bottom margin during printing, and the corresponding wrap mode is not set, the following characters are ignored.
- Writing text will increase the row or column address and therefore change the cursor position.
- To get the horizontal and vertical space occupied by the outputted text use the [Get Unicode Text Extent](#) command.
- The text can contain any ANSI sequence (see [ANSI Support](#)) to e.g. control the cursor position (on a character or graphic based position) when the ANSI mode is on.

**Response:** [ACK]

#### Example

```
\iDUHello World!\0
```

Writes "Hello World!" to the screen at the current cursor position and with the currently selected unicode font and attributes.

Not supported by: DPC3020, DPC2060, DPC10xx

#### See also:

[Write Text](#)  
[Write Text Message](#)  
[Set Cursor Position](#)  
[Set Font](#)  
[Get Unicode Text Extent](#)  
[ANSI Support](#)  
[LCD Attribute Commands](#)

## Write Text Message

#### by index:

```
\i D t word::message_index
```

Parameter	Type	Range	Description
message_index	<a href="#">word</a>	0 ... max. text message index	index of the defined message



**by name:**

```
\i D t by_name::message_name
```

Parameter	Type	Range	Description
message_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated message name

Writes a text string similar to the [Write Text](#) command, except that it uses messages previously defined in iLCD Manager XE.

**Note**

- Text messages can contain ANSI sequences as well.
- When addressing by index, the message offset (see [Set Message Offset](#)) is taken into account.
- When addressing by name, the message prefix (refer to [Set Message Name Prefix](#)) and suffix (refer to [Set Message Name Suffix](#)) are taken into account.

**Response:** [ACK]

**Example**

```
\iDt\D3
\iDt\mMESSAGE\0
```

Writes the message with index 3 and message "MESSAGE" to the screen at the current cursor position with the currently selected font and attributes.

**See also:**

[Write Text](#)  
[Set Cursor Position](#)  
[Set Font](#)  
[Get Text Extent](#)  
[ANSI Support](#)  
[LCD Attribute Commands](#)

**Scroll Up**

```
\i D S U word::scroll_y
```

Parameter	Type	Range	Description
scroll_y	<a href="#">word</a>	0 ... display height - 1	vertical distance in pixels to be scrolled

Scrolls the currently active viewport *scroll\_y* pixels up.

**Note**

- The new rows at the bottom are drawn in the current background color.

**Response:** [ACK]

**Example**

```
\iDSU\25
```

Scrolls the viewport content 25 pixels up.

**See also:**

[Scroll Down](#)  
[Scroll Left](#)  
[Scroll Right](#)  
[Scroll Up Screen](#)  
[Scroll Down Screen](#)  
[Scroll Left Screen](#)  
[Scroll Right Screen](#)

---

**Scroll Down**

```
\i D S D word::scroll_y
```

Parameter	Type	Range	Description
scroll_y	<a href="#">word</a>	0 ... display height - 1	vertical distance in pixels to be scrolled

Scrolls the currently active viewport *scroll\_y* pixels down.

**Note**

- The new rows at the top are drawn in the current background color.

**Response:** [ACK]

**Example**

```
\iDSU\50
```

Scrolls the viewport content 50 pixels down.

**See also:**

[Scroll Up](#)  
[Scroll Left](#)  
[Scroll Right](#)  
[Scroll Up Screen](#)

[Scroll Down Screen](#)  
[Scroll Left Screen](#)  
[Scroll Right Screen](#)

## **Scroll Left**

```
\i DSL word::scroll_x
```

Parameter	Type	Range	Description
scroll_x	<a href="#">word</a>	0 ... display width - 1	horizontal distance in pixels to be scrolled

Scrolls the currently active viewport *scroll\_x* pixels to the left.

### **Note**

- The new columns at the right are drawn in the current background color.

**Response:** [ACK]

### **Example**

```
\iDSL\25
```

Scrolls the viewport content 25 pixels to the left.

### **See also:**

[Scroll Right](#)  
[Scroll Up](#)  
[Scroll Down](#)  
[Scroll Up Screen](#)  
[Scroll Down Screen](#)  
[Scroll Left Screen](#)  
[Scroll Right Screen](#)

## **Scroll Right**

```
\i DSL word::scroll_x
```

Parameter	Type	Range	Description
scroll_x	<a href="#">word</a>	0 ... display width - 1	horizontal distance in pixels to be scrolled

Scrolls the currently active viewport *scroll\_x* pixels to the right.

**Note**

- The new columns at the left are drawn in the current background color.

**Response:** [ACK]

**Example**

```
\iDSR\250
```

Scrolls the viewport content 250 pixels to the right.

**See also:**

[Scroll Left](#)  
[Scroll Up](#)  
[Scroll Down](#)  
[Scroll Up Screen](#)  
[Scroll Down Screen](#)  
[Scroll Left Screen](#)  
[Scroll Right Screen](#)

**Write Scan Line**

```
\i D N W word::no_of_pixels word::p0 ...
```

Parameter	Type	Range	Description
no_of_pixels	<a href="#">word</a>	1 ... display width	number of pixels to write
p0 ...	<a href="#">color_16bit</a>	0x0000 ... 0xFFFF	color value for pixels

Writes a horizontal scan line consisting of *no\_of\_pixels* pixels onto the screen.

**Note**

- Data for any sent pixel is made up as a [16-Bit Color Value](#).
- The location where the pixels are written to (where the horizontal line begins) is determined by the current cursor position.
- The column address is incremented after every pixel. If the column address exceeds the right margin during printing, [NACK] is returned and the command processing returns and waits for the next command introducer (\i). Even so, the scan line is drawn correctly until it hits the screen margin.

**Response:** [ACK]

**Example**

```
\iDNW\D7\X0\FFFFFF\XF8\FFFFFF\XF8\FFFFFF\X0
```

This example will write 7 pixels to the screen, starting from the current cursor position. The first and the last pixel are black, pixels in between are white or red alternatingly.

**See also:**

[Write Scan Line and Advance](#)  
[Read Scan Line](#)  
[Write 1D/2D Run-Length Encoded Scan Line](#)  
[Read 1D/2D Run-Length Encoded Scan Line](#)  
[16-Bit Color Values](#)

## Read Scan Line

`\i D N R word::no_of_pixels`

Parameter	Type	Range	Description
no_of_pixels	<a href="#">word</a>	1 ... display width	number of pixel to be scanned

Reads the color information of *no\_of\_pixels* pixels in a horizontal line from left to right, starting from the current cursor position.

**Note**

- Data for any reported pixel is made up as [16-Bit Color Value](#).
- If *no\_of\_pixels* + the current column value exceeds the right margin, a *[NACK]* is sent instead of reporting the data.

**Response:** `[ACK] p0 ... [ACK]`

Parameter	Type	Range	Description
p0 ...	<a href="#">word</a>	0x0000 ... 0xFFFF	color value for pixels

### Response Example

`[ACK] [00] [00] [00] [00] [FF] [FF] [FF] [FF] [FF] [FF] [ACK]`

In this example a line of 5 pixels has been scanned. The first two pixels are black and pixels 3 to 5 are white.

**See also:**

[Write Scan Line](#)  
[Read 1D/2D Run-Length Encoded Scan Line](#)  
[Write 1D/2D Run-Length Encoded Scan Line](#)  
[16-Bit Color Values](#)

## Write Scan Line and Advance

```
\i D N A word::no_of_pixels word::p0 ...
```

Parameter	Type	Range	Description
no_of_pixels	<a href="#">word</a>	1 ... display width	number of pixels to write
p0 ...	<a href="#">color_16bit</a>	0x0000 ... 0xFFFF	color value for pixels

Writes a horizontal scan line consisting of *no\_of\_pixels* pixels onto the screen.

### Note

- Data for any sent pixel is made up as a [16-Bit Color Value](#).
- The location where the pixels are written to (where the horizontal line begins) is determined by the current cursor position.
- The column address is incremented after every pixel. If the column address exceeds the right margin during printing, *[NACK]* is returned and the command processing returns and waits for the next command introducer (*\i*). Even so, the scan line is drawn correctly until it hits the screen margin.
- After the command the x cursor position is the same as before and the y cursor position is incremented.

**Response:** [ACK]

### Example

```
\iDNA\D7\X0\FFFFFF\XF8\FFFFFF\XF8\FFFFFF\X0
```

This example will write 7 pixels to the screen, starting from the current cursor position. The first and the last pixel are black, pixels in between are white or red alternatingly.

### See also:

[Write Scan Line](#)  
[Read Scan Line](#)  
[Write 1D/2D Run-Length Encoded Scan Line](#)  
[Read 1D/2D Run-Length Encoded Scan Line](#)  
[16-Bit Color Values](#)

## Write 1D/2D Run-Length Encoded Scan Line

```
\i D N w byte::prev_line_offset word::no_of_pixels word::no_of_rle_bytes  
byte::b0 ...
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

prev_line_offset	<a href="#">signed byte</a>	-8 ... 8	y-offset of scanline referred to (0 = none = 1D encoded)
no_of_pixels	<a href="#">word</a>	1 ... display width	number of pixels to write
no_of_rle_bytes	<a href="#">word</a>	2 ... display width + 2	length of encoded data in bytes
b0 ...	<a href="#">bytes</a>	0x00 ... 0xFF	run-length encoded data

Writes a horizontal scan line consisting of *no\_of\_pixels* pixels onto the screen. The *no\_of\_rle\_bytes* long data is sent with run-length encoding.

### Note

- The location where the pixels are written to (where the horizontal line begins) is determined by the current cursor position.
- The column address is incremented after every pixel. If the column address exceeds the right margin during printing, *[NACK]* is returned and the command processing returns and waits for the next command introducer (*\i*). Even so, the scan line is drawn correctly until it hits the screen margin.
- When *prev\_line\_offset* is 0 or *prev\_line\_offset* + the current row is negative, only 1D encoded data will be accepted. When a sequence indicating 2D encoding is received in this case, drawing is aborted and a *[NACK]* is returned.
- When *prev\_line\_offset* is other than 0 and *prev\_line\_offset* + the current row is 0 or higher, the data can include sequences for 2D encoding with reference to the relatively specified row.
- Color values in the reported data are made up as [16-Bit Color Values](#).
- For detailed information about the encoding algorithm and its usage, please refer to the [2D Run-Length Encoding Application Note](#).

There is also a local *2D Run-Length Encoding Application Note* version available which is named as *2D RLE Application Note.pdf*, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

**Response:** [ACK]

### Example

```
\iDNw\0\D50\D3\xB1\X1F00
```

This example will write 50 blue pixels to the screen, starting from the current cursor position. The encoded data consists of 3 bytes: 0xB1 indicates a horizontal repetition (bit 7 set) of 50 pixels and 0x1F00 is the 16-bit color value for pure blue.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Read 1D/2D Run-Length Encoded Scan Line](#)  
[Write Scan Line](#)  
[Read Scan Line](#)  
[16-Bit Color Values](#)

## Read 1D/2D Run-Length Encoded Scan Line

```
\i D N r byte::prev_line_offset word::no_of_pixels
```

Parameter	Type	Range	Description
prev_line_offset	<a href="#">signed byte</a>	-8 ... 8	y-offset of scanline referred to (0 = none = 1D encoded)
no_of_pixels	<a href="#">word</a>	1 ... display width	number of pixel to be scanned

Reads the color information of *no\_of\_pixels* pixels in a horizontal line from left to right, starting from the current cursor position and reports the data with run-length encoding.

### Note

- If *no\_of\_pixels* + the current column value exceeds the right margin, a *[NACK]* is sent instead of reporting the data.
- When *prev\_line\_offset* is 0 or *prev\_line\_offset* + the current row is negative, the data will be 1D encoded.
- When *prev\_line\_offset* is other than 0 and *prev\_line\_offset* + the current row is 0 or higher, 2D encoding with reference to the relatively specified row is carried out.
- Color values in the reported data are made up as [16-Bit Color Values](#).
- For detailed information about the encoding algorithm and its usage, please refer to the [2D Run-Length Encoding Application Note](#).

There is also a local *2D Run-Length Encoding Application Note* version available which is named as *2D RLE Application Note.pdf*, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

**Response:** [ACK] no\_of\_bytes b0 ... [ACK]

Parameter	Type	Range	Description
no_of_rle_bytes	<a href="#">word</a>	2 ... 2 * display width + display width / 32 + 2	length of encoded data in bytes
p0 ...	<a href="#">bytes</a>	0x00 ... 0xFF	run-length encoded data

### Response Example

```
[ACK] [00] [03] [B1] [00] [F8] [ACK]
```

In this example a line of 50 red pixels has been scanned. The encoded data consists of 3 bytes: 0xB1 indicates a horizontal repetition (bit 7 set) of 50 pixels and 0x00F8 is the 16-bit color value for pure red.

Not supported by: DPC3020, DPC2060, DPC10xx



**See also:**

[Write 1D/2D Run-Length Encoded Scan Line](#)  
[Read Scan Line](#)  
[Write Scan Line](#)  
[16-Bit Color Values](#)

**Set/Clear Pixel**

```
\i D P bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether a pixel is set (1) or cleared (0)

Sets or clears the pixel at the current cursor position.

**Note**

- No advancing of the current cursor position will occur.
- Setting/clearing a pixel is done with the current foreground/background color. When inverse mode is on, setting/clearing a pixel will be done in the opposite way.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), the pixel is drawn with the according opacity.

**Response:** [ACK]

**Example**

```
\iDP\1
```

Sets a pixel at the current cursor position.

**See also:**

[Set/Clear Pixel at X/Y](#)  
[Set Background Color](#)  
[Set Foreground Color](#)  
[Set Inverse Mode](#)  
[Set Alpha](#)

**Set/Clear Pixel at X/Y**

```
\i D p word::pos_x word::pos_y bool::on_off
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

r			
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal position
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical position
on_off	<a href="#">bool</a>	0 ... 1	determines whether a pixel is set (1) or cleared (0)

Sets or clears a pixel at the specified position.

### Note

- The cursor position will be updated after this command.
- Setting/clearing a pixel is done with the current foreground/background color. When inverse mode is on, setting/clearing a pixel will be done in the opposite way.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), the pixel is drawn with the according opacity.

**Response:** [ACK]

### Example

```
\iDp\D120\D55\1
```

Sets or clears a pixel at the position x = 120 and y = 55 pixels.

### See also:

[Set Background Color](#)  
[Set Foreground Color](#)  
[Set Inverse Mode](#)  
[Set Alpha](#)

### Draw Dot

```
\i D D color::color_value
```

Parameter	Type	Range	Description
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color of the dot

Sets the pixel at the current cursor position to *color\_value*.

### Note

- No advancing of the current cursor position will occur.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), the pixel is drawn with the according opacity.

**Response:** [ACK]

**Example**

```
\iDD\#FF0000
```

Draws a red dot at the current cursor position.

**See also:**

[Draw Dot at X/Y](#)  
[Set/Clear Pixel](#)  
[Set/Clear Pixel at X/Y](#)  
[Set Alpha](#)

**Draw Dot at X/Y**

```
\i D d word::pos_x word::pos_y color::color_value
```

Parameter	Type	Range	Description
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal position
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical position
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color of the dot

Sets or clears a pixel at the specified position.

**Note**

- The cursor position will be updated after this command.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), the pixel is drawn with the according opacity.

**Response:** [ACK]

**Example**

```
\iDd\D20\D35\#00FF00
```

Draws a green dot at the position x = 20 and y = 35 pixels.

**See also:**

[Draw Dot](#)  
[Set/Clear Pixel](#)  
[Set/Clear Pixel at X/Y](#)  
[Set Alpha](#)

## Draw Line

```
\i D L word::end_x word::end_y
```

Parameter	Type	Range	Description
end_x	<a href="#">word</a>	0 ... display width - 1	horizontal ending position of the line
end_y	<a href="#">word</a>	0 ... display height - 1	vertical ending position of the line

Draws a line with the currently selected Line Style (see [Set Line Style](#)) starting from the current cursor position to the specified ending point.

### Note

- After the line is drawn the cursor position will be the ending point of the line.
- Drawing a line is done with the current foreground color. When inverse mode is on, the current background color is used instead.

**Response:** [ACK]

### Example

```
\iDL\D799\D479
```

Draws a line from the top left corner to the lower right corner, assuming the cursor position was 0/0 before this command was issued and the display has a resolution of 800x480 pixels.

### See also:

[Draw Rectangle](#)  
[Draw Circle](#)  
[Get Cursor Position](#)  
[Set Foreground Color](#)  
[Set Inverse Mode](#)  
[Set Line Style](#)

## Draw Rectangle

```
\i D R bits::mode word::width word::height
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 0 ... 7	flags for the rectangle properties
width	<a href="#">word</a>	1 ... display width	width of the rectangle
height	<a href="#">word</a>	1 ... display height	height of the rectangle

Draws a rectangle starting from the current cursor position to the ending point specified via *width* and *height*. The parameter *mode* allows to set several properties for the rectangle:

Bit	Description
0	rounded corners
1	drop shadow
2	surrounded with background color
3	filled with background color
4	use foreground color for filling (bit 3 must be set too)
5	filled with current fill pattern (bits 3 and 4 ignored if set)
6	draw without frame
7	alpha blending for shadow only

### Note

- If *mode* bit 6 is not set, the rectangle frame is drawn with the current border color (see [Set Border Color](#)). If the line thickness is greater than 1 (refer to [Set Line Thickness](#)), the lines of the rectangle expand to both sides of the specified corner points.
- By default, alpha blending is done for every activated feature (frame, filling, shadow). When *mode* bit 7 is set, the shadow will still be drawn with alpha blending, but frame and filling will have full opacity (refer to [Set Alpha](#)).
- The rectangle frame is drawn with regard to the current line style (see [Set Line Style](#)) only if the corner radius and alpha are set to their default values (refer to [Set Rectangle Corner Radius](#) and [Set Alpha](#)).
- If *mode* bit 1 is set, the drop shadow is drawn with the border shadow color (see [Set Border Shadow Color](#)) with regard to the current shadow offset (see [Set Shadow Offset](#)). If the offset leads to a negative left/upper corner of the shadow rectangle, no shadow will be drawn.
- When filling with a pattern (*mode* bit 5), the fill pattern has to be set in advance via [Set Filling Color](#), [Set Filling Gradient](#) or [Set Filling Tile](#). When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).
- When the rounded corners flag (*mode* bit 0) is set, the attribute rectangle corner radius is taken into account (see [Set Rectangle Corner Radius](#)). If the corner radius exceeds half of the rectangle *width* and/or *height*, the flag is ignored and a rectangle without rounded corners is drawn.
- The surrounding line (*mode* bit 2) is only drawn when corner radius and alpha are set to their default values (refer to [Set Rectangle Corner Radius](#) and [Set Alpha](#)) and a frame is drawn (*mode* bit 6 is not set).
- Filling with a fill pattern as well as drawing without a frame (*mode* bits 5 and 6) are supported by firmware versions 4.20 and higher. In previous versions, those bits are ignored.

**Response:** [ACK]

### Example

```
\iDR\x3\D50\D30
```

Draws a rectangle with a width of 50 pixels and a height of 30 pixels. Moreover, the rectangle has rounded corners and a drop shadow.

**See also:**

[Draw Circle](#)  
[Draw Styled Circle](#)  
[Get Cursor Position](#)  
[Set Inverse Mode](#)  
[Set Line Style](#)  
[Set Border Color](#)  
[Set Border Shadow Color](#)  
[Set Line Thickness](#)  
[Set Line Caps Style](#)  
[Set Filling Color](#)  
[Set Filling Gradient](#)  
[Set Filling Tile](#)  
[Set Alpha](#)  
[Erase Display Area](#)

**Draw Circle**

```
\i D C word::radius
```

Parameter	Type	Range	Description
radius	<a href="#">word</a>	1 ... max[width, height]	radius of the circle

Draws a circle with the radius *radius* and the current cursor position as center.

**Note**

- For *radius*, the column coordinates scale factor is used (see [Set Column Coordinates Scaling](#)).
- Drawing a circle is done with the current foreground color. When inverse mode is on, the current background color is used instead.
- This command draws an empty circle, ignoring line thickness and active fill patterns. To draw a circle with the current attributes, the command [Draw Styled Circle](#) can be used.

**Response:** [ACK]

**Example**

```
\iDC\D10
```

Draws a circle with a radius of 10 pixels to the current cursor position.

**See also:**

[Draw Line](#)  
[Draw Styled Circle](#)  
[Draw Rectangle](#)  
[Draw Ellipse](#)

[Get Cursor Position](#)  
[Set Inverse Mode](#)  
[Set Foreground Color](#)

## Draw Styled Circle

```
\i D c bits::mode word::radius
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 1, 3 ... 6	flags for the circle properties
radius	<a href="#">word</a>	1 ... max[width, height]	radius of the circle

Draws a circle using the currently selected line thickness with the radius *radius* and the current cursor position as center. The parameter *mode* allows to set several properties for the circle:

Bit	Description
1	drop shadow
3	filled with background color
4	use foreground color for filling (bit 3 must be set too)
5	filled with current fill pattern (bits 3 and 4 ignored if set)
6	draw without outline
7	alpha blending for shadow only

### Note

- Drawing the circle outline is done with the current border color (see [Set Border Color](#)). The shadow is drawn with the current border shadow color (see [Set Border Shadow Color](#)). Outline and shadow drawing is done with alpha blending if set before (see [Set Alpha](#)).
- By default, alpha blending is done for every activated feature (outline, filling, shadow). When *mode* bit 7 is set, the shadow will still be drawn with alpha blending, but outline and filling will have full opacity (refer to [Set Alpha](#)).
- The line style attribute is ignored for circle drawing (see [Set Line Style](#)).
- If the drop shadow bit is set, but the shadow offset (see [Set Shadow Offset](#)) leads to a negative center of the shadow circle, no shadow will be drawn.
- When filling with a pattern (bit 5), the fill pattern has to be set in advance via [Set Filling Color](#), [Set Filling Gradient](#) or [Set Filling Tile](#). When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).

**Response:** [ACK]

### Example

```
\iDc\x62\D125
```

Draws a filled circle without an outline with a radius of 125 pixels and a drop shadow.

Not supported by: DPC3020, DPC2060, DPC10xx

#### See also:

[Draw Line](#)  
[Draw Circle](#)  
[Draw Rectangle](#)  
[Draw Ellipse](#)  
[Set Border Color](#)  
[Set Border Shadow Color](#)  
[Set Line Thickness](#)  
[Set Filling Color](#)  
[Set Filling Gradient](#)  
[Set Filling Tile](#)  
[Set Alpha](#)

## Draw Ellipse

```
\i D Y bits::mode word::vertex_a word::vertex_b
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 1, 3 ... 6	flags for the ellipse properties
vertex_a	<a href="#">word</a>	1 ... display width	distance from center to intersection of ellipse and x-axis
vertex_b	<a href="#">word</a>	1 ... display height	distance from center to intersection of ellipse and y-axis

Draws an ellipse with the selected line thickness, using the current cursor position as center. The vertices *vertex\_a* and *vertex\_b* specify the dimensions, the parameter *mode* allows to set several properties for the ellipse:

Bit	Description
1	drop shadow
3	filled with background color
4	use foreground color for filling (bit 3 must be set too)
5	filled with current fill pattern (bits 3 and 4 ignored if set)
6	draw without outline
7	alpha blending for shadow only

#### Note

- Drawing the ellipse outline is done with the current border color (see [Set Border Color](#)). The shadow is drawn with the current border shadow color (see [Set Border Shadow Color](#)). Outline and shadow drawing is done with alpha blending if set before (see [Set Alpha](#)).



- By default, alpha blending is done for every activated feature (outline, filling, shadow). When *mode* bit 7 is set, the shadow will still be drawn with alpha blending, but outline and filling will have full opacity (refer to [Set Alpha](#)).
- The line style attribute is ignored for ellipse drawing (see [Set Line Style](#)).
- If the drop shadow bit is set, but the shadow offset (see [Set Shadow Offset](#)) leads to a negative center of the shadow ellipse, no shadow will be drawn.
- When filling with a pattern (*mode* bit 5), the fill pattern has to be set in advance via [Set Filling Color](#), [Set Filling Gradient](#) or [Set Filling Tile](#). When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).

**Response:** [ACK]

### Example

```
\iDY\x22\D250\D100
```

Draws a filled ellipse with the vertices 250 and 100 and a drop shadow.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Draw Line](#)  
[Draw Rectangle](#)  
[Draw Circle](#)  
[Draw Styled Circle](#)  
[Set Border Color](#)  
[Set Border Shadow Color](#)  
[Set Shadow Offset](#)  
[Set Line Thickness](#)  
[Set Filling Color](#)  
[Set Filling Gradient](#)  
[Set Filling Tile](#)  
[Set Alpha](#)

### Adjust Display

```
\i D A bits::mode
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 0 ... 4	flags for possible modifications

Adjusts the color values of all pixels on the screen. The value for any selected modification has to be set in advance and the corresponding *mode* bit must be true:

Bit	Description
0	brightness (see <a href="#">Set Brightness Adjustment</a> )

1	contrast (see <a href="#">Set Contrast Adjustment</a> )
2	hue (see <a href="#">Set Hue Adjustment</a> )
3	saturation (see <a href="#">Set Saturation Adjustment</a> )
4	invert

### Note

- All adjustment modifications can be combined and changed simultaneously with this command. The modifications are processed in the following order: invert - hue/saturation - brightness/contrast.
- This command can be used to dim or brighten the entire screen by adjusting brightness and contrast.

**Response:** [ACK]

### Example

```
\iDA\x3
```

Adjusts brightness and contrast of the entire screen, using the previously set values.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Adjust Display Area](#)  
[Set Brightness Adjustment](#)  
[Set Contrast Adjustment](#)  
[Set Hue Adjustment](#)  
[Set Saturation Adjustment](#)  
[Set Adjustment for Graphics](#)

## Adjust Display Area

```
\i D a bits::mode word::width word::height
```

Parameter	Type	Range	Description
mode	<a href="#">bits</a>	Bit 0 ... 4	flags for possible modifications
width	<a href="#">word</a>	0 ... display width (0 = full width)	horizontal value for the area to be adjusted
height	<a href="#">word</a>	0 ... display height (0 = full height)	vertical value for the area to be adjusted

Adjusts the color values of all pixels in the area specified by *width* and *height*, beginning from the current cursor position. The value for any selected modification has to be set in advance and the corresponding *mode* bit must be true:

Bit	Description
0	brightness (see <a href="#">Set Brightness Adjustment</a> )
1	contrast (see <a href="#">Set Contrast Adjustment</a> )
2	hue (see <a href="#">Set Hue Adjustment</a> )
3	saturation (see <a href="#">Set Saturation Adjustment</a> )
4	invert

### Note

- All adjustment modifications can be combined and changed simultaneously with this command. The modifications are processed in the following order: invert - hue/saturation - brightness/contrast.
- This command can be used to dim or brighten part of the screen by adjusting brightness and contrast of an area.
- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without sending an error.

**Response:** [ACK]

### Example

```
\iDa\xC\D200\D100
```

Adjusts hue and saturation in a 200x100 pixel area starting at the cursor position, using the previously set values.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Adjust Display](#)  
[Set Brightness Adjustment](#)  
[Set Contrast Adjustment](#)  
[Set Hue Adjustment](#)  
[Set Saturation Adjustment](#)  
[Set Adjustment for Graphics](#)  
[Set Cursor Position](#)  
[Get Cursor Position](#)

---

### Fill Display

```
\i D F
```

Fills the entire active viewport with the active fill pattern (see [Set Filling Color](#), [Set Filling Gradient](#) or [Set Filling Tile](#)). Afterwards the cursor position is set to 0/0.

**Note**

- If an alpha value other than 255 was set before (see [Set Alpha](#)), filling is done with the according opacity.
- When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).

**Response:** [ACK]

**Example**

```
\iDF
```

Fills the display with the active fill pattern.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Fill Display Area](#)  
[Set Filling Color](#)  
[Set Filling Gradient](#)  
[Set Filling Tile](#)  
[Set Alpha](#)  
[Set Adjustment for Graphics](#)

**Fill Display Area**

```
\i D f word::width word::height
```

Parameter	Type	Range	Description
width	<a href="#">word</a>	0 ... display width (0 = full width)	horizontal value for the area to be filled
height	<a href="#">word</a>	0 ... display height (0 = full height)	vertical value for the area to be filled

Fills an area of the screen with the active fill pattern (see [Set Filling Color](#), [Set Filling Gradient](#) or [Set Filling Tile](#)), beginning from the current cursor position.

**Note**

- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without sending an error.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), filling is done with the according opacity.
- When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).

**Response:** [ACK]

## Example

```
\iDf\D100\D50
```

Fills and area of 100x50 pixels starting from the cursor position.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Fill Display](#)  
[Set Filling Color](#)  
[Set Filling Gradient](#)  
[Set Filling Tile](#)  
[Set Alpha](#)  
[Set Adjustment for Graphics](#)  
[Get Cursor Position](#)  
[Set Cursor Position](#)

---

## LCD Graphics Commands

### General Information About Animated Graphics

All graphics – including the animated ones – are loaded into the Flash memory of the iLCD or stored on a MicroSD card via iLCD Manager XE. Although there can be an unlimited number of animated graphics, a maximum of 8 animated graphics can be animated at the same time on a LCD screen. While animated graphics are shown on the screen, any other command like drawing text or graphics to any cursor position can be carried out, as the iLCD's animation engine runs in the background of the iLCD Controller's firmware. Scrolling the screen while animations are active can be done, but will cause unwanted effects, as the position of animated graphics are not changed via the scroll screen commands.

When working with animated graphics, the first thing to do is to load an animated graphic to one of the 8 animation controls (referred to as *anim\_loc* in the following commands). All other animation related commands refer to the index of the animation control regardless of which animated graphic is loaded into it.

Please note that the single frames of an animated graphic may contain areas of the image, which are smaller in width and/or height than the complete image. This is exactly how animated GIFs work (iLCD Manager XE imports animated GIFs and takes care about the frames' size and position). So setting a certain frame number via the [Stop Animation and Set Frame Number](#) command may cause surprising effects when one does not keep in mind that only the (smaller) frame contents of the selected frame will be drawn, and not the whole image which is normally created by the sequence of consecutive frames.

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Display Local Graphic](#)
- [Display Graphic Area](#)
- [Load Animated Graphics](#)
- [Set Animation Coordinates to X/Y](#)
- [Set Animation Coordinates to Cursor Position](#)

- [Start or Restart Animation](#)
- [Stop Animation and Set Frame Number](#)
- [Stop \(Break\) Animation](#)
- [Set Animation Repetitions](#)
- [Erase Animation Image Area](#)
- [Erase Animation Frame Area](#)
- [Suspend Animation Engine](#)
- [Resume Animation Engine](#)
- [Move Animation to Frame To Frame](#)
- [Set Animation Background Color](#)
- [Set Animation Background Frame](#)
- [Set Animation Background Graphic](#)
- [Set Animation Background Screen](#)
- [Remove Animation Background](#)

---

## Display Local Graphic

by index:

`\i G word::graphic_index`

Parameter	Type	Range	Description
graphic_index	<a href="#">word</a>	0 ... max. graphic index	index of the graphic

by name:

`\i G by_name::graphic_name`

Parameter	Type	Range	Description
graphic_name	<a href="#">by_name</a> <a href="#">e</a>	ASCII chars (0x01 .. 0xFF)	0-terminated graphic name

by filename:

`\i G by_file::graphic_filename`

Parameter	Type	Range	Description
graphic_filename	<a href="#">by_file</a>	DOS filename (8.3 format)	name and path of the graphics file

Draws a graphic according to the graphic's *graphic\_index*, *graphic\_name* or *graphic\_filename* at the current cursor position.

### Note

- If no graphic with *graphic\_index*, *graphic\_name* or *graphic\_filename* is available, *[NACK]* is returned.
- Graphic alignment is taken into account (refer to [Set Graphic Alignment](#)). If the alignment area is smaller than the specified graphic, it will be cropped accordingly.

- When addressing by index, the graphic offset (see [Set Graphic Offset](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Set Graphic Name Prefix](#)) and suffix (refer to [Set Graphic Name Suffix](#)) are taken into account.
- The cursor position is not changed.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), filling is done with the according opacity.
- Any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).
- Additional to the [by\\_name](#) and [by\\_file](#) indices, the *graphic\_index* range from 0xFFE0 to 0xFFFF is reserved for internal use!
- When inverse mode is on, the graphic is shown in inverse mode.
- Graphics with transparencies are always drawn transparent, regardless of the transparent mode (see [Set Transparent Mode On/Off](#)).
- Monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.
- While graphics with a color depth of 8 bit occupy less space in the memory, 16 bit graphics are drawn faster as there is no need for converting calculations.
- To show a graphic from the MicroSD card, the file has to be converted into in the RII (Raw iLCD Graphics Image) format. To do that, use the "Save As" button on iLCD Manager XE's "Graphics" page and select the \*.rii filetype.
- When a graphic is stored on the MicroSD card automatically (by activating the option "Store graphics to SD card" in the "Project Settings" section of iLCD Manager XE's "Settings" page), it can still be addressed by its index or name as if they would be stored in flash memory. Graphics stored manually on the MicroSD card can only be loaded by their filenames.

**Response:** [ACK]

### Example

```
\iG\D1  
\iG\mGRAPHIC\0  
\iG\fdIR/FILE.RII\0
```

Displays the graphic with index 1 and graphic "GRAPHIC" from the on-board flash as well as graphic "FILE.RII" from the SD card's "DIR" folder at the current cursor position.

### See also:

[Get Graphic Info](#)  
[Set Graphic Alignment](#)  
[Display Graphic Area](#)  
[Set Graphic Offset](#)  
[Set Graphic Name Prefix](#)  
[Set Graphic Name Suffix](#)  
[Set Alpha](#)  
[Set Adjustment for Graphics](#)  
[Load Animated Graphics](#)

## Display Graphic Area

### by index:

```
\i g A word::start_x word::start_y word::width word::height
word::graphic_index
```

Parameter	Type	Range	Description
start_x	<a href="#">word</a>	0 ... graphic width - 1	horizontal offset in graphic
start_y	<a href="#">word</a>	0 ... graphic height - 1	vertical offset in graphic
width	<a href="#">word</a>	1 ... graphic width	width of the area to display
height	<a href="#">word</a>	1 ... graphic height	height of the area to display
graphic_index	<a href="#">word</a>	0 ... max. graphic index	index of the graphic

### by name:

```
\i g A word::offset_x word::offset_y word::width word::height
by_name::graphic_name
```

Parameter	Type	Range	Description
start_x	<a href="#">word</a>	0 ... graphic width - 1	horizontal offset in graphic
start_y	<a href="#">word</a>	0 ... graphic height - 1	vertical offset in graphic
width	<a href="#">word</a>	1 ... graphic width	width of the area to display
height	<a href="#">word</a>	1 ... graphic height	height of the area to display
graphic_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated graphic name

### by filename:

```
\i g A word::offset_x word::offset_y word::width word::height
by_file::graphic_filename
```

Parameter	Type	Range	Description
start_x	<a href="#">word</a>	0 ... graphic width - 1	horizontal offset in graphic
start_y	<a href="#">word</a>	0 ... graphic height - 1	vertical offset in graphic
width	<a href="#">word</a>	1 ... graphic width	width of the area to display
height	<a href="#">word</a>	1 ... graphic height	height of the area to display
graphic_filename	<a href="#">by_file</a>	DOS filename (8.3 format)	name and path of the graphics file

Draws a part of a graphic according to the graphic's *graphic\_index*, *graphic\_name* or *graphic\_filename* at the current cursor position.



## Note

- If no graphic with *graphic\_index*, *graphic\_name* or *graphic\_filename* is available, *[NACK]* is returned.
- If one of the offsets *x* or *y* exceeds the graphic dimensions, a *[NACK]* is returned. Coordinates scaling is not taken into account for these values, as they refer to the graphic and not to the screen (refer to [Set Column Coordinates Scaling](#) and [Set Row Coordinates Scaling](#)).
- Graphic alignment is taken into account (refer to [Set Graphic Alignment](#)). If the alignment area is smaller than the specified graphic area, it will be cropped accordingly.
- When addressing by index, the graphic offset (see [Set Graphic Offset](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Set Graphic Name Prefix](#)) and suffix (refer to [Set Graphic Name Suffix](#)) are taken into account.
- The cursor position is not changed.
- If an alpha value other than 255 was set before (see [Set Alpha](#)), filling is done with the according opacity.
- Any previously set adjustments for graphics are taken into account (see [Set Adjustment for Graphics](#)).
- Additional to the [by\\_name](#) and [by\\_file](#) indices, the *graphic\_index* range from 0xFFE0 to 0xFFFF is reserved for internal use!
- When inverse mode is on, the graphic is shown in inverse mode.
- Graphics with transparencies are always drawn transparent, regardless of the transparent mode (see [Set Transparent Mode On/Off](#)).
- Monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.
- While graphics with a color depth of 8 bit occupy less space in the memory, 16 bit graphics are drawn faster as there is no need for converting calculations.
- To convert an image to a Raw iLCD Graphics Image, use the "Save As" button on iLCD Manager XE's "Graphics" page and select the \*.rii filetype.
- Graphics stored manually on the MicroSD card can be loaded by their filenames, but not by their graphic names.

**Response:** [ACK]

## Example

```
\igA\D10\D20\D100\D200\D5  
\igA\D10\D20\D100\D200\mGRAPHIC\0  
\igA\D10\D20\D100\D200\fdIR/FILE.RII\0
```

Displays an area (100x200 pixel from pixel at position 10/20) of graphic with index 5 and graphic "GRAPHIC" from the on-board flash as well as graphic "FILE.RII" from the SD card's "DIR" folder at the current cursor position.

Not supported by: DPC3020, DPC2060, DPC10xx

## See also:

[Display Local Graphic](#)  
[Set Graphic Alignment](#)  
[Set Graphic Offset](#)  
[Set Graphic Name Prefix](#)  
[Set Graphic Name Suffix](#)  
[Set Alpha](#)  
[Set Adjustment for Graphics](#)

## Load Animated Graphics

### by index:

```
\i g L byte::anim_loc word::anim_graph_index
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
anim_graph_index	<a href="#">word</a>	1 ... max. graphic index	index of the animated graphic

### by name:

```
\i g L byte::anim_loc by_name::anim_graph_name
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
anim_graph_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated graphic name

### by filename:

```
\i g L byte::anim_loc by_file::anim_graph_filename
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
anim_graph_filename	<a href="#">by_file</a>	DOS filename (8.3 format)	name and path of the graphics file

Loads an animated graphic to the animation control.

### Note

- This command does not draw the animation, it just loads it to the animation control.
- The animation is placed at the current cursor position, graphic alignment is taken into account (refer to [Set Graphic Alignment](#)). If the alignment area is smaller than the specified animated graphic, a `[NACK]` is returned.
- Positioning, alignment and cropping is done with respect to the currently selected viewport (refer to [Select Viewport](#)). All re-positioning (refer to [Set Animation Coordinates to Cursor Position](#) and [Set Animation Coordinates to X/Y](#)) will be interpreted in this viewport too.
- The current background color (refer to [Set Background Color](#)) is stored in the animation control and subsequently used for frame removal if the disposal method [Remove Animation Background](#) is used.
- If no graphic with `anim_graph_index`, `anim_graph_name` or `anim_graph_filename` is available, `[NACK]` is returned.
- When addressing by index, the graphic offset (see [Set Graphic Offset](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Set Graphic Name Prefix](#)) and suffix (refer to [Set Graphic Name Suffix](#)) are taken into account.

- Additional to the [by\\_name](#) and [by\\_file](#) indices, the `anim_graph_index` range from 0xFFE0 to 0xFFFF is reserved for internal use!
- If the animation is intended to be run in both directions (e.g. for a gauge), a background has to be assigned to the animation engine (refer to [Set Animation Background Color](#), [Set Animation Background Frame](#), [Set Animation Background Graphic](#) or [Set Animation Background Screen](#)).
- To convert an animated image to a Raw iLCD Graphics Image, use the "Save As" button on iLCD Manager XE's "Graphics" page and select the \*.rii filetype.
- Animated graphics stored manually on the MicroSD card can be loaded by their filenames, but not by their graphic names.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### Example

```
\igL\1\D4
\igL\2\mANIMATION\0
\igL\3\f/ANIM.RII\0
```

Loads the animated graphic with index 4 into animation control #1, graphic "ANIMATION" into control #2 and graphic "ANIM.RII" from the SD card's root folder into control #3.

### See also:

[Get Graphic Info](#)  
[Set Graphic Alignment](#)  
[Set Graphic Offset](#)  
[Set Graphic Name Suffix](#)  
[Set Graphic Name Prefix](#)  
[Display Local Graphic](#)  
[Start or Restart Animation](#)  
[Move Animation To Frame](#)

---

## Set Animation Coordinates to X/Y

```
\i g K byte::anim_loc word::pos_x word::pos_y
```

Parameter	Type	Range	Description
<code>anim_loc</code>	<a href="#">byte</a>	0 ... 7	index of the animation control
<code>pos_x</code>	<a href="#">word</a>	0 ... display width - 1	horizontal position of the graphic
<code>pos_y</code>	<a href="#">word</a>	0 ... display height - 1	vertical position of the graphic

Allows to specify coordinates for a graphic loaded into the animation control.

**Note**

- Graphic alignment is taken into account (refer to [Set Graphic Alignment](#)). If the alignment area is smaller than the specified animated graphic, a `[NACK]` is returned.
- The coordinates are assigned to the animation control independent from the currently selected viewport. They will be interpreted with respect to the origin of the viewport in which the animation was loaded (refer to [Load Animated Graphics](#)).
- Setting the coordinate without stopping an already running animation may cause unwanted effects.
- If `anim_loc` has never been loaded with an animated graphic before or `x` and/or `y` exceed the display width/height, a `[NACK]` is returned.

**Response:** `[ACK]`

**Example**

```
\igK\0\D220\D150
```

This example will access the animated graphic referenced in the animation control with index 0 and set its drawing coordinates to `x = 220` pixels and `y = 150` pixels.

**See also:**

[Load Animated Graphics](#)  
[Display Local Graphic](#)

**Set Animation Coordinates to Cursor Position**

```
\i g k byte::anim_loc
```

Parameter	Type	Range	Description
<code>anim_loc</code>	<a href="#">byte</a>	0 ... 7	index of the animation control

Sets the coordinates for an animated graphic stored in the animation control to the current cursor position.

**Note**

- Graphic alignment is taken into account (refer to [Set Graphic Alignment](#)). If the alignment area is smaller than the specified animated graphic, a `[NACK]` is returned.
- The coordinates are assigned to the animation control independent from the currently selected viewport. They will be interpreted with respect to the origin of the viewport in which the animation was loaded (refer to [Load Animated Graphics](#)).
- Setting the coordinate without stopping an already running animation may cause unwanted effects.
- If `anim_loc` has never been loaded with an animated graphic before a `[NACK]` is returned.

**Response:** `[ACK]`

## Example

```
\igk\2
```

This example sets the coordinates for the graphic referenced with index 2 by the animation control to the current cursor position.

### See also:

[Set Animation Coordinates to X/Y](#)  
[Load Animated Graphics](#)  
[Display Local Graphic](#)

---

## Start or Restart Animation

```
\i g S byte::anim_loc
```

Parameter	Type	Range	Description
anim_loc	byte	0 ... 7	index of the animation control

Starts or restarts (if previously stopped) the animated graphic loaded to the animation control referred by *anim\_loc*.

### Note

- If the animated graphic does not run endlessly (that means the number of repetitions are greater than 0), the internal repetition counter is reset before the animation is (re)started.
- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.
- If the animation control *anim\_loc* is already running, this command has no effect.
- By default, all animations are not running. Hence all animations are stopped on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#). The loaded animation controls (refer to [Load Animated Graphics](#)) as well as set backgrounds (refer to [Set Animation Background Color](#), [Set Animation Background Frame](#), [Set Animation Background Graphic](#) or [Set Animation Background Screen](#)) are not deleted by this command.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

## Example

```
\igS\4
```

Starts the animation loaded into animation control 4.

### See also:

[Set Animation Repetitions](#)  
[Stop Animation and Set Frame Number](#)

[Stop \(Break\) Animation](#)  
[Suspend Animation Engine](#)

## Stop Animation and Set Frame Number

```
\i g F byte::anim_loc word::frame
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
frame	<a href="#">word</a>	0 ... number of frames - 1	frame of the animation

Stops the animation referenced via *anim\_loc* and shows the specified *frame*.

### Note

- If *frame* exceeds the number of existing frames or *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.
- The animation doesn't have to run to show a specific frame.
- The command [Set Animation Coordinates to X/Y](#) can be used to modify the position where the frame is drawn.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### Example

```
\igF\2\4
```

Stops the animation control with index 2 and displays frame 4 of the animation.

### See also:

[Set Animation Repetitions](#)  
[Set Animation Coordinates to X/Y](#)  
[Stop \(Break\) Animation](#)  
[Suspend Animation Engine](#)

## Stop (Break) Animation

```
\i g B byte::anim_loc
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	A, 0 ... 7	index of the animation control (A = all)

Stops the animation specified via *anim\_loc*.

### Note

- When *anim\_loc* is 'A' (0x41), this command stops all running animations.
- Animations stopped by this command can be restarted via the [Start or Restart Animation](#) command exactly where they have been stopped.
- If *anim\_loc* is other than 'A' and has never been loaded with an animated graphic before, a *[NACK]* is returned.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### Example

```
\igB\2
```

Stops the animation stored under position 2 at its current frame.

### See also:

[Suspend Animation Engine](#)  
[Start or Restart Animation](#)  
[Resume Animation Engine](#)

## Set Animation Repetitions

```
\i g R byte::anim_loc word::repeat
```

Parameter	Type	Range	Description
<i>anim_loc</i>	<a href="#">byte</a>	0 ... 7	index of the animation control
<i>repeat</i>	<a href="#">word</a>	0 ... 65535	number for animation repetitions (0 = endless)

Sets the repetitions for a specific animation.

### Note

- Normally the repetitions of an animated graphic are set via iLCD Manager XE, but the number of repetitions until the animation stops automatically can be overwritten with this command.
- A repeat value of 0 sets unlimited repetitions.
- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.

**Response:** [ACK]

### Example

```
\igR\0\D100
```

This will cause the animation 0 to be repeated 100 times.

**See also:**

[Start or Restart Animation](#)

[Stop Animation and Set Frame Number](#)

### **Erase Animation Image Area**

```
\i g E byte::anim_loc
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control

Erases the area covered by the complete animated image.

**Note**

- By default, the animation area is filled with the current background color. If any background was assigned to the animation control however, the animation area is restored to the assigned background (refer to [Set Animation Background Color](#), [Set Animation Background Frame](#), [Set Animation Background Graphic](#) or [Set Animation Background Screen](#)).
- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.

**Response:** [ACK]

**Example**

```
\igE\2
```

Erases the area on the screen covered by the complete animation 2 referenced by the animation control.

**See also:**

[Erase Animation Frame Area](#)

### **Erase Animation Frame Area**

```
\i g e byte::anim_loc
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control



Erases the area covered by the current frame of the animated image.

### Note

- By default, the frame area is filled with the current background color. If any background was assigned to the animation control however, the frame area is restored to the assigned background (refer to [Set Animation Background Color](#), [Set Animation Background Frame](#), [Set Animation Background Graphic](#) or [Set Animation Background Screen](#)).
- The area may be smaller than the area of the complete image and the upper left corner of the area to be erased is given by the X/Y offset of the frame.
- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.

**Response:** [ACK]

### Example

```
\ige\2
```

Erases the area on the screen covered by the current frame of animation 2 referenced by the animation control.

### See also:

[Erase Animation Image Area](#)

---

## **Suspend Animation Engine**

```
\i g s
```

Allows to stop all animated graphics at the current frame by stopping the animation engine.

### Note

- Compared to the [Stop \(Break\) Animation](#) command with parameter 'A', this command allows for a more convenient way of resuming all the animations by simple using the [Resume Animation Engine](#) command from the point where they have stopped.
- By default, the animation engine is running. Hence it will be automatically started on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- Using this command is highly recommended before reading the LCD's screen content via the [Read Scan Line](#) command, as scrambled screen contents may be read when animations are active!
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### See also:

[Stop \(Break\) Animation](#)  
[Resume Animation Engine](#)

## **Resume Animation Engine**

```
\i g r
```

Resumes the animation engine when previously stopped via [Suspend Animation Engine](#).

### **Note**

- This command is especially useful when used in conjunction with the [Suspend Animation Engine](#) command to conveniently stop and resume all animations on the screen.
- By default, the animation engine is running. Hence it will be automatically started on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### **Example**

```
\igr
```

Resumes the animation engine and all animations at the frame they have stopped.

### **See also:**

[Suspend Animation Engine](#)

## **Set Animation Background Color**

```
\i g b C byte::anim_loc color::color_value
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
color_value	<a href="#">color</a>	0x000000 ... 0xFFFFFFFF	color value for the animation background

Assigns the background color *color\_value* to the animation engine *anim\_loc*.

### **Note**

- When a background color is assigned to an animation control, every frame is filled with the specified background color before the next one is drawn.
- If the animation was never drawn before, the whole animation area is filled with the background color before the first frame is drawn.
- Restoring frames to background allows an animation to run backwards (refer to [Move Animation To Frame](#)), but slows it down a bit.
- The background is filled with the assigned background color when using the commands [Erase Animation Image Area](#) or [Erase Animation Frame Area](#) as well.

- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### Example

```
\igbc\5\#FFFF00
```

This will assign a yellow color to the animation control #5.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Remove Animation Background](#)  
[Set Animation Background Graphic](#)  
[Set Animation Background Frame](#)  
[Set Animation Background Screen](#)  
[Erase Animation Image Area](#)  
[Erase Animation Frame Area](#)

## Set Animation Background Frame

```
\i g b F byte::anim_loc word::frame
```

Parameter	Type	Range	Description
<i>anim_loc</i>	<a href="#">byte</a>	0 ... 7	index of the animation control
<i>frame</i>	<a href="#">word</a>	0 ... number of frames - 1	background frame of the animation

Specifies the frame *frame* as background for the animation engine *anim\_loc*.

### Note

- When a background frame is assigned to an animation control, every frame is restored to the underlying part of the background before the next one is drawn.
- As soon as a background frame is assigned, this frame will be skipped in running animations. It can still be set manually though (refer to [Stop Animation and Set Frame Number](#)).
- If the background frame doesn't cover the entire animation area, only the part where the areas of the background and the previous frame overlap will be restored.
- If the animation was never drawn before, the whole animation area is filled with the background frame before the first frame is drawn.
- Restoring frames to background allows an animation to run backwards (refer to [Move Animation To Frame](#)), but slows it down a bit.
- The background is restored to the assigned background frame when using the commands [Erase Animation Image Area](#) or [Erase Animation Frame Area](#) as well.
- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### Example

```
\igbF\7\D12
```

This will specify frame #12 as the background frame for animation control #5.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Remove Animation Background](#)  
[Set Animation Background Color](#)  
[Set Animation Background Graphic](#)  
[Set Animation Background Screen](#)  
[Erase Animation Image Area](#)  
[Erase Animation Frame Area](#)

## Set Animation Background Graphic

### by index:

```
\i g b G byte::anim_loc word::offset_x word::offset_y  
word::graphic_index
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
offset_x	<a href="#">word</a>	0 ... graphic width - 1	horizontal offset in background graphic
offset_y	<a href="#">word</a>	0 ... graphic height - 1	vertical offset in background graphic
graphic_index	<a href="#">word</a>	0 ... max. graphic index	index of the background graphic

### by name:

```
\i g b G byte::anim_loc word::offset_x word::offset_y  
by_name::graphic_name
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
offset_x	<a href="#">word</a>	0 ... graphic width - 1	horizontal offset in background graphic
offset_y	<a href="#">word</a>	0 ... graphic height - 1	vertical offset in background graphic
graphic_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated graphic name

**by filename:**

```
\i g b G byte::anim_loc word::offset_x word::offset_y
by_file::graphic_filename
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
offset_x	<a href="#">word</a>	0 ... graphic width - 1	horizontal offset in background graphic
offset_y	<a href="#">word</a>	0 ... graphic height - 1	vertical offset in background graphic
graphic_filename	<a href="#">by_file</a>	DOS filename (8.3 format)	name and path of the graphics file

Assigns the background graphic specified by *graphic\_index*, *graphic\_name* or *graphic\_filename* to the animation engine *anim\_loc*.

**Note**

- When a background graphic is assigned to an animation control, every frame is restored to the underlying part of the background before the next one is drawn.
- If the animation was never drawn before, the whole animation area is filled with the according part of background graphic before the first frame is drawn.
- If no graphic with *graphic\_index*, *graphic\_name* or *graphic\_filename* is available or the *anim\_loc* has never been loaded with an animated graphic before, *[NACK]* is returned.
- When addressing by index, the graphic offset (see [Set Graphic Offset](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Set Graphic Name Prefix](#)) and suffix (refer to [Set Graphic Name Suffix](#)) are taken into account.
- Background graphics can be larger than the animation area. In this case, *offset\_x* and *offset\_y* have to be set to the offset between the origin of the background graphic and the origin of the animation area.
- Restoring frames to background allows an animation to run backwards (refer to [Move Animation To Frame](#)), but slows it down a bit. Even more so if animated or background graphic have to be loaded from SD card.
- The background is restored to the assigned background graphic when using the commands [Erase Animation Image Area](#) or [Erase Animation Frame Area](#) as well.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

**Example**

```
\igbG\1\D0\D0\D7
\igbG\2\D10\D20\mGRAPHIC\0
\igbG\3\D0\D0\fdIR/FILE.RII\0
```

These commands will assign:

- graphic with index #7 without offset to the animation control #1
- part of the graphic with name "GRAPHIC" starting from position 10/20 to the animation control #2

- graphic with filename "FILE.RII" from the SD card's "DIR" folder without offset to the animation control #3

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Remove Animation Background](#)  
[Set Animation Background Color](#)  
[Set Animation Background Frame](#)  
[Set Animation Background Screen](#)  
[Erase Animation Image Area](#)  
[Erase Animation Frame Area](#)

## **Set Animation Background Screen**

```
\i g b S byte::anim_loc word::pos_x word::pos_y byte::screen
```

Parameter	Type	Range	Description
anim_loc	byte	0 ... 7	index of the animation control
pos_x	word	0 ... display width - 1	horizontal position on background screen
pos_y	word	0 ... display height - 1	vertical position on background screen
screen	byte	M, 0 ... number of screens - 1	index of the background screen (M = main screen)

Assigns the background screen *screen* to the animation engine *anim\_loc*.

**Note**

- When a background screen is assigned to an animation control, every frame is restored to the underlying part of the background before the next one is drawn.
- If the animation was never drawn before, the whole animation area is filled with the content of the background screen before the first frame is drawn.
- If the animation was never drawn before, the whole animation area is filled with the content of the background screen before the first frame is drawn.
- The background can be located anywhere on the background screen. *pos\_x* and *pos\_y* can be used to specify the position of the background on the screen.
- Restoring frames to background allows an animation to run backwards (refer to [Move Animation To Frame](#)), but slows it down a bit.
- The background is restored to the assigned screen content when using the commands [Erase Animation Image Area](#) or [Erase Animation Frame Area](#) as well.
- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

## Example

```
\igbS\1\D20\D30\2
```

This will specify the content of screen #2 starting from cursor position 20/30 as background for the animation control #1.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Remove Animation Background](#)  
[Set Animation Background Color](#)  
[Set Animation Background Graphic](#)  
[Set Animation Background Frame](#)  
[Erase Animation Image Area](#)  
[Erase Animation Frame Area](#)

## Remove Animation Background

```
\i g b N byte::anim_loc
```

Parameter	Type	Range	Description
anim_loc	byte	0 ... 7	index of the animation control

Removes any previously assigned background from the animation engine *anim\_loc*.

### Note

- If *anim\_loc* has never been loaded with an animated graphic before, a *[NACK]* is returned.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

## Example

```
\igbN\4
```

This will remove any previously set background from the animation control #4.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Set Animation Background Color](#)  
[Set Animation Background Graphic](#)  
[Set Animation Background Frame](#)  
[Set Animation Background Screen](#)

## Move Animation To Frame

```
\i g M byte::anim_loc word::frame_idx bool::backwards
```

Parameter	Type	Range	Description
anim_loc	<a href="#">byte</a>	0 ... 7	index of the animation control
frame	<a href="#">word</a>	0 ... number of frames - 1	frame of the animation
backwards	<a href="#">bool</a>	0 ... 1	determines whether the animation runs forwards (0) or backwards (1)

Start the animated graphic loaded to the animation control referred by *anim\_loc* and stop again when the specified *frame* is reached. This can be done in both directions, dependent on the value of *backwards*.

### Note

- Given correct usage, this command returns an [ACK] immediately. Be aware that this acknowledgement means that the animation has begun moving to the desired frame, not that the frame has already been reached.
- Generally, animated gifs are not able to run in both directions. To run an animation backwards, a background must be assigned to the animation engine (refer to [Set Animation Background Color](#), [Set Animation Background Frame](#), [Set Animation Background Graphic](#) or [Set Animation Background Screen](#)).
- Specifying a frame of 0xFFFF will let the animation run indefinitely (same as [Start or Restart Animation](#)) in forward or backward direction.
- If the frame *frame* is currently showing, an [ACK] is returned but the animation stays as it is.
- If the animation engine has never been drawn before it will start from the first (running forward) or last frame (running backwards).
- If *anim\_loc* has never been loaded with an animated graphic before, a [NACK] is returned.
- Find general information about animated graphics in chapter [LCD Graphics Commands](#).

**Response:** [ACK]

### Example

```
\igM\3\D5\1
```

Starts the animation loaded into animation control #3 in backwards direction and stops at frame #5.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[LCD Graphics Commands](#)  
[Load Animated Graphics](#)  
[Start or Restart Animation](#)  
[Get Graphic Info](#)



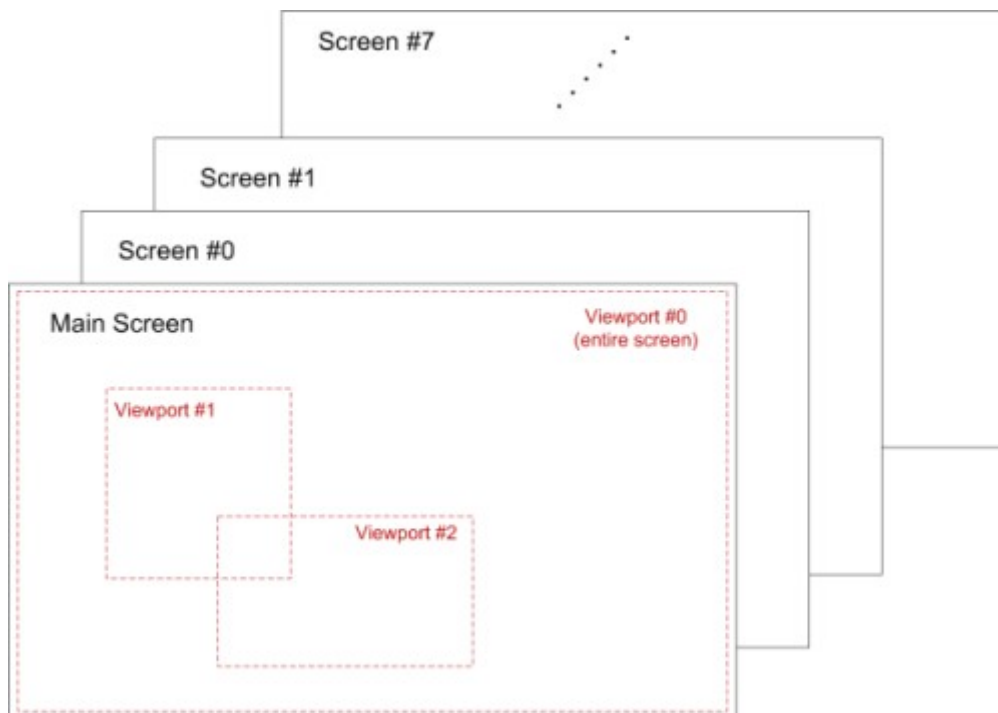
## Screen Memory Related Commands

Color iLCD Controllers DPC3080 and higher contain some RAM for storing complete screen contents. This allows the user to select a screen memory area to draw to (draw screen) and a different one that is displayed (view screen). When changing the view screen, the previously drawn content is displayed instantly without any visible buildup. This can be used in turn to create multiple screens or pages with minimum switching delays.

Color iLCDs need a rather high amount of memory to store a screen (width \* height \* 2 bytes), so these models use an external RAM of typical 8 MByte to store screens into. When using a 1024 x 600 pixel LCD, four user accessible screens are available. For all other models with iLCD Controller DPC3080 and higher, the number of screens is limited to a maximum of eight. To retrieve the number of available screens, the [Get # of Screens](#) command can be used.

iLCDs with the DPC3050 controller are not equipped with any external RAM, so the command [Get # of Screens](#) will return 0 for these models. Therefore, the commands in this chapter except [Copy Screen Area](#) will always return with a `[NACK]`.

Every iLCD additionally provides a main screen, which can be addressed by the screen index 'M' (ASCII code 0x4D). The main viewport - which represents the entire screen area - of each screen is defined as 0.



Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Get # of Screens](#)
- [Set View Screen](#)
- [Get View Screen Parameters](#)
- [Set Draw Screen](#)

- [Get Draw Screen Parameters](#)
- [Copy Screen Area](#)
- [Copy Screen To](#)
- [Copy Screen From](#)
- [Paint Screen From](#)
- [Invert Screen](#)
- [Scroll Up Screen](#)
- [Scroll Down Screen](#)
- [Scroll Left Screen](#)
- [Scroll Right Screen](#)
- [Set Height of Screen](#)
- [Set Width of Screen](#)

## Get # of Screens

`\i M S ?`

Returns the number of available screens excluding the main screen.

### Note

- Before calling any of the screen memory related commands, the number of available screens should be retrieved.
- The number returned may be 0 when using large LCDs and, if so, the screen memory related commands cannot be used then.
- The number of maximum available screens is limited to 8 regardless of the controller RAM available.

**Response:** [ACK] number [ACK]

Parameter	Type	Description
number	<a href="#">byte</a>	number of screens

### Response Example

[ACK] [08] [ACK]

There are 8 screens available on this iLCD model.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)  
[Set View Screen](#)  
[Set Draw Screen](#)

## Set View Screen

```
\i M V byte::index
```

Parameter	Type	Range	Description
index	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the view screen (M = main screen)

Selects the screen with index *screen* as the current view screen.

### Note

- When changing the view screen, its content is displayed immediately.
- In case of a touchscreen event, only touch fields defined on the currently active view screen are evaluated. Where touch fields are defined is dependent on the active draw screen when the command is carried out.
- If screen *index* was never activated as a draw screen before or *index* exceeds the available screens, *[NACK]* is returned.
- The default value for *screen* is 'M' (main screen). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iMV\3
```

Sets screen 3 as view screen and shows its content on the display.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)  
[Get View Screen Parameters](#)  
[Set Draw Screen](#)

## Get View Screen Parameters

```
\i M ? V
```

Returns the parameters of the currently active view screen.

### Note

- The active viewport index is only relevant when the according screen is activated as draw screen. In this case, the reported viewport will be active immediately after switching draw screens.

**Response:** [ACK] screen viewport vp\_ori text\_ori pos\_x pos\_y [ACK]

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the active view screen (M = main screen)
viewport	<a href="#">byte</a>	0 ... 8	index of the screen's active viewport
vp_ori	<a href="#">byte</a>	0 ... 3	orientation of the viewport relative to the screen orientation
text_ori	<a href="#">byte</a>	0 ... 3	orientation of the text/graphic items relative to the viewport orientation
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal coordinate of the cursor position related to the viewport
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical coordinate of the cursor position related to the viewport

### Response Example

```
[ACK] [03] [05] [02] [00] [01] [2C] [00] [00] [ACK]
```

The currently active view screen has the index of 3, viewport 5 of this screen is selected, the viewport is orientated in landscape mode upside down (180°), the text/graphic orientation is unchanged, and the cursor is currently at the coordinates x = 300 (0x12C) and y = 0 (0x0).

### Note

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Set View Screen](#)

[Set Draw Screen](#)

[Get Draw Screen Parameters](#)

### Set Draw Screen

```
\i M D byte::screen
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the draw screen (M = main screen)

Selects the screen with index *screen* as the current draw screen.

**Note**

- Cursor position, active viewport and all attributes (see [LCD Attribute Commands](#)) are restored to the state when this draw screen was lastly active.
- All subsequently issued drawing commands are sent to this screen until another one is activated. To show the results of these commands on the display, the according screen has to be activated as view screen.
- The default value for *screen* is 'M' (main screen). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iMD\0
```

Sets screen 0 as the current draw screen.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[Screen Memory Related Commands](#)  
[Get Draw Screen Parameters](#)  
[Set View Screen](#)

**Get Draw Screen Parameters**

```
\i M ? D
```

Returns the parameters of the currently active draw screen.

**Response:** [ACK] screen viewport vp\_ori text\_ori pos\_x pos\_y [ACK]

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the active draw screen (M = main screen)
viewport	<a href="#">byte</a>	0 ... 8	index of the screen's active viewport
vp_ori	<a href="#">byte</a>	0 ... 3	orientation of the viewport relative to the screen orientation
text_ori	<a href="#">byte</a>	0 ... 3	orientation of the text/graphic items relative to the viewport orientation
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal coordinate of the cursor position related to the viewport
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical coordinate of the cursor position related to the viewport

## Response Example

```
[ACK]M[05][00][01][01][2C][00][00][ACK]
```

The currently active draw screen is the main screen ('M'), viewport 5 of this screen is selected, the viewport orientation is unchanged, text and graphics are rotated by 90° and the cursor is currently at the coordinates x = 300 (0x12C) and y = 0 (0x0).

### Note

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Set Draw Screen](#)

[Set View Screen](#)

[Get View Screen Parameters](#)

[LCD Attribute Commands](#)

## Copy Screen Area

```
\i M A word::width word::height byte::screen word::pos_x word::pos_y
```

Parameter	Type	Range	Description
width	<a href="#">word</a>	1 ... display width	width of the area to copy
height	<a href="#">word</a>	1 ... display height	height of the area to copy
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to copy to (M = main screen)
pos_x	<a href="#">word</a>	0 ... display width - 1	horizontal position to copy the content to
pos_y	<a href="#">word</a>	0 ... display height - 1	vertical position to copy the content to

Copies the content of an area of the currently active draw screen starting from the current cursor position.

### Note

- If *index* exceeds the available screens or *width* or *height* exceed the screen size, *[NACK]* is returned.
- When the selected area exceeds the screen, it is truncated accordingly.
- *index* can also be set to the currently active draw screen, allowing copying an area to a different position on the same screen.
- Even if the target screen was never selected as draw screen before, it is possible to select it as view screen after copying an area to it.
- If using a DPC3050 iLCD controller, index has to be M in any case.

**Response:** [ACK]

## Example

```
\iMA\D25\D45M\D500\D250
```

This example copies an area of 25 pixels width and 45 pixels height (starting from the current cursor position) from the currently active draw screen to main screen at coordinates x = 500 and y = 250 pixels.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Copy Screen To](#)

[Copy Screen From](#)

[Paint Screen From](#)

[Set View Screen](#)

## Copy Screen To

```
\i M S S byte::screen
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to copy content to (M = main screen)

Captures the current draw screen's content and copies it to the screen referenced by the index *screen*.

### Note

- If *index* exceeds the available screens, a *[NACK]* is returned.
- The currently set inverse mode (see [Set Inverse Mode](#)) is ignored.
- When capturing to a screen that has been cropped with [Set Height of Screen](#) or [Set Width of Screen](#), the width and height of this screen are restored to the full display size and it can be selected as view or draw screen again.
- [Copy Screen From](#) can be used to restore a screen copied with this command.

**Response:** [ACK]

## Example

```
\iMSS\2
```

This example copies the content of the currently active draw screen to screen 2.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[Screen Memory Related Commands](#)  
[Syntax used in iLCD Manager XE](#)  
[Copy Screen From](#)

**Copy Screen From**

```
\i M S C byte::screen
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to copy from (M = main screen)

Restores the content of a screen specified by the index *screen* to the current draw screen.

**Note**

- This command is only available when the main viewport (index 0) of the active draw screen is activated. If another viewport is active, *[NACK]* is returned.
- The currently set inverse mode (see [Set Inverse Mode](#)) is ignored (different to [Paint Screen From](#)).
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.
- This command can be used to restore a screen previously copied with [Copy Screen To](#).

**Response:** *[ACK]*

**Example**

```
\iMSC\2
```

Provided the screen with index 2 was activated, its contents is copied to the active draw screen.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[Screen Memory Related Commands](#)  
[Copy Screen To](#)  
[Paint Screen From](#)  
[Set Draw Screen](#)  
[Set View Screen](#)



## Paint Screen From

```
\i M S P byte::screen
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to paint from (M = main screen)

Paints the content of a screen specified by the index *screen* to the current draw screen at the current cursor position.

### Note

- This command is only available when the main viewport (index 0) of the active draw screen is activated. If another viewport is active, *[NACK]* is returned.
- The currently set inverse mode (see [Set Inverse Mode](#)) is taken into account (different to [Copy Screen From](#)).
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.

**Response:** [ACK]

### Example

```
\iMSP\2
```

Provided the screen with index 2 was activated, its contents is copied to the current cursor position on the active draw screen.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Copy Screen To](#)

[Set Inverse Mode](#)

[Set Draw Screen](#)

[Set View Screen](#)

## Invert Screen

```
\i M S I byte::screen
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to invert (M = main screen)

Inverts the contents of the screen indexed by *screen*.

#### Note

- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.

**Response:** [ACK]

#### Example

```
\iMSI\2
```

Provided the screen with index 2 was activated, its contents will be inverted.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

#### See also:

[Screen Memory Related Commands](#)

[Invert Display](#)

### Scroll Up Screen

```
\i M S U byte::screen word::scroll_y
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
scroll_y	<a href="#">word</a>	0 ... display height - 1	vertical distance for the screen to scroll up

Scrolls the screen with index *screen* *scroll\_y* pixels up.

#### Note

- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.
- The new rows at the bottom are drawn in the current background color.
- The currently set inverse mode (see [Set Inverse Mode](#)) is taken into account accordingly.

**Response:** [ACK]

#### Example

```
\iMSU\3\D50
```

Provided the screen with index 3 was activated, its contents will be scrolled 50 pixels up.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[Screen Memory Related Commands](#)

[Set Inverse Mode](#)

[Set Background Color](#)

[Scroll Down Screen](#)

[Scroll Left Screen](#)

[Scroll Right Screen](#)

## Scroll Down Screen

```
\i M S D byte::screen word::scroll_y
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
scroll_y	<a href="#">word</a>	0 ... display height - 1	vertical distance for the screen to scroll down

Scrolls the screen with index *screen* *scroll\_y* pixels down.

**Note**

- The new rows at the top are drawn in the current background color.
- The currently set inverse mode (see [Set Inverse Mode](#)) is taken into account accordingly.
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.

**Response:** [ACK]

**Example**

```
\iMSD\2\D150
```

Provided the screen with index 2 was activated, its contents will be scrolled 150 pixels down.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[Screen Memory Related Commands](#)

[Set Inverse Mode](#)

[Set Background Color](#)

[Scroll Up Screen](#)

[Scroll Left Screen](#)

[Scroll Right Screen](#)

## Scroll Left Screen

```
\i M S L byte::screen word::scroll_x
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
scroll_x	<a href="#">word</a>	0 ... display width - 1	horizontal distance for the screen to scroll left

Scrolls the screen with index *screen* *scroll\_x* pixels to the left.

### Note

- The new rows at the right are drawn in the current background color.
- The currently set inverse mode (see [Set Inverse Mode](#)) is taken into account accordingly.
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.

**Response:** [ACK]

### Example

```
\iMSL\5\D450
```

Provided the screen with index 5 was activated, its contents will be scrolled 450 pixels to the left.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Set Inverse Mode](#)

[Set Background Color](#)

[Scroll Up Screen](#)

[Scroll Down Screen](#)

[Scroll Right Screen](#)

## Scroll Right Screen

```
\i M S R byte::screen word::scroll_x
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
scroll_x	<a href="#">word</a>	0 ... display width - 1	horizontal distance for the screen to scroll right

Scrolls the screen with index *screen* *scroll\_x* pixels to the right.

#### Note

- The new rows at the left are drawn in the current background color.
- The currently set inverse mode (see [Set Inverse Mode](#)) is taken into account accordingly.
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.

**Response:** [ACK]

#### Example

```
\iMSR\1\D88
```

Provided the screen with index 1 was activated, its contents will be scrolled 88 pixels to the right.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

#### See also:

[Screen Memory Related Commands](#)

[Set Inverse Mode](#)

[Set Background Color](#)

[Scroll Up Screen](#)

[Scroll Down Screen](#)

[Scroll Left Screen](#)

## Set Height of Screen

```
\i M S H byte::screen word::height
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to set the height (M = main screen)
height	<a href="#">word</a>	0 ... display height	value for the height to be set

Sets the height of the screen indexed by *screen*.

#### Note

- The height of the main screen (index 'M') is not modifiable.

- Width (refer to [Set Width of Screen](#)) and height of a screen are taken into account when being copied (refer to [Copy Screen From](#)) or painted (refer to [Paint Screen From](#)).
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.
- If the height of an image is decreased the bottom part of the image will be cut.
- When cropping a screen, this screen can not be selected as draw or view screen any longer. To make a screen selectable again, the width and height have to be restored to the full display size.

**Response:** [ACK]

### Example

```
\iMSH\1\D52
```

Provided the screen with index 1 was activated, its height will be set to 52 pixels.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Set Width of Screen](#)

[Copy Screen From](#)

[Paint Screen From](#)

## Set Width of Screen

```
\i M S W byte::screen word::width
```

Parameter	Type	Range	Description
screen	<a href="#">byte</a>	M, 0 ... number of screens - 1	index of the screen to set the height (M = main screen)
width	<a href="#">word</a>	0 ... display width	value for the width to be set

Sets the width of the screen with index *screen*.

### Note

- The width of the main screen (index 'M') is not modifiable.
- Width and height (refer to [Set Height of Screen](#)) of a screen are taken into account when being copied (refer to [Copy Screen From](#)) or painted (refer to [Paint Screen From](#)).
- If the screen with index *screen* was never activated as a draw screen before or *screen* exceeds the available screens, *[NACK]* is returned.
- If the width of a screen is decreased, the right hand part of the screen will be cut.
- When cropping a screen, this screen can not be selected as draw or view screen any longer. To make a screen selectable again, the width and height have to be restored to the full display size.

**Response:** [ACK]

### Example

```
\iMSW\4\D123
```

Provided the screen with index 4 was activated, its width will be set to 123 pixels.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Screen Memory Related Commands](#)

[Set Height of Screen](#)

[Copy Screen From](#)

[Paint Screen From](#)

## Cursor Memory Related Commands

The iLCD Controller contains 8 memory locations for saving the cursor position and the current attributes (see [LCD Attribute Commands](#)). This enables the user to easily save and restore the current state of commonly used screen locations when outputting text to the display.

Find following commands in this chapter as well as in the category "LCD Screen Memory..." when using the parameter completion feature of iLCD Manager XE:

- [Save Cursor & Attributes to Memory](#)
- [Restore Cursor & Attributes from Memory](#)

### Save Cursor & Attributes to Memory

```
\i M C S byte::index
```

Parameter	Type	Range	Description
index	<a href="#">byte</a>	0 ... 7	index of the storage location

Saves current cursor position, viewport and attributes (see [LCD Attribute Commands](#)) to memory position *index*.

### Example

```
\iMCS\2
```

Saves current cursor and attributes to the memory at the *index* 2.

**See also:**

[Restore Cursor & Attributes from Memory](#)  
[Cursor Memory Related Commands](#)

---

**Restore Cursor & Attributes from Memory**

```
\i M C C byte::index
```

Parameter	Type	Range	Description
index	<a href="#">byte</a>	0 ... 7	index of the storage location

Restores cursor position, viewport and attributes (see [LCD Attribute Commands](#)) from memory position *index*.

**Note**

- If no previous saving has been done to this *index*, *[NACK]* is returned.

**Response:** *[ACK]*

**Example**

```
\iMCC\3
```

Restores the cursor and attributes from the memory at the *index* 3.

**See also:**

[Save Cursor & Attributes to Memory](#)  
[Cursor Memory Related Commands](#)

---

**Macro Related Commands**

Macros are a very powerful feature of the iLCD panel controllers and allow you to condense multiple commands into one macro which can be executed at any time via a command.

Macros can even call other macros with the maximum nesting depth of 8. If the nesting depth is exceeded, the macro will be stopped and a *[NACK]* is returned to the user. The same is true if one of the commands executed returns an error. If all commands within the macro are executed successfully, one *[ACK]* is sent back to the user. This means that not all command responses within a macro are seen by the user.



## New Commands or Touch Events Abort Running Macros

If a macro is active and a command introducer (`\i`) is entered, the currently running macro is stopped. No response is sent in this case as the command following the `\i` has to be acknowledged. Normal execution of macros will also be stopped if a touch field key is reported (refer to [Enable/Disable Reporting Touch-Coordinates](#), a key is pressed and key reporting is enabled (refer to [Enable/Disable Keyboard Reporting](#)) or when the power switch is pressed and power-off notification is on (refer to [Set Power-Off Notification On/Off](#)). Protected macros are only aborted by a new command introducer (refer to [Execute Protected Macro](#)).

Macros are stored in the iLCD Controllers Flash via iLCD Manager XE and can be tested on the "Macros" page, although macro delays and execution speed settings are ignored when not executed from the memory.

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Execute Macro](#)
- [Execute Protected Macro](#)
- [Jump to Macro](#)
- [Delay Macro Execution](#)
- [Set Macro Execution Speed](#)
- [Allow Keyboard/Touch Events to be Processed](#)
- [Set Macro Timer](#)
- [Set Java Main Method Arguments](#)
- [Start Java Binary](#)

---

## Execute Macro

by index:

```
\i O E byte::macro_index
```

Parameter	Type	Range	Description
macro_index	<a href="#">word</a>	0 ... max. macro index	index of the macro to be executed

by name:

```
\i O E by_name::macro_name
```

Parameter	Type	Range	Description
macro_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated macro name

Executes (calls) the macro specified by *macro\_index* or *macro\_name*. Macros executed with this command will be interrupted by a new command sent via any interface as well as touch and keyboard events.

## Note

- If the macro does not exist or one of the called commands returns an error, `[NACK]` is returned.
- When the macro is fully executed, an `[ACK]` is sent. When called within another macro, execution returns to and continues processing the calling macro.
- When addressing by index, the macro offset (see [Set Macro Offset](#)) is taken into account.
- When addressing by name, the macro prefix (refer to [Set Macro Name Prefix](#)) and suffix (refer to [Set Macro Name Suffix](#)) are taken into account.
- Macros can call other macros, as long as the maximum nesting level is not exhausted. If a macro is executed within another macro ("nested"), the calling macro is resumed after the position where the macro was called once the called macro is finished.
- For building endless loops, use the [Jump to Macro](#) command to jump to the beginning of the current macro instead of using the [Execute Macro](#) command (this would overflow the macro stack and an error would be reported).

**Response:** `[ACK]`

## Example

```
\iOE\D1
\iOE\mMACRO\0
```

Executes the macro with index 1 and macro "MACRO".

## See also:

[Macro Related Commands](#)  
[Execute Protected Macro](#)  
[Jump to Macro](#)  
[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Macro Name Suffix](#)

## Execute Protected Macro

### by index:

```
\i O e byte::macro_index
```

Parameter	Type	Range	Description
macro_index	<a href="#">word</a>	0 ... max. macro index	index of the macro to be executed

### by name:

```
\i O e by_name::macro_name
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

macro_name	by_name	ASCII chars (0x01 .. 0xFF)	0-terminated macro name
------------	---------	----------------------------	-------------------------

Executes (calls) the macro specified by *macro\_index* or *macro\_name*. Macros called with this command will not be interrupted by touch or keyboard events. The first touch event happening during execution of a protected macro will be buffered and reported after the macro returns or a [Allow Keyboard/Touch Events to be Processed](#) command is executed within the macro. New commands sent via any interface can still interrupt the macro.

### Note

- If the macro does not exist or one of the called commands returns an error, *[NACK]* is returned.
- When the macro is fully executed, an *[ACK]* is sent. When called within another macro, execution returns to and continues processing the calling macro.
- When addressing by index, the macro offset (see [Set Macro Offset](#)) is taken into account.
- When addressing by name, the macro prefix (refer to [Set Macro Name Prefix](#)) and suffix (refer to [Set Macro Name Suffix](#)) are taken into account.
- Macros can call other macros, as long as the maximum nesting level is not exhausted. If a macro is executed within another macro ("nested"), the calling macro is resumed after the position where the macro was called once the called macro is finished.
- For building endless loops, use the [Jump to Macro](#) command to jump to the beginning of the current macro instead of using the [Execute Macro](#) command (this would overflow the macro stack and an error would be reported).

**Response:** [ACK]

### Example

```
\iOe\D1
\iOe\mMACRO\0
```

Executes the macro with index 1 and macro "MACRO" protected from touch or keyboard events.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Macro Related Commands](#)

[Execute Macro](#)

[Jump to Macro](#)

[Set Macro Offset](#)

[Set Macro Name Prefix](#)

[Set Macro Name Suffix](#)

---

## Jump to Macro

### by index:

```
\i O J byte::macro_index
```

Parameter	Type	Range	Description
macro_index	<a href="#">word</a>	0 ... max. macro index	index of the macro to jump to

**by name:**

```
\i O J by_name::macro_name
```

Parameter	Type	Range	Description
macro_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated macro name

Jumps to the macro referred by *macro\_index* or *macro\_name*. Macros executed with this command will be interrupted by a new command sent via any interface as well as touch and keyboard events.

**Note**

- If the macro does not exist or one of the called commands returns an error, *[NACK]* is returned.
- When the macro is fully executed, an *[ACK]* is sent. When called within another macro, execution does not return to the calling macro.
- When addressing by index, the macro offset (see [Set Macro Offset](#)) is taken into account.
- When addressing by name, the macro prefix (refer to [Set Macro Name Prefix](#)) and suffix (refer to [Set Macro Name Suffix](#)) are taken into account.
- Using this command is useful at the end of a macro to branch to the beginning of the current or another macro for building endless loops, as this command does not use the macro stack.

**Response:** *[ACK]*

**Example**

```
\iOJ\D5
```

Jumps to the macro with index 5.

**See also:**

[Macro Related Commands](#)  
[Execute Macro](#)  
[Execute Protected Macro](#)  
[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Macro Name Suffix](#)

**Delay Macro Execution**

```
\i O D word::delay
```

Paramete	Type	Range	Description
----------	------	-------	-------------

r			
delay	<a href="#">word</a>	0 ... 65535	delay for macro execution in units of 10ms

Waits for the specified amount of time (*delay* times 10ms) until the currently executed macro will be continued.

#### Note

- The only way to terminate such a delay is to send any command (such as the [No Operation](#) command) to the iLCD Controller. This is the reason why delays are ignored in normal command mode and why entering the command does not have any effect when not called via a macro.

**Response:** [ACK]

#### Example

```
\iOD\D3000
```

Waits for 30 seconds until the operation of the current macro is continued.

#### See also:

[Macro Related Commands](#)  
[Execute Macro](#)  
[Set Macro Execution Speed](#)

### Set Macro Execution Speed

```
\i O S word::speed
```

Parameter	Type	Range	Description
speed	<a href="#">word</a>	0 ... 65535	execution speed in units of 10ms

Sets the execution speed of a macro.

#### Note

- The bytes of the command that follows this command will each be executed at intervals of speed times 10ms.
- This command can be used to simulate a typewriter.
- There can be a maximum delay of close to 11 minutes for processing one command character.
- Macro execution speed can be set before or inside a macro and is reset when macro execution is finished.

**Response:** [ACK]

### Example

```
\iOS\D10
\iDTThis text is typed in 'typewriter mode'.\0
```

This example will cause the next command [Write Text](#) (`\iDT`) to process each single character with a delay of 100ms.

### See also:

[Macro Related Commands](#)

[Execute Macro](#)

[Delay Macro Execution](#)

---

## **Allow Keyboard/Touch Events to be Processed**

```
\i O K
```

Allows the controller to react to a new keyboard or touch event before a running touch macro is completed.

### Note

- This command has to be inserted into the macro to be interruptible, it is ignored when not called via a macro.
- Touch macros (see [The Concept of iLCD's Touch Fields](#)) called automatically when a touch field is pressed/released usually will not be stopped by any keyboard or touch event to avoid partially drawn touch fields when a new touch field is pressed or the previous one is released before the macro has been finished.
- While a touch field's make or break macro is being executed, keyboard and touch events are not being reported and touch macros are not started.
- A typical case for using this command is as follows: At the end of the drawing, carried out in the break macro of a touch field, a delay might need to be started before jumping to a "timeout" macro. Normally no other touch macro would be executed until this delay is finished (causing a "dead" touch screen), but when executing the above command before the delay, any newly pressed touch field aborts the current macro (causing the delay to stop) and executes the macro attached to the new touch field.

**Response:** [ACK]

### See also:

[Macro Related Commands](#)

[Execute Macro](#)

[Jump to Macro](#)

## Set Macro Timer

```
\i O T byte::time
```

Parameter	Type	Range	Description
time	<a href="#">byte</a>	0 ... 50	time in units of 100ms

Defines a time period *time*, in which incoming commands are buffered so that running macros will not be aborted.

### Note

- This allows calling macros with subsequent command lines from iLCD Manager XE's "Device" page as it prevents macros from being stopped by any command introducer (`\i`). When macro execution is finished, the subsequent commands will be executed immediately regardless of the remaining time.
- The timer is only started if the macro is executed by an incoming command. When a macro is called by another macro, the timer is not restarted, but keeps running if there is time left.
- The Macro Timer will be started every time a macro is called, to deactivate it the value time has to be set to 0.
- The default value for *time* is 0 (Macro Timer disabled). It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iOT\20
```

Incoming commands are buffered for 2 seconds after a macro was started.

### See also:

[Macro Related Commands](#)  
[Execute Macro](#)  
[Jump to Macro](#)

## Set Java Main Method Arguments

```
\i O V A string::args
```

Parameter	Type	Range	Description
args	<a href="#">string</a>	ASCII chars (0x01 .. 0xFF)	Arguments passed to the main method

Sets the arguments that will be passed to the main method

```
public static void main(String[] args)
```

when the Java VM is started with `\iOVS` or with the debugger.

#### Note

- To terminate a string, a NULL character (`\0`) has to be placed.
- Whitespaces separate the arguments.

**Response:** [ACK]

#### Example

```
\iOVAArg1 Arg2 Arg3\0
```

In Java this arguments will be passed to the main method in

```
args[0] args[1] args[2]
```

#### See also:

[Java](#)  
[Start Java Binary](#)  
[Can Run Java](#)

---

### **Start Java Binary**

```
\i O V S
```

Starts the Java VM with the Java program previously deployed. If the Java program shall be executed on power-up define a macro including this command and set it as startup macro on the [Macros Tab](#).

#### Note

- If main method arguments have been set before with [Set Java Main Method Arguments](#) they will be passed to the main method.
- The Java program might have been deployed via the Java Tab (Start::Deploy) or a normal iLCD program write.

**Response:** [ACK] [ACK]

The first [ACK] acknowledges the reception of the command. The second is sent after the Java program has finished.

#### See also:

[Java](#)  
[Set Java Main Method Arguments](#)  
[Can Run Java](#)  
[Java IDE](#)



---

## Touch Screen Related Commands

### The Concept of iLCD's Touch Fields

To allow maximum flexibility and easy use of touch fields without using any resources of the application's controller, the following concept was implemented:

You may define up to 64 touch fields with any rectangle size and where fields may overlap each other. The maximum field size is the pixel screen size used by the current iLCD. Any touch field may have a key assigned which is reported to the application when the touch field is pressed and released. In addition, any touch field may have assigned a text message, a make and/or a break macro which are automatically executed when the field is pressed/released.

When using multiple screens, note that touch fields are created at the screen index of the currently active draw screen. On the other hand, only touch fields on the activated view screen get evaluated and reported.

### PCAP Touch Screen

Projected capacitive (PCAP) touch screens support up to 5 simultaneously evaluated touch points. The sensitivity is determined by values stored in the EEPROM (refer to [EEPROM Related Commands](#)).

### Make/Break Events and Macros

The iLCD Controller automatically checks in the order of field indices if the touch panel is pressed within one of the previously defined fields. If the touch pressure is within the bounds of a touch field, the make key sequence ([Touch Field Press/Release](#)) is reported to the application if a key is assigned to this field. This key sequence is exactly the same sequence as used for a keyboard (refer to [Key Press/Release](#)).

After the key has been reported, the make macro assigned to the field is executed as long as automatic execution is not disabled via the [Enable/Disable Automatic Touch Macro Executing](#) command. Before execution of this macro starts, the iLCD Controller sets the 'current touch field index' accordingly and the cursor to the upper/left corner of the touch field, allowing the execution of all necessary drawing stuff within the make macro (usually painting the touch field in a "pressed" state).

After the make macro has been fully executed (or an [Allow Keyboard/Touch Events to be Processed](#) command has been found), the iLCD Controller checks if the touch screen has been released or waits for release. On release, the corresponding break key sequence is sent to the application and the break macro is executed. If there are any new touch events while the make macro is still running, the break event is ignored. Touch macros do not send an extra [ACK] after execution is finished. They do however send a [NACK] if one of the macro commands fail.

### Touch Macros are NOT Aborted by Touch Events

Please keep in mind, that any touch events happening during execution of a make or break macro are ignored unless the command [Allow Keyboard/Touch Events to be Processed](#) is executed within the running touch macro.

## Current Touch Field Index and Synchronization

The iLCD Controller also keeps track of which touch field has been pressed using an internal variable known as the 'current touch field index'. This value is automatically set when a touch field is created, pressed or released, but the user can also set it manually by using [Set Current Touch Field Index](#).

This can be useful when using touch macros and synchronizing with a controlling application at the same time. To realize this behaviour, make and break macros are assigned to touch fields but automatic execution is disabled using [Enable/Disable Automatic Touch Macro Executing](#). After the key sequence of the pressed/released touch field is reported to the application, the according make/break macro can be started using the 'current touch field index' (parameter 0xFF) with the command [Execute Touch Make Macro](#) resp. [Execute Touch Break Macro](#). This way, the controlling application receives acknowledgements when macro execution is finished.

## Reuse Macros for Similar Touch Fields

One make/break macro can be used for any number of different touch fields on different locations, as the cursor is automatically set to the upper/left corner of the touch field before the macro starts (either automatically or with the command [Execute Touch Make Macro](#) resp. [Execute Touch Break Macro](#)). Painting individual touch field texts can be done via [Draw Touch Field Text Message](#) using the 'current touch field index'. Cursor movements within make/break macros can be done relative to the current cursor position via the [Increment/Decrement Column Address](#) and [Increment/Decrement Row Address](#) to allow the use of the macro for different touch fields on different cursor positions. Using the [Cursor Memory Related Commands](#) can further help you to simplify the field drawing routines and to increase the usability of macros by multiple touch fields.

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Calibrate Touch Screen](#)
- [Calibrate Touch Screen and Report](#)
- [Verify Touch Screen Calibration](#)
- [Set Touch Field Width](#)
- [Set Touch Field Height](#)
- [Set Touch Field Make Macro](#)
- [Set Touch Field Break Macro](#)
- [Enable/Disable Automatic Touch Macro Executing](#)
- [Set Touch Field Text Message](#)
- [Create/Define Touch Field](#)
- [Remove Touch Field](#)
- [Global Enable/Disable Touch Fields](#)
- [Set Current Touch Field Index](#)
- [Execute Touch Make Macro](#)
- [Execute Touch Break Macro](#)
- [Draw Touch Field Text Message](#)
- [Enable/Disable Touch Field Reporting](#)
- [Enable/Disable Reporting Touch-Coordinates](#)
- [Enable/Disable Reporting Movements](#)
- [Retrieve Last Touch Screen Event](#)
- [Set Number of Touch Fingers](#)
- [Set Threshold for Movement Reporting](#)
- [Set Cursor to Touch Field](#)
- [Enable/Disable Touch Field Queue](#)
- [Start Demonstration Mode](#)

## **Calibrate Touch Screen**

```
\i T C
```

Starts the procedure to calibrate a resistive touch screen.

### **Note**

- This command returns a *[NACK]* for a capacitive touch screen.
- All iLCD panels with a touch panel are calibrated after production, but you may repeat the calibration process anytime.
- When issuing this command, the iLCD panel shows instructions for calibration on the screen. After pressing two positions, the calibration is done and the calibration values are stored in the iLCD controller automatically.
- Sending [Reset All](#) stops the calibration process, the previous existing values are used then.
- To get a report after successful calibration, use the [Calibrate Touch Screen and Report](#) command.

**Response:** [ACK]

### **Example**

```
\iTC
```

Start the calibration procedure.

### **See also:**

[Calibrate Touch Screen and Report](#)  
[Verify Touch Screen Calibration](#)

---

## **Calibrate Touch Screen and Report**

```
\i T c
```

Starts the touch screen calibration for a resistive touch screen (see also [Calibrate Touch Screen](#)) and also sends an asynchronous report (refer to [Calibrate Touch Screen Done](#)) if the calibration was successful.

### **Note**

- This command returns a *[NACK]* for a capacitive touch screen.
- If the second calibration point is too close to the first one, the report is sent with a *[NACK]*.

**Response:** [ACK]

### **Response Example**

```
[ACK]
```

The on-screen touch calibration was started.

**See also:**

[Calibrate Touch Screen](#)  
[Calibrate Touch Screen Done](#)

---

## **Verify Touch Screen Calibration**

```
\i T V
```

The calibration of the touch screen can be tested after sending this command. The position of every touch event is marked with a small cross on the screen.

**Note**

- The validation can be stopped via the [Reset All](#) command.

**Response:** [ACK]

**Example**

```
\iT V
```

When a touch event occurs, the screen turns white and every touch event shows a small cross.

**See also:**

[Calibrate Touch Screen](#)  
[Create/Define Touch Field](#)

---

## **Set Touch Field Width**

```
\i T W word::width
```

Parameter	Type	Range	Description
width	<a href="#">word</a>	0 ... display width	width of the touch field (0 = full width)

Sets up the width of any subsequently defined touch field (see [Create/Define Touch Field](#)) according to *width*.

**Note**

- Sending this command does not change anything for touch fields already defined.
- If *width* is 0, the full width of the current viewport is used.
- After startup and after sending the [Reset All](#) command *width* is set to 30.

**Response:** [ACK]

### Example

```
\iTW\D120
```

Sets the width for the next touch field to be created to 120 pixels.

### See also:

[Touch Screen Related Commands](#)

[Set Touch Field Height](#)

[Create/Define Touch Field](#)

## Set Touch Field Height

```
\i T H word::height
```

Parameter	Type	Range	Description
height	<a href="#">word</a>	0 ... display height	height of the touch field (0 = full height)

Sets up the height of any subsequently defined touch field (see [Create/Define Touch Field](#)) according to *height*.

### Note

- Sending this command does not change anything for touch fields already defined.
- If *height* is 0, the full height of the current viewport is used.
- After startup and after sending the [Reset All](#) command *height* is set to 20.

**Response:** [ACK]

### Example

```
\iTH\D25
```

Sets the height for the next touch field to be created to 25 pixels.

### See also:

[Touch Screen Related Commands](#)

[Set Touch Field Width](#)

[Create/Define Touch Field](#)

## Set Touch Field Make Macro

by index:

```
\i T M word::macro_index
```

Parameter	Type	Range	Description
macro_index	<a href="#">word</a>	0 ... max. macro index (-1 = none)	index of the macro

by name:

```
\i T M by_name::macro_name
```

Parameter	Type	Range	Description
macro_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated macro name

Assigns the macro to be called when any subsequently defined (see [Create/Define Touch Field](#)) touch field is pressed.

### Note

- If a make macro is assigned to a touch field, the cursor position is automatically updated to the origin of the pressed touch field before executing the macro.
- When automatic execution of touch macros is disabled (refer to [Enable/Disable Automatic Touch Macro Executing](#)) and the touch field is pressed, the cursor position will be updated but the macro will not be executed.
- Automatically executed Touch Macros do not send an extra [ACK] after execution is finished. They do however send a [NACK] if one of the macro commands fail.
- Setting *macro\_index* to 0xFFFF (decimal word -1) causes no macro to be executed when the subsequently defined touch field is pressed and the cursor position will not be updated.
- After startup and after sending the [Reset All](#) command *macro\_index* is set to -1.
- At this time, the attributes macro offset, prefix and suffix are ignored and the existence of the macro index or name is not verified (see [Set Macro Offset](#), [Set Macro Name Prefix](#) and [Set Macro Name Suffix](#)). Offset resp. pre-/suffix is taken into account when the macro is actually executed (in case of the according make event or by the [Execute Touch Make Macro](#) command).

☐ Sending this command does not change anything for touch fields already defined.

**Response:** [ACK]

### Example

```
\iTM\D2
```

Assigns the macro with index 2 as make macro to the subsequently defined touch fields.

**See also:**

[Touch Screen Related Commands](#)  
[Set Touch Field Break Macro](#)  
[Create/Define Touch Field](#)  
[Enable/Disable Automatic Touch Macro Executing](#)

[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Macro Name Suffix](#)

## **Set Touch Field Break Macro**

**by index:**

```
\i T B word::macro_index
```

Parameter	Type	Range	Description
macro_index	<a href="#">word</a>	0 ... max. macro index (-1 = none)	index of the macro

**by name:**

```
\i T B by_name::macro_name
```

Parameter	Type	Range	Description
macro_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated macro name

Assigns the macro to be called when any subsequently defined (see [Create/Define Touch Field](#)) touch field is released.

### **Note**

- If a break macro is assigned to a touch field, the cursor position is automatically updated to the origin of the released touch field before executing the macro.
- When automatic execution of touch macros is disabled (refer to [Enable/Disable Automatic Touch Macro Executing](#)) and the touch field is released, the cursor position will be updated but the macro will not be executed.
- Automatically executed Touch Macros do not send an extra [ACK] after execution is finished. They do however send a [NACK] if one of the macro commands fail.
- Setting *macro\_index* to 0xFFFF (decimal word -1) causes no macro to be executed when the subsequently defined touch field is released and the cursor position will not be updated.
- After startup and after sending the [Reset All](#) command *macro\_index* is set to -1.
- At this time, the attributes macro offset, prefix and suffix are ignored and the existence of the macro index or name is not verified (see [Set Macro Offset](#), [Set Macro Name Prefix](#) and [Set Macro Name Suffix](#)). Offset resp. pre-/suffix is taken into account when the macro is actually executed (in case of the according break event or by the [Execute Touch Break Macro](#) command).

☐ Sending this command does not change anything for touch fields already defined.

**Response:** [ACK]

### **Example**

```
\iT B \D1
```

Assigns the macro with index 1 as break macro to the subsequently defined touch fields.

**See also:**

[Touch Screen Related Commands](#)  
[Set Touch Field Make Macro](#)  
[Create/Define Touch Field](#)  
[Enable/Disable Automatic Touch Macro Executing](#)  
[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Macro Name Suffix](#)

**Enable/Disable Automatic Touch Macro Executing**

```
\i T E A bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether auto-executing is on (1) or off (0)

Enables or disables automatic execution of touch field's make or break macros.

**Note**

- When automatic execution is disabled, assigned make or break macros (refer to [Set Touch Field Make Macro](#) and [Set Touch Field Break Macro](#)) are not executed when the touch field is pressed or released but the cursor position for touch fields with assigned macros will still be updated.
- Make or break macros can still be assigned to touch fields while the controlling application is now able to receive touch reports (refer to [Touch Field Press/Release](#)). As a response to this event, the assigned macro must then be called by the application (refer to [Execute Touch Make Macro](#) and [Execute Touch Break Macro](#)).
- This way, all advantages of the touch macro concept (refer to [Touch Screen Related Commands](#)) can be used while the touch report allows synchronization with the controlling application.
- The default value for *on\_off* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iTEA\0
```

Disables automatic execution of touch macros.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Touch Screen Related Commands](#)  
[Set Touch Field Make Macro](#)  
[Set Touch Field Break Macro](#)



[Execute Touch Make Macro](#)  
[Execute Touch Break Macro](#)  
[Enable/Disable Reporting Touch-Coordinates](#)  
[Enable/Disable Reporting Movements](#)

## **Set Touch Field Text Message**

**by index:**

```
\i T T word::message_index
```

Parameter	Type	Range	Description
message_index	<a href="#">word</a>	0 ... max. text message index	index of the text message

**by name:**

```
\i T T by_name::message_name
```

Parameter	Type	Range	Description
message_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated message name

Assigns the text message defined by *message\_index* or *message\_name* to any subsequently defined touch field (see [Create/Define Touch Field](#)).

### **Note**

- Sending this command does not change anything for touch fields already defined.
- Setting *message\_index* to 0xFFFF (decimal word "65535") causes no text to be assigned to the subsequently defined touch field.
- After startup and after sending the [Reset All](#) command *message\_index* is set to -1.
- Text messages assigned to touch fields are used by the [Draw Touch Field Text Message](#) command.
- The text message can even contain any valid ANSI sequence so allowing various attributes/fonts to be changed easily.
- At this time, the attributes message offset, prefix and suffix are ignored and the existence of the message index or name is not verified (see [Set Message Offset](#), [Set Message Name Prefix](#) and [Set Message Name Suffix](#)). Offset resp. pre-/suffix is taken into account when the message is actually drawn (see [Draw Touch Field Text Message](#)).

**Response:** [ACK]

### **Example**

```
\iTT\D3
```

Assigns the text message with index 3 to any subsequently defined touch field.

**See also:**

[Touch Screen Related Commands](#)  
[Draw Touch Field Text Message](#)  
[Create/Define Touch Field](#)  
[Draw Touch Field Text Message](#)  
[Set Current Touch Field Index](#)  
[Set Message Offset](#)  
[Set Message Name Prefix](#)  
[Set Message Name Suffix](#)

**Create/Define Touch Field**

```
\i T A byte::field_idx byte::key
```

Parameter	Type	Range	Description
field_idx	byte	0 ... 63 (-1 = next free touch field index)	index of the touch field
key	byte	0 ... 255	value reported in case of touch event

Creates or redefines the touch field with index *field\_idx* with the current values and properties previously set (e.g. [Set Touch Field Width](#), [Set Touch Field Make Macro](#), etc.) and also assigns *key* to the field.

**Note**

- The upper left position of the touch field is taken from the current cursor position.
- *field\_idx* may have the hex value 0xFF (decimal byte -1), in this case the controller will choose the next free (unassigned) touch field index.
- Assigning a *key* value of 0 will inhibit the reporting of the make/break sequence to the application. The make/break macros however, if assigned, will still be executed.
- Using an invalid *field\_idx* or a value of 0xFF when all 64 touch fields are already defined causes a *[NACK]* to be sent by the iLCD Controller.
- This command can even be used within a make or break macro to redefine the behavior of the current touch field to implement, for example, a toggle field behavior.
- Using this command also sets the 'current touch field index' (see [Set Current Touch Field Index](#)), which can be used by [Execute Touch Make Macro](#), [Execute Touch Break Macro](#) and [Draw Touch Field Text Message](#).

**Response:** [ACK]

**Example**

```
\iTA\xFF\x1
```

This will create a touch field with the next free index. If this touch field is pressed afterwards, *K/k[01] [ACK]* will be reported via the active interface.

**See also:**

[Touch Screen Related Commands](#)  
[Remove Touch Field](#)  
[Enable/Disable Touch Field Reporting](#)  
[Set Current Touch Field Index](#)  
[Execute Touch Make Macro](#)  
[Execute Touch Break Macro](#)  
[Draw Touch Field Text Message](#)

**Remove Touch Field**

```
\i T R byte::field_idx
```

Parameter	Type	Range	Description
field_idx	byte	0 ... 63	index of the touch field (-1 = all, -2 = all in viewport)

Removes the touch field specified by *field\_idx*.

**Note**

- If *field\_idx* is set to 0xFF (decimal byte 255), all touch fields on all screens and viewports are removed.
- In firmware versions 4.17 and higher, a *field\_idx* of 0xFE (decimal byte -2) removes all touch fields in the active viewport. When the active viewport is 0, all touch fields that are not defined in user-defined viewports are removed. If screens are supported, this can be used to remove all touch fields on the current draw screen (refer to [Screen Memory Related Commands](#)).
- The commands [Reset All](#) and [Reboot Panel Controller](#) remove all touch fields as well.

**Response:** [ACK]

**Example**

```
\iTR\12
```

Removes the touch field with index 12.

**See also:**

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)

**Global Enable/Disable Touch Fields**

```
\i T G bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	touch fields enabled (1) or disabled (0)

Enables or disables all touch fields.

#### Note

- The definition of any touch field is not changed by this command, the fields are simply ignored when they are disabled.
- Disabling touch fields until all touch fields are defined can be a sensible precaution because it prevents the possible interruption of any macro used to define several fields.
- Executing the [Reset All](#) removes all touch fields and globally enables touch field processing then.

**Response:** [ACK]

#### Example

```
\iTG\0
```

Disables all touch fields.

#### See also:

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)

### **Set Current Touch Field Index**

```
\i T I byte::field_idx
```

Parameter	Type	Range	Description
field_idx	<a href="#">byte</a>	0 ... 63	current touch field index

Sets the iLCD Controller's 'current touch field index' to *field\_idx*.

#### Note

- The iLCD Controller's 'current touch field index' is an internal variable which is automatically updated when a touch field is created, pressed or released.
- The value of the 'current touch field index' is used by [Execute Touch Make Macro](#), [Execute Touch Break Macro](#) and [Draw Touch Field Text Message](#).

**Response:** [ACK]

## Example

```
\iTI\23
```

Sets the 'current touch field index' of the iLCD Controller to 23.

### See also:

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)  
[Global Enable/Disable Touch Fields](#)  
[Execute Touch Make Macro](#)  
[Execute Touch Break Macro](#)  
[Draw Touch Field Text Message](#)

## Execute Touch Make Macro

```
\i T E M byte::field_idx
```

Parameter	Type	Range	Description
field_idx	byte	0 ... 63	index of the touch field (-1 = current index)

Executes the make macro assigned to the touch field with index *field\_idx*.

### Note

- The current macro offset, prefix and suffix are taken into account (refer to [Set Macro Offset](#), [Set Macro Name Prefix](#), and [Set Macro Name Suffix](#)).
- If *field\_idx* is set to 0xFF (decimal byte -1) the 'current touch field index' is used (refer to [Touch Screen Related Commands](#)).
- Before execution of the make macro, the cursor position is set to the upper/left corner of the corresponding touch field as it would when automatically executed. This way, one make macro can be used for multiple identical buttons, e.g. using the touch field text message for captions (refer to [Draw Touch Field Text Message](#)).
- If *field\_idx* is out of bounds, the index has no field attached yet or the touch field indexed has no make macro assigned to, a *[NACK]* is sent by the iLCD Controller.
- Calling a make macro is normally done automatically via the iLCD Controller when the appropriate touch field is pressed, but can be done via this command as well.

**Response:** [ACK]

## Example

```
\iTEM\35
```

Executes the make macro that was assigned to the touch field with index of 35.

**See also:**

[Touch Screen Related Commands](#)  
[Execute Touch Break Macro](#)  
[Set Current Touch Field Index](#)  
[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Macro Name Suffix](#)

**Execute Touch Break Macro**

```
\i T E B byte::field_idx
```

Parameter	Type	Range	Description
field_idx	byte	0 ... 63	index of the touch field (-1 = current index)

Executes the break macro assigned to the touch field with index *field\_idx*.

**Note**

- The current macro offset, prefix and suffix are taken into account (refer to [Set Macro Offset](#), [Set Macro Name Prefix](#), and [Set Macro Name Suffix](#)).
- If *field\_idx* is set to 0xFF (decimal byte -1) the 'current touch field index' is used (refer to [Touch Screen Related Commands](#)).
- Before execution of the break macro, the cursor position is set to the upper/left corner of the corresponding touch field as it would when automatically executed. This way, one break macro can be used for multiple identical buttons, e.g. using the touch field text message for captions (refer to [Draw Touch Field Text Message](#)).
- If *field\_idx* is out of bounds, the index has no field attached yet or the touch field indexed has no break macro assigned to, a *[NACK]* is sent by the iLCD Controller.
- Calling a break break is normally done automatically via the iLCD Controller when the appropriate touch field is released, but can be done via this command as well. This can be useful to draw the released state of a touch field after definition.

**Response:** [ACK]

**Example**

```
\iTEB\10
```

Executes the break macro that was assigned to the touch field with index of 10.

**See also:**

[Touch Screen Related Commands](#)  
[Execute Touch Make Macro](#)  
[Set Current Touch Field Index](#)  
[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Macro Name Suffix](#)

## Draw Touch Field Text Message

```
\i T D byte::field_idx
```

Parameter	Type	Range	Description
field_idx	byte	0 ... 63	index of the touch field (-1 = current index)

Draws the text message assigned to the touch field with index *field\_idx* in its top left corner.

### Note

- The message offset, prefix and suffix are taken into account (see [Set Message Offset](#), [Set Message Name Prefix](#) and [Set Message Name Suffix](#)).
- If *field\_idx* is set to 0xFF (decimal byte -1) the 'current touch field index' is used (refer to [Touch Screen Related Commands](#)).
- If no text message is assigned to the field and the key assigned to the field is other than 0x0, the key is printed instead. This behavior is useful for single character touch fields like "keyboard" fields, as no text message has to be assigned for every touch field.
- If *field\_idx* is out of bounds, the corresponding index has no active field attached or no text can be displayed, a *[NACK]* is sent by the iLCD Controller.
- Note that when a make or break macro is executed, the cursor position is automatically set to the upper/left corner of the corresponding touch field. So when using this command inside a touch macro, text output can be placed or aligned (see [Set Text Alignment](#)) relative to the touch field position.

**Response:** [ACK]

### Example

```
\iDT\5
```

Draws the text message assigned to the touch field with index 5 in its top left corner, given the settings for the message offset aren't changed.

### See also:

[Set Current Touch Field Index](#)  
[Set Touch Field Text Message](#)  
[Set Message Offset](#)  
[Set Message Name Prefix](#)  
[Set Message Name Suffix](#)

## Set Cursor to Touch Field

```
\i T E C byte::field_idx
```

Parameter	Type	Range	Description
<code>field_idx</code>	<a href="#">byte</a>	0 ... 63	index of the touch field (-1 = current index)

Sets the cursor to the upper/left corner of the touch field specified by `field_idx`.

#### Note

- If `field_idx` is set to 0xFF (decimal byte -1) the 'current touch field index' is used (refer to [Touch Screen Related Commands](#)).

**Response:** [ACK]

#### Example

```
\iTS\5
```

Sets the cursor position to the upper/left corner of the touch field with index 5.

#### See also:

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)

### Enable/Disable Touch Field Reporting

```
\i T P bool::on_off
```

Parameter	Type	Range	Description
<code>on_off</code>	<a href="#">bool</a>	0 ... 1	determines whether touch events are reported (1) or not (0)

Enables or disables the reporting of touch field make and break events.

#### Note

- Any assigned make or break macros will still be executed.
- The default value for `on_off` is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

#### Example

```
\iTP\0
```

Disables the reporting of touch field events.



Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)  
[Moving Coordinate of Touch Field Press](#)  
[Touch Field Press/Release](#)  
[Enable/Disable Reporting Touch-Coordinates](#)  
[Enable/Disable Reporting Movements](#)

## **Enable/Disable Reporting Touch-Coordinates**

```
\i TK bool::on_off
```

Parameter	Type	Range	Description
on_off	bool	0 ... 1	determines whether reporting is on (1) or off (0)

Enables or disables the reporting of touch coordinates.

**Note**

- When reporting a pressed or released touch field normally only the key is reported as described in [Touch Field Press/Release](#). Under certain circumstances the coordinates of pressing and releasing the field is of interest as well. With this command it can be set that reports also contain the coordinates of a touch event (see [Touch Field Press/Release + Coordinate of Event](#)).
- The default value for *on\_off* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iTK\1
```

Enables the reporting of touch event coordinates.

**See also:**

[Touch Screen Related Commands](#)  
[Touch Field Press/Release](#)  
[Touch Field Press/Release + Coordinate of Event](#)  
[Enable/Disable Reporting Movements](#)

## **Enable/Disable Reporting Movements**

```
\i T O bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether movements are reported (1) or not (0)

Enables or disables the asynchronous reporting of touch event movements (changing coordinates of the pressed position due to movement of the finger).

### **Note**

- For the reported sequence, refer to [Moving Coordinate of Touch Field Press](#).
- Only touch fields having a key attached get reported!
- The distance threshold for movement reports can be modified with the command [Set Threshold for Movement Reporting](#).
- When polling touch events with the command [Retrieve Last Touch Screen Event](#), movement sequences are sent in any case.
- The default value for *on\_off* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### **Example**

```
\iTO\1
```

Enables the reporting of touch field movements.

### **See also:**

[Touch Screen Related Commands](#)  
[Moving Coordinate of Touch Field Press](#)  
[Touch Field Press/Release](#)  
[Enable/Disable Reporting Touch-Coordinates](#)  
[Set Threshold for Movement Reporting](#)

## **Retrieve Last Touch Screen Event**

```
\i T ?
```

Returns the most recent touch event, which is always stored by the controller (even if no key is assigned to the touch field).

### **Note**

- Coordinates can become negative and the minimum/maximum coordinate values which are reported can even fall outside the physical screen dimensions by one pixel when converted to absolute coordinates.

- It is recommended to wait for the report before sending additional commands, since touch events use a different buffer and could otherwise be delayed by the responses to following commands.
- When more than one touch point are evaluated simultaneously (refer to [Set Number of Touch Fingers](#)), a *point\_id* is inserted to the report (refer to [Touch Field Press/Release & Coordinate of Event](#)).

**Response:** [ACK] event (point\_id) field\_idx ev\_coord\_x ev\_coord\_y [ACK]

Parameter	Type	Range	Description
event	event	K, k, M	event type (refer to table below)
point_id	<a href="#">byte</a>	0 ... 4	(optional) ID of the touch point (finger)
field_idx	<a href="#">byte</a>	0 ... 63	index of the touch field
ev_coord_x	<a href="#">word</a>	0 ... display width	horizontal position of the event
ev_coord_y	<a href="#">word</a>	0 ... display height	vertical position of the event

Event	Description
K	touch has been pressed (make)
k	touch has been released (break)
M	location of the press has been moved
NULL	no event has occurred

### Response Example

```
[ACK]K[01][00][7D][00][96][ACK]
```

The last touch event was a make event (touch field was pressed) reported from the touch field with index 1 at coordinates x = 125 pixels and y = 150 pixels. Here, the *point\_id* is missing since only one touch point is evaluated.

### See also:

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)  
[Set Number of Touch Fingers](#)

## Set Number of Touch Fingers

```
\i T S N byte::fingers
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

<code>fingers</code>	<code>byte</code>	<code>1 ... 5</code>	number of simulataneously evaluated touch points
----------------------	-------------------	----------------------	--

Sets the number of evaluated touch points to *fingers*.

### Note

- While *fingers* is higher than 1, an automatically assigned point ID is appended to all touch field reports (refer to [Touch Field Press/Release](#) and [Touch Field Press/Release + Coordinate of Event](#)).
- On iLCD panels with resistive touch screen, *fingers* is always 1.
- On iLCD panels with capacitive touch screen, the maximum value for *fingers* is 5.
- The default value for *fingers* is 1 in any case. It will be automatically set to default on startup.

**Response:** [ACK]

### Example

```
\iTSN\2
```

Sets the number of touch points evaluated simultaneously to 2 (works on iLCD panels with capacitive touch screen only).

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)  
[Touch Field Press/Release](#)

## Set Threshold for Movement Reporting

```
\i T S M byte::threshold
```

Parameter	Type	Range	Description
<code>threshold</code>	<code>byte</code>	<code>0 ... 255</code>	threshold for touch move events

When reporting of touch movements is enabled (refer to [Enable/Disable Reporting Movements](#)), reports are sent every time a touch point moves by *threshold* pixels.

### Note

- Sending this command does not change anything for touch fields already defined.
- The default value for *threshold* is 8. It will be automatically set to default on startup.

**Response:** [ACK]

### Example

```
\iTSM\20
```

Movement reports are sent every time the touch events moves by 20 pixels in any direction.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Touch Screen Related Commands](#)  
[Enable/Disable Reporting Movements](#)

## Enable/Disable Touch Field Queue

```
\i T Q bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether touch events are queued (1) or not (0)

Enables or disables a queue buffering touch events during the execution of a non-interruptible macro (protected, make or break macros).

### Note

- After the macro is finished (and there are no bytes left to process in the input buffer), the queued events are evaluated.
- The default value for *on\_off* is 0. It will be automatically set to default on startup.

**Response:** [ACK]

### Example

```
\iTQ\1
```

Enables the touch queue.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Touch Screen Related Commands](#)  
[Create/Define Touch Field](#)  
[Execute Protected Macro](#)  
[Set Touch Field Make Macro](#)  
[Set Touch Field Break Macro](#)

## PCAP only: Start Demonstration Mode

### by index:

```
\i T v byte::mode word::timeout word::macro_index
```

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	1	demo mode (for future use)
timeout	<a href="#">word</a>	0 ... 65535	timeout in units of 10ms
macro_index	<a href="#">word</a>	0 ... max. macro index (-1 = none)	index of the macro

### by name:

```
\i T v byte::mode word::timeout by_name::macro_name
```

Parameter	Type	Range	Description
mode	<a href="#">byte</a>	1	demo mode (for future use)
timeout	<a href="#">word</a>	0 ... 65535	timeout in units of 10ms
macro_name	<a href="#">by_name</a>	ASCII chars (0x01 .. 0xFF)	0-terminated macro name

Starts a demonstration according to *mode* and jumps to the macro specified in *macro\_index* or *macro\_name* after the *timeout*.

### Note

- Any event on the touchscreen will reset the timeout period.
- All running animations are stopped (refer to [LCD Graphics Commands](#)).
- If the macro does not exist or one of the called commands returns an error, *[NACK]* is returned.
- When addressing by index, the macro offset (see [Set Macro Offset](#)) is taken into account.
- When addressing by name, the macro prefix (refer to [Set Macro Name Prefix](#)) and suffix (refer to [Set Macro Name Suffix](#)) are taken into account.
- Demonstration mode 1 is utilized by the "Fancy PCAP Touch Demo" (iLCD Manager XE -> File -> New -> Demos).

**Response:** [ACK]

### Example

```
\iTv\1\D500\mReturn from Demo\0
```

Starts demonstration mode 1 and jumps to macro "Return from Demo" after 5 seconds of inactivity.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Touch Screen Related Commands](#)

---

## Input/Output Related Commands

### General Information about Inputs/Outputs

The iLCD Controllers allow most port pins to be assigned as digital or analog inputs, outputs (pull down or push/pull) or keyboard columns via iLCD Manager XE. All commands referring to port pins below refer to the logical port name, not the physical port pin name.

This means the physical port pin "Keyboard column 3" may have the logical name e.g. "Output #9", so turning on this pin (= setting it to high when it is defined as a push/pull pin) can be done via the [Set Output](#) command like this:

```
\i I L S 09H 01H
```

When using the same port as a pull-down output port, sending the above command pulls the pin to low (making it "active") instead. Note that the Color iLCD Controllers can only source/sink 4mA. Please check the documentation for your particular board.

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Set Output](#)
- [Set Multiple Outputs](#)
- [Set Output Blink Frequency](#)
- [Set Relays On/Off/PWM](#)
- [Relays One Shot](#)
- [Enable/Disable Rotary Encoder Reporting](#)
- [Enable/Disable Keyboard](#)
- [Enable/Disable Keyboard Reporting](#)
- [Get Keyboard State](#)
- [Set Baud Rate](#)
- [Get Current Communication-Port](#)
- [Disable Communication-Ports](#)
- [Get Enabled Communication-Ports](#)
- [Get Inputs State](#)
- [Get ADC Value](#)
- [Set DAC Value](#)

---

## Set Output

```
\i I L S byte::out_no byte::mode
```

Parameter	Type	Range	Description
-----------	------	-------	-------------

out_no	<a href="#">byte</a>	0 ... 15	output number to be turned on/off
mode	<a href="#">byte</a>	0 ... 2	mode to set

This command is used for controlling outputs via the following modes:

Mode	Description
0	sets output to off
1	sets output to on
2	sets output to blink

#### Note

- Multiple outputs can be controlled with the [Set Multiple Outputs](#) command.
- When the output is set to blink, the frequency can be adjusted with the [Set Output Blink Frequency](#) command.
- The startup values can be defined on the "Settings" page of iLCD Manager XE. All outputs will be automatically set to the startup value on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- When trying to address an output port that is not defined as an output via iLCD Manager XE, the command is simply ignored; a `[NACK]` is returned then.

**Response:** [ACK]

#### Example

```
\iILS\3\2
```

After sending this command, output number 3 will blink (periodically toggles state after the interval specified by the blink frequency).

#### See also:

[Input/Output Related Commands](#)  
[Set Multiple Outputs](#)  
[Set Output Blink Frequency](#)

## Set Multiple Outputs

```
\i I L s word::out_mask word::blink_mask
```

Parameter	Type	Range	Description
out_mask	<a href="#">wbits</a>	0x0000 ... 0xFFFF	every set bit represents an output turned on
blink_mask	<a href="#">wbits</a>	0x0000 ... 0xFFFF	every set bit represents a blinking output



Sets multiple outputs, similar to the [Set Output](#) command.

### Note

- The two masks refer to bit addressed outputs. Where bit 0 refers to output #0, bit 15 to output #15.
- If pin is not previously defined as an output via iLCD Manager XE, setting this bit has no effect.
- The startup values can be defined on the "Settings" page of iLCD Manager XE. All outputs will be automatically set to the startup value on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iILs\XA\X500
```

This turns on outputs #1 and #3 (0xA = Bin 0000 0000 0000 1010) and sets outputs #8 and #10 (0x500 = Bin 0000 0101 0000 0000) to blink.

### See also:

[Input/Output Related Commands](#)  
[Set Output](#)  
[Set Output Blink Frequency](#)

## Set Output Blink Frequency

```
\i I L F byte::period
```

Parameter	Type	Range	Description
period	<a href="#">byte</a>	1 ... 255	interval of state changes in units of 10ms

Sets the blinking frequency for all outputs by defining the interval of state changes.

### Note

- When an output is currently in blink mode (see [Set Output](#) and [Set Multiple Outputs](#)), the frequency changes immediately.
- The default value for *period* is 20, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- The pin used must first be enabled using functions or the settings tab.

**Response:** [ACK]

## Example

```
\iILF\25
```

This will set the blinking frequency to 2 Hertz (250ms on and 250ms off).

### See also:

[Input/Output Related Commands](#)

[Set Output](#)

[Set Multiple Outputs](#)

---

## Set Relays On/Off/PWM

```
\i I R byte::relay_no byte::mode
```

Parameter	Type	Range	Description
relay_no	<a href="#">byte</a>	0 ... 1	number of the relay
mode	<a href="#">byte</a>	0 ... 2	mode of the relay

Sets relay output *relay\_no* to one of the following values for *mode*:

Mode	Description
0	set relay off
1	set relay on
2	enable PWM on relay

### Note

- Setting *mode* to 2 enables the PWM 0 on relay output 0 or PWM 1 on relay output 1. (see [Pulse Width Modulation \(PWM\) Related Commands](#)).
- By default, both relay outputs are off. This can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

## Example

```
\iIR\0\2
```

Enables PWM 0 on relay output 0.

**See also:**[Input/Output Related Commands](#)[Pulse Width Modulation \(PWM\) Related Commands](#)[Relays One Shot](#)**Relays One Shot**

```
\i I r byte::relay_no bool::on_off word::time
```

Parameter	Type	Range	Description
relay_no	<a href="#">byte</a>	0 ... 1	number of the relay
on_off	<a href="#">byte</a>	0 ... 2	mode of the relay
time	<a href="#">word</a>	0 ... 65535	period of the state change in units of 10ms

Switches the specified relay for the given time to the specified *mode*:

Mode	Description
0	set relay off
1	set relay on
2	enable PWM on relay

**Note**

- The maximum value for time is 65535 (0xFFFF), which will give a maximum one-shot duration of close to 11 minutes.
- The corresponding relay output pulses with the PWM signal according to [Pulse Width Modulation \(PWM\) Related Commands](#) instead of being statically switched on or off.

**Response:** [ACK]

**Example**

```
\iIr\1\1\D300
```

Turns relay number 1 on for 3 seconds.

**See also:**[Input/Output Related Commands](#)[Set Relays On/Off/PWM](#)

## **Enable/Disable Rotary Encoder Reporting**

```
\i I E bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether rotary encoder events are reported (1) or not (0)

Enables or disables the reporting of rotary encoder events (refer to [Rotary Encoder Turn](#)).

### **Note**

- The default value for *on\_off* is 1. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### **Example**

```
\iIE\0
```

Disables the reporting of rotary encoder events.

Not supported by: DPC3020, DPC2060, DPC10xx

### **See also:**

[Rotary Encoder Turn](#)

## **Enable/Disable Keyboard**

```
\i I K E bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether keyboard is on (1) or off (0)

Enables or disables keyboard scanning.

### **Note**

- When keyboard scanning is disabled, reading the keyboard state will return the same result as if no key is pressed (see [Get Keyboard State](#)).
- Enabling/Disabling the keyboard does not change keyboard reporting (see [Enable/Disable Keyboard Reporting](#)), although a disabled keyboard will not report any keys.

- The keyboard is enabled by the default. This can be changed on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iIKE\0
```

Disables the keyboard.

### See also:

[Enable/Disable Keyboard Reporting](#)

## Enable/Disable Keyboard Reporting

```
\i I K R bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether keyboard reporting is on (1) or off (0)

Enables or disables keyboard reporting.

### Note

- When keyboard reporting is on (*on\_off* = 1), any key pressure and release gets reported (see [Key Press/Release](#)) automatically and any running macro is stopped.
- When the terminal mode (see [Terminal Mode](#)) is on, only the key itself gets reported when the key is pressed (this is what is expected from a standard ANSI terminal).
- Switching keyboard reporting off (*on\_off* = 0) avoids stopping macros.
- Keyboard reporting is enabled by the default. This can be changed on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iIKR\1
```

Activates keyboard reporting.

### See also:

[Key Press/Release](#)  
[Enable/Disable Keyboard](#)

[Terminal Mode](#)  
[Go Terminal Mode](#)

## Get Keyboard State

```
\i I K ?
```

Returns the state of all keys asynchronously.

### Note

- When keyboard scanning is disabled, reading the keyboard state will always return 0x00 for all columns (see [Enable/Disable Keyboard](#)).
- Keyboard columns not having assigned a KeybCol # via iLCD Manager XE report all corresponding keys as not pressed.

**Response** [ACK] state\_column\_0 state\_column\_1 ... state\_column\_15 [ACK]  
 :

### Response Example

```
[ACK] [00] [00] [00] [01] [00] [06] [00] [00] [00] [00] [00] [00] [00] [00] [00] [00]
[ACK]
```

In column #3, the button connecting to row #0 is pressed. In column #5, the buttons connecting to rows #1 and #2 are pressed.

### See also:

[Enable/Disable Keyboard](#)  
[Enable/Disable Keyboard Reporting](#)

## Set Baud Rate

```
\i I B byte::port_no long::baud
```

Parameter	Type	Range	Description
port_no	<a href="#">byte</a>	0 ... 2 (0 = currently active)	serial port to be configured
baud	<a href="#">long</a>	300 ... 921600	baud rate

Sets the baud rate according to *baud* for the serial communication-port specified. A *port\_no* of 1 refers to Serial Port 0, a *port\_no* of 2 refers to Serial Port 1 (not enabled by default).

**Note**

- Although the baud rate is usually set via iLCD Manager XE, baud rates higher than 115200 can only be set via this command to prevent loss of communication on slower serial ports.
- Restarting the iLCD Controller with the [Reboot Panel Controller](#) command or via the reset pin causes the controller to start with the baud rate set via iLCD Manager XE.
- When using the value 0 for *port\_no*, the baud rate for the serial port currently in use is changed. If *port\_no* is 0 and the communication-port in use is not a serial port, the command is ignored although an *[ACK]* is returned.
- *baud* must have one of the following values: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600

**Response:** [ACK]

**Example**

```
\iIB\0\L9600
```

Changes the baud rate for the currently used communication-port to 9600 baud.

**See also:**

[Controlling the iLCD via USB](#)

**Get Current Communication-Port**

```
\i I ? C
```

Returns the current communication-port in use.

**Response** [ACK] port\_id [ACK]  
:

port_id	Description
01	serial port 0
02	serial port 1
03	USB port
04	I <sup>2</sup> C port
05	SPI port
06	ethernet port

**Response Example**

```
[ACK] [03] [ACK]
```

The current communication-port is the USB port.

**See also:**

[Set Baud Rate](#)  
[Controlling Options](#)

## **Enable/Disable Communication-Ports**

```
\i I C D bits::port_mask word::timeout
```

Parameter	Type	Range	Description
port_mask	<a href="#">bits</a>	Bits 1 ... 6	bitmask for communication-port
timeout	<a href="#">word</a>	0 ... 65535	timeout in units of 10ms

Disables the communication-ports that are 0-bits in the *port\_mask*. After the *timeout* period, all ports are enabled again. The bits of are defined as follows:

Bit	Description
1	serial port 0
2	serial port 1
3	USB port
4	I <sup>2</sup> C port
5	SPI port
6	ethernet port

**Note**

- Sending this command a second time will restart the timeout.
- Unused bits of the byte are ignored. Hence, all ports are disabled with a *port\_mask* of 0x0 and enabled with 0xFF.

**Response:** [ACK]

**Example**

```
\iICD\x76\D500
```

Disables the USB port for 5 seconds.

Not supported by: DPC3020, DPC2060, DPC10xx



**See also:**[Controlling Options](#)[Get Enabled Communication-Ports](#)[Get Current Communication-Port](#)**Get Enabled Communication-Ports**`\i I C ?`

Returns a port mask where a 0-bit means that the corresponding communication-port is currently disabled.

**Response:** [ACK] port\_mask [ACK]

Bit	Description
1	serial port 0
2	serial port 1
3	USB port
4	I <sup>2</sup> C port
5	SPI port
6	ethernet port

**Response Example**`[ACK] [08] [ACK]`

Currently, all communication-ports except USB are disabled.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**[Controlling Options](#)[Disable Communication-Ports](#)[Get Current Communication-Port](#)**Get Inputs State**`\i I ? I`

Returns the current state of all inputs in a bit mask.

**Note**

- The inputs are reported in a bit orientated manner, where bit 0 of *state* refers to Input #0 and bit 15 refers to Input #15.
- Undefined inputs are reported as low, pins defined as inputs but left unconnected (=floating) can have any value.

**Response:** [ACK] state [ACK]

Parameter	Type	Range	Description
state	<a href="#">word</a>	0x0000 ... 0xFFFF	every set bit represents an active input

**Response Example**

```
[ACK] [02] [80] [ACK]
```

This response means that inputs #7 and #9 (0x280 = Bin 0000 0010 1000 0000) are currently active.

**See also:**

[Input/Output Related Commands](#)

**Get ADC Value**

```
\i I ? A byte::port
```

Parameter	Type	Range	Description
port	<a href="#">byte</a>	0 ... 3	logical port name

Returns the value of any of the 4 ADC (Analog Digital Converter) ports defined as "Analog In" in the "I/O Settings" section of iLCD Manager XE's "Settings" page.

**Note**

- The DPC3050's ADCs have a resolution of 12 bits, resulting in a value range of decimal 0 ... 4095 (0xFFFF) where 0 is an input voltage of 0V and 4095 is an input voltage of 3V or 3.3V, depending on the value of the supply voltage of the iLCD Controller.
- The DPC3080's ADCs have a resolution of 10 bits, resulting in a value range of decimal 0 ... 1023 (0x3FF) where 0 is an input voltage of 0V and 1023 is an input voltage of 3.3V, the value of the supply voltage of the iLCD Controller.
- The DPC3090's ADCs have a resolution of 12 bits, resulting in a value range of decimal 0 ... 4095 (0xFFFF) where 0 is an input voltage of 0V and 4095 is an input voltage of 3.3V, the value of the supply voltage of the iLCD Controller.

**Response:** [ACK] value [ACK]

## Response Example

```
[ACK] [01] [A8] [ACK]
```

This response means the requested ADC currently has a value of 424 (0x1A8).

### See also:

[Input/Output Related Commands](#)

## Set DAC Value

```
\i I A word::value
```

Parameter	Type	Range	Description
value	<a href="#">word</a>	0 ... 1023	DAC output value

This command is used to adjust the DAC output voltage to the specified value. When the system starts up the DAC is adjusted to the value 0.

### Note

- The DAC has a resolution of 10 bits, resulting in a value range of decimal 0 ... 1023 (0x03FF). The decimal value 0 responds to an output voltage of 0V. The decimal value 1023 responds to an output voltage of 3.3V.
- The DAC is available for DCP3090 only.
- This command responds *[NACK]* if the "DAC Out" is not set within the "I/O Settings" tab.

**Response:** [ACK]

### Example

```
\i IA\D1023
```

### See also:

[Input/Output Related Commands](#)  
[setDACValue](#)

## MicroSD Card Related Commands

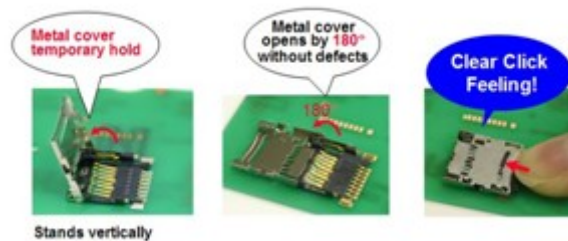
A MicroSD card can be inserted in a card holder on the backside of any iLCD with DPC3080 controller or higher. The iLCD Controller supports low capacity cards as well as HC cards with FAT12, FAT16 and FAT32 filesystem. Long filenames are not supported, all filenames must be entered in 8.3 format and are case-insensitive when using characters between A-Z respectively a-z.

The NT Reserved Byte 0x0C of each file determines if the filename and extension are interpreted (and thus reported) lowercase. When creating files (refer to [Open File](#) and [Make Directory](#)), filename and extension are lowercase only if there are no uppercase characters in the corresponding string.

There is no drive letter to be used, every path has to be declared absolutely, "\" directory separators can not be used and are not allowed within filenames as well. Relative paths like "../xxx" and working directories are not supported.

Graphics and animations can be displayed directly from the MicroSD card (refer to [Display Local Graphic](#) and [Load Animated Graphics](#)).

When a card is inserted it is automatically mounted and ready for data storing and retrieving. When the card is removed, all files are closed and the card is dismounted. You should not remove a card while a file is open. Problems can arise when a file is open for writing, appending or when a file is truncated but not yet closed. When the card is removed in such a case the opened file could remain with wrong length or with garbage data appended to the file. Loss of data might occur as well. To prevent this, use the [Unmount MicroSD Card](#) command before removing the card.



To insert a MicroSD card, slide the connector in the direction of the OPEN-arrow engraved in the metal plate and lift it. Insert the card with the contact area facing the board, then fold the connector back in and push carefully in the direction of the LOCK-arrow until it makes a click sound.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Request Disk Status](#)
- [Make Directory](#)
- [Get File Status](#)
- [Open Directory and Read First Entry](#)
- [Read Next Entry](#)
- [Delete File](#)
- [Open File](#)
- [Close File](#)
- [Tell Position in File](#)
- [Set Position in File](#)
- [Read File](#)
- [Read String from File](#)
- [Write File](#)
- [Write String to File](#)
- [Truncate File](#)
- [Write Application Data to Flash](#)
- [Unmount MicroSD Card](#)
- [Format MicroSD Card](#)

## Request Disk Status

```
\i S ?
```

Requests the free and total disk memory size in Megabyte.

**Response:** [ACK] free total [ACK]

Parameter	Type	Range	Description
free	<a href="#">word</a>	0 ... total disk space	free memory in MB
total	<a href="#">word</a>	0 ... 65535	total memory in MB

### Response Example

```
[ACK] [02] [AA] [04] [00] [ACK]
```

This response means that the card has a total memory size of 1GB (0x400) and currently there are 628MB (0x2AA) free.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)

[Get File Status](#)

[Read File](#)

## Make Directory

```
\i S M string::dirname
```

Parameter	Type	Range	Description
dirname	<a href="#">file</a>	DOS filename (8.3 format)	name of the directory to be created

Creates a new directory depicted by path and name.

### Note

- Empty directories can be deleted just like normal files.
- *dirname* is a 0 terminated string as the name and path for the directory to be made.
- The name of the directory will be converted to uppercase as long as at least one uppercase letter is present.

**Response:** [ACK]

### Example

```
\iSM/DIR1/DIR2\0
```

This will create the directory "DIR2" in directory "DIR1" which in turn is located in the root of the SD-Card. "DIR1" in this example must exist, otherwise an error is set and [NACK] is returned.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)  
[Open Directory and Read First Entry](#)  
[Open File](#)  
[Write File](#)  
[Delete File](#)

## Get File Status

```
\i S G string::filename
```

Parameter	Type	Range	Description
filename	<a href="#">file</a>	DOS filename (8.3 format)	name and path of the file

Returns the status information of the file *filename*.

### Note

- *dirname* is a 0 terminated string as the name and path for the directory to be read.

**Response:** [ACK] status [ACK]

Bytes	Description
0 ... 3	file size in bytes
4	file attributes (refer to table below)
5 ... 7	modification date (see <a href="#">Get Date</a> , no weekday)
8 ... 10	modification time (see <a href="#">Get Time</a> )
11 ... 22	DOS filename (8.3 format) filled up with 0

Bit	Description
-----	-------------

0	write protected
1	hidden
2	system file
4	directory
5	archive

### Example

```
\iSG/DIR/IMG00000.RII\0
```

This will read the status of "IMG00000.RII", which is located in "DIR" which in turn is located in the root of the SD-Card.

### Response Example

```
[ACK]
[00] [00] [96] [CC]
[20]
[0C] [07] [0C]
[0C] [24] [0E]
IMG00000.RII
[ACK]
```

The requested file has a size of 38.604 bytes, has the archive attribute set, the modification date is July 12th, 2012 at 12:36:14 and the filename is IMG00000.RII.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)  
[Open Directory and Read First Entry](#)  
[Read Next Entry](#)  
[Open File](#)

## Open Directory and Read First Entry

```
\i S F string::dirname
```

Parameter	Type	Range	Description
dirname	<a href="#">file</a>	DOS filename (8.3 format)	name of the directory to open

Opens the directory depicted by *dirname* and read the file status of the first entry.

**Note**

- If there are no entries in the directory, no data is sent and a `[NACK]` is returned.
- `dirname` is a 0 terminated string as the name and path for the directory to be read.
- This command can be used to explore the file organization (also refer to [Read Next Entry](#)).

**Response:** `[ACK] status [ACK]`

Bytes	Description
0 ... 3	file size in bytes
4	file attributes (refer to table below)
5 ... 7	modification date (see <a href="#">Get Date</a> , no weekday)
8 ... 10	modification time (see <a href="#">Get Time</a> )
11 ... 22	DOS filename (8.3 format) filled up with 0

Bit	Description
0	write protected
1	hidden
2	system file
4	directory
5	archive

**Example**

```
\iSF/DIR1/DIR2\0
```

This will open the directory "DIR2" which is located in "DIR1" which in turn is located in the root of the SD-Card and read the status of the first entry.

**Response Example**

```
[ACK] [00] [00] [96] [CC] [20] [0C] [07] [0C] [0C] [24] [0E] IMG00000.RII [ACK]
```

The requested file has a size of 38.604 bytes, has the archive attribute set, the modification date is July 12th, 2012 at 12:36:14 and the filename is IMG00000.RII.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)

[Get File Status](#)

[Read Next Entry](#)

[Open File](#)



## Read Next Entry

```
\i S N
```

Reads the status of the next entry in the directory prior specified by the command [Open Directory and Read First Entry](#).

### Note

- If there are no more entries or no [Open Directory and Read First Entry](#) command was sent before, an error is set and `[NACK]` is returned.
- This command can be used to explore the file organization or to realize a slideshow from a folder which can be dynamically filled with graphic files

**Response:** `[ACK] status [ACK]`

Bytes	Description
0 ... 3	file size in bytes
4	file attributes (refer to table below)
5 ... 7	modification date (see <a href="#">Get Date</a> , no weekday)
8 ... 10	modification time (see <a href="#">Get Time</a> )
11 ... 22	DOS filename (8.3 format) filled up with 0

Bit	Description
0	write protected
1	hidden
2	system file
4	directory
5	archive

### Example

```
\iSF/DIR1\0  
\iSN
```

Read the status of the next (in this case second) file in the directory ("DIR1") specified by the preceding command.

### Response Example

```
[ACK] [00] [00] [96] [CC] [20] [0C] [07] [0C] [0C] [24] [0E] IMG00000.RII [ACK]
```

The requested file has a size of 38.604 bytes, has the archive attribute set, the modification date is July 12th, 2012 at 12:36:14 and the filename is IMG00000.RII.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)  
[Get File Status](#)  
[Open Directory and Read First Entry](#)  
[Open File](#)

## Delete File

```
\i S K string::filename
```

Parameter	Type	Range	Description
filename	<a href="#">file</a>	DOS filename (8.3 format)	name and path of the file to be deleted

Deletes a file depicted by path and name via *filename*.

**Note**

- Empty directories can be deleted just like normal files.
- *filename* is a 0 terminated string containing the name and path for the file to be deleted.

**Response:** [ACK]

**Example**

```
\iSK/DIR1/FILE1\0
```

This will delete the file "FILE1" in directory "DIR1" which in turn is located in the root of the SD-Card.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)  
[Make Directory](#)  
[Open File](#)  
[Close File](#)  
[Write File](#)

## Open File

```
\i S O byte::handle byte::mode string::filename
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for the opened file
mode	<a href="#">byte</a>	ASCII: 'R', 'W', 'A'	mode to open the file
filename	<a href="#">file</a>	DOS filename (8.3 format)	name and path of the file

Opens a file specified with *filename* from the SD card and assigns *handle*, according to the following values for *mode*:

ASCII	Hex	Description
R	0x52	Open file in read only mode.
W	0x57	Create file and set file pointer to 0. If the file already exists, it will be deleted and created anew.
A	0x41	The file will be opened or created for modifying or appending. The file pointer will be set after the last byte of the file by default.

### Note

- A maximum of 4 files can be opened at once.
- *filename* is a 0 terminated string describing the name and path for the file to be opened.
- If *mode* is other than 'R', 'W' or 'A', the file is opened in read only mode.
- All files have to be named in 8.3 format in order to be accessible.
- When creating a new file with this command, the filename as well as its extension will be converted to uppercase as long as at least one uppercase letter is present in the corresponding string.
- If you want to open an existing file to add data, the append mode (*mode* = A) has to be chosen. To edit the existing data you have to manually set the file pointer to the desired position (see [Set Position in File](#)).
- All opened files will automatically be closed by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

### Example

```
\iSO\0R\DIR1\DIR2\BACKGR~1.RII\0
```

This will open the file "BACKGROUND.RII" in directory "DIR2" which in turn is located in the directory "DIR1", which is located in the root of the SD-Card. The file will be opened via the first handle (0) and be in read-only mode.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**[MicroSD Card Related Commands](#)[Delete File](#)[Close File](#)[Write File](#)[Make Directory](#)**Close File**

```
\i S C byte::handle
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for opened file

Closes a previously opened file with the ID *handle*.

**Response:** [ACK]

**Example**

```
\iSC\1
```

Closes the file on the handle 1.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**[MicroSD Card Related Commands](#)[Open File](#)[Delete File](#)[Write File](#)[Make Directory](#)**Tell Position in File**

```
\i S P byte::handle
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for opened file

Requests the current position of the file pointer in the file accessed via *handle*.

**Response** [ACK] position [ACK]  
:

Parameter	Type	Range	Description
position	<a href="#">long</a>	0 ... file size	current position of the file pointer

### Response Example

```
[ACK] [00] [0D] [B0] [F5] [ACK]
```

This response means that the file pointer is currently located at byte 897269 (0xDB0F5) of the specified file.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)

[Set Position in File](#)

[Open File](#)

## Set Position in File

```
\i S S byte::handle long::position
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for opened file
position	<a href="#">long</a>	0 ... file size	position of the pointer in the file

Sets the current position of the file pointer in the file accessed via *handle*.

### Note

- When the file is open for writing (W or A as *mode* see [Open File](#)) and the new file position is beyond the end of file, the file is extended to the new position.
- The data in the extended area is undefined.

**Response:** [ACK]

### Example

```
\iSS\2\L100
```

Set the file pointer to byte 100 within the open file.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)  
[Tell Position in File](#)  
[Open File](#)

**Read File**

```
\i S R byte::handle word::count
```

Parameter	Type	Range	Description
handle	byte	0 ... 3	unique identifier for opened file
count	word	0 ... 65535	bytes to read

Reads count *bytes* from a previously opened file with ID *handle* from the current position of the file pointer (see [Set Position in File](#)).

**Note**

- The requested number of bytes are returned as a byte stream.
- After the operation, the file pointer is set to the position after the last byte read.
- If the end of file is reached before all data is sent, the remaining bits are sent as zeroes followed by an *[ACK]*.
- When an end of file condition exists at the beginning of the read command, no data is sent at all, instead a *[NACK]* is sent and the error code is set to END\_OF\_FILE (see [Error Codes](#)).
- When wanting to read all data from a file, read commands have to be issued until a *[NACK]* is returned. The error code END\_OF\_FILE should be checked then. Another approach is using the [Get File Status](#) command to request the filesize and then read the exact number of bytes.

**Response:** [ACK] data [ACK]

**Example**

```
\iSO\2R\DIR1\FILE.EXT\0
\iSS\2\L25
\iSR\2\D4
```

Opens the file "FILE.EXT" in folder "DIR1" with handle 2 in read-only mode, sets the file pointer to position 25 and reads 4 bytes.

**Response Example**

```
[ACK] [00] [01] [02] [03] [ACK]
```

The four bytes requested contain ascending values, the file pointer is moved after the last byte read (in this case position 29).

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)  
[Read String from File](#)  
[Tell Position in File](#)  
[Set Position in File](#)  
[Get File Status](#)  
[Write File](#)

## Read String from File

```
\i Sr byte::handle char::endchar
```

Parameter	Type	Range	Description
byte	handle	0 ... 3	unique identifier for opened file
endchar	<a href="#">byte</a>	ASCII char (0x01 ... 0xFF)	ending character to read up to

Reads string data from a previously opened file (same as [Read File](#)), but the number of bytes returned is determined by the byte value of *endchar*.

### Note

- Data is read until either the end character *endchar* is found or the end of file is reached. In the latter case, the end character is appended although it is not found in the file and a *[NACK]* is sent at the end of the response.
- The file pointer is then set to the byte after the last byte read.
- When end of file condition exists at the beginning of the read command, no data is sent at all, just a *[NACK]* is sent and the error code is set to END\_OF\_FILE (see [Error Codes](#)).

**Response:** [ACK] data [ACK]

### Example

```
\iSO\1A\CSV.TXT\0
\iSr\1;
```

Opens the file "CSV.TXT" in the root folder with handle 1 in append mode and reads data until a ';' character is found.

### Response Example

```
[ACK] 1, 2, 3, 4, 5; [ACK]
```

The file contains ascending ASCII values, the file pointer is moved after the last byte read (in this case position 10). In a comma separated text file, sending the same (second) command again would read the next line.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)

[Read File](#)

[Tell Position in File](#)

[Set Position in File](#)

[Get File Status](#)

[Write File](#)

## Write File

```
\i S W byte::handle word::count byte::b0 ...
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for opened file
count	<a href="#">word</a>	0 ... 65535	number of bytes to write
b0 ...	<a href="#">bytes</a>	0x00 ... 0xFF	data bytes to be written

Writes *count* bytes to a previously opened file starting at the current file position.

### Note

- *count* describes the "raw" data bytes only, which means a quoted command introducer (*0xAA*) counts as one character.
- The file pointer is then set to the byte after the last byte written.
- When entering this command in iLCD Manager XE, the space character (*0x20*) has to be put in by its hex value or `\s`.

**Response:** [ACK]

### Example

```
\iSO\1Wtext.txt\0
\iSW\1\D12Hello\sWorld!\r
\iSO\2Wbinary.bin\0
\iSW\2\D5\x1\x2\x3\x4\x5
```

This example writes "Hello World!" followed by a line break to the file "text.txt" and ascending hex values 1 through 5 to the file "binary.bin". Both files are located in the root folder of the SD card.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx



**See also:**

[MicroSD Card Related Commands](#)

[Open File](#)

[Read File](#)

[Tell Position in File](#)

[Set Position in File](#)

[Get File Status](#)

**Write String to File**

```
\i S w byte::handle string::text_string
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for opened file
text_string	<a href="#">string</a>	ASCII chars (0x01 .. 0xFF)	text string to be written

Writes data in form of *text\_string* to a previously opened file starting at the current file position.

**Note**

- To terminate a string, a NULL character (`\0`) has to be placed.
- The number of bytes written is equal to the length of the string.
- The terminating 0-character is not written to the file.
- The file pointer is updated to the byte after the last byte written.

**Response:** [ACK]

**Example**

```
\iSw\0Hello World!\0
```

Writes the string "Hello World!" to the file opened via handle 0.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)

[Open File](#)

[Read File](#)

[Read String from File](#)

[Tell Position in File](#)

[Set Position in File](#)

[Get File Status](#)

## Truncate File

```
\i S T byte::handle
```

Parameter	Type	Range	Description
handle	<a href="#">byte</a>	0 ... 3	unique identifier for opened file

Truncates the file at the current file position.

### Note

- The new end of file is set to the current position and all data beyond is lost.

**Response:** [ACK]

### Example

```
\iST\1
```

Truncates the file opened via handle 1 at the current file position.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)

[Open File](#)

[Set Position in File](#)

[Tell Position in File](#)

## Write Application Data to Flash

```
\i S A bits::flags string::filename
```

Parameter	Type	Range	Description
flags	<a href="#">bits</a>	Bit 0, 1, 7	flags for the flash update
filename	<a href="#">file</a>	DOS filename (8.3 format)	name and path of the file

This command allows to copy a raw flash file (\*.rid or \*.lcdp-rflash) from the MicroSD card to the internal flash memory of the iLCD Controller, whereat *flags* consists of the following options:

Bit	Description
0	no reboot

1	no verification
7	verbose mode

### Note

- Verbose mode enables the iLCD Controller to send progress information. It issues a "." character every 100kByte written and a "v" character when verifying is started.
- When setting bit 0 (no reboot) in *flags*, the new flash data is not accessible until the iLCD Controller is rebooted. Until then, only a basic flash setup is available.
- To create a raw flash file, use the "Export" feature on the "File" page of iLCD Manager XE.
- The corresponding Java method for this command is [writeApplicationDataToFlash\(\)](#) (part of the class [General](#)).

**Response:** [ACK]

### Example

```
\iSA\x1/DIR1/BACKUP_4.RID\0
```

This example will copy the file "BACKUP\_4.RID" from the directory "DIR1" to the internal flash memory without rebooting.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[MicroSD Card Related Commands](#)  
[Request Disk Status](#)  
[Open Directory and Read First Entry](#)  
[Unmount MicroSD Card](#)

---

## Unmount MicroSD Card

```
\i S U
```

Closes all files and safely removes the MicroSD card at runtime.

### Note

- If animated graphics stored on the card are loaded into an animation control location and running, the fetching of the next frame will reconnect the card immediately.

**Response:** [ACK]

### Example

```
\iSU
```

Unmounts the MicroSD card.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)  
[Request Disk Status](#)  
[java.io.File.unmount](#)

---

## **Format MicroSD Card**

```
\i S f =
```

Formats the memory card. All data on the memory card will be deleted.

**Response:** [ACK]

**Example**

```
\iSf=
```

Formats the MicroSD card.

Not supported by: DPC3080, DPC3050, DPC3020, DPC2060, DPC10xx

**See also:**

[MicroSD Card Related Commands](#)  
[Request Disk Status](#)  
[java.io.File.format](#)

---

## **Time/Date Related Commands**

When the real time clock (RTC) is backed up via a battery (either on-board or external) during power loss, the RTC counts the time/date information even without any supply voltage applied. Leap years are handled automatically.

Find following commands in this chapter as well as in the category "Input/Output..." when using the parameter completion feature of iLCD Manager XE:

- [Set Time](#)
  - [Get Time](#)
  - [Set Date](#)
  - [Get Date](#)
-

## Set Time

```
\i I T byte::hour byte::minute byte::second
```

Parameter	Type	Range	Description
hour	<a href="#">byte</a>	0 ... 23	hour of the clock
minute	<a href="#">byte</a>	0 ... 59	minutes of the clock
second	<a href="#">byte</a>	0 ... 59	seconds of the clock

Sets the time of the real time clock.

**Response:** [ACK]

### Example

```
\iIT\14\15\32
```

Sets the clock to 14:15:32.

**See also:**

[Get Time](#)

[Set Date](#)

## Get Time

```
\i I ? T
```

Gets the current time of the real time clock.

**Response:** [ACK] hour minute second [ACK]

Parameter	Type	Range	Description
hour	<a href="#">byte</a>	0 ... 23	hour of the clock
minute	<a href="#">byte</a>	0 ... 59	minutes of the clock
second	<a href="#">byte</a>	0 ... 59	seconds of the clock

### Response Example

```
[ACK] [0A] [20] [10] [ACK]
```

The time is 10:32:16.

**Note****See also:**

[Set Time](#)  
[Get Date](#)

---

**Set Date**

```
\i I D byte::year byte::month byte::day byte::weekday
```

Parameter	Type	Range	Description
year	<a href="#">byte</a>	0 ... 99	year 2000 ... 2099
month	<a href="#">byte</a>	1 ... 12	month
day	<a href="#">byte</a>	1 ... 31	day
weekday	<a href="#">byte</a>	0 ... 6	weekday as numerical representation

Sets the date of the real time clock.

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Response:** [ACK]

**Example**

```
\iID\12\3\15\4
```

Sets the date to march 15th, 2012 and assigns 4 to the weekday.

**See also:**

[Get Date](#)  
[Set Time](#)

---

**Get Date**

```
\i I ? D
```

Returns the date of the real time clock.

### Note

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Response:** [ACK] year month day weekday [ACK]

Parameter	Type	Range	Description
year	<a href="#">byte</a>	0 ... 99	year 2000 ... 2099
month	<a href="#">byte</a>	1 ... 12	month
day	<a href="#">byte</a>	1 ... 31	day
weekday	<a href="#">byte</a>	0 ... 6	weekday as numerical representation

### Response Example

```
[ACK] [0C] [06] [18] [03] [ACK]
```

The date is June 18th, 2012 and the weekday is 3 (meaning Wednesday when automatically set by iLCD Manager XE).

### Note

#### See also:

[Set Date](#)  
[Get Time](#)

## Pulse Width Modulation (PWM) Related Commands

The iLCD Controllers can control the two relay outputs via two internal separate pulse-width modulation circuits too. This allows driving a speaker or buzzer via relay output 0 to produce a sound like mobile phone's ring tones (use macros to control the sequences of different tone heights and durations). Using a simple R/C filter on any of the two relay outputs can allow the output of a controllable analog voltage as well.

Relay output 1 can use the fixed frequency (1.0 kHz) PWM 1 with variable duty cycle only, relay output 0 can be controlled via PWM 0. The frequency of PWM 0 can be set between 1 Hz and 1 MHz with exception of DPM5050 controller which can be adjusted from 20 Hz to 1 MHz.

Find following commands in this chapter as well as in the category "Input/Output..." when using the parameter completion feature of iLCD Manager XE:

- [Set PWM #0](#)

- [Set PWM #1](#)

## Set PWM #0

```
\i I P long::freq word::duty_cycle
```

Parameter	Type	Range	Description
freq <sup>1</sup>	<a href="#">long</a>	1 ... 1000000	frequency in Hertz
duty_cycle	<a href="#">word</a>	1 ... 9999	duty cycle in units of 0.01%

Configures the PWM 0 (on relay output 0) with the specified parameters.

### Note

- Relay output 0 has to be set to PWM mode for this setting to become relevant (see [Set Relays On/Off/PWM](#)).
- On startup *duty\_cycle* is set to 50% (5000) duty cycle automatically.
- <sup>1</sup> DPM5050 is adjustable from 20 Hz to 1 MHz. [NACK] will be returned if a smaller or a greater value is applied.

**Response:** [ACK]

### Example

```
\iIP\0\L500\D8000
```

After sending this command, the PWM 0 on relay output 0 is running with 500 Hertz and a duty cycle of 80%. The resulting signal will have a rising edge every 2ms (1/500Hz = 0.002s) and stay high for 1.6ms (80% of 2ms).

### See also:

[Set PWM #1](#)  
[Pulse Width Modulation \(PWM\) Related Commands](#)

## Set PWM #1

```
\i I P word::duty_cycle
```

Parameter	Type	Range	Description
duty_cycle	<a href="#">word</a>	1 ... 9999	duty cycle in units of 0.01%

Configures the PWM 1 (on relay output 1) with the specified duty cycle.



**Note**

- The PWM 1 has a fixed frequency of 1kHz.
- Relay output 1 has to be set to PWM mode for this setting to become relevant (see [Set Relays On/Off/PWM](#)).
- On startup *duty\_cycle* is set to 50% (5000) duty cycle automatically.

**Response:** [ACK]

**Example**

```
\iIP\1\D2000
```

After sending this command, the PWM 1 on relay output 1 is running with a duty cycle of 20%. The resulting signal will have a rising edge every 1ms (1/1kHz = 1ms) and stay high for 0.2ms (20% of 1ms).

**See also:**

[Set PWM #0](#)  
[Pulse Width Modulation \(PWM\) Related Commands](#)

**EEPROM Related Commands**

The DPC3090 iLCD Controller offers an on-chip EEPROM for user data. DPC3050 and DPC3080 controllers contain an EEPROM emulation behaving like a real EEPROM, which means that you can write any value into any memory location at any time. This is different to usual Flash memory behavior, which must be erased before overwriting a value. The only difference to a real EEPROM is that in rare cases, the write process may take up to one second due to organizational overhead. During this period no other commands can be carried out. For example, blinking outputs or animated graphics are stopped. Any data sent to the iLCD Controller during this period will be lost as well.

The EEPROM emulation on DPC3050 controllers provides 240 bytes and on DPC3080 496 byte of space; the DPC3090's EEPROM has a size of 4016 bytes. Only the first 9 values are used by the iLCD Controller itself, however these values can be freely written to. Any other location may be used for the controlling application to store and retrieve its private data which is kept, even after a power loss.

The 9 special values affect the default startup values for the LCD

Location	Contents	Range of Values	Reference
0	LCD Contrast	0 ... 255	<a href="#">Set LCD Contrast</a>
1	LCD Backlight Intensity	0 ... 15	<a href="#">Set Backlight Intensity</a>
2	LCD Backlight Mode	0 ... 2	<a href="#">Set Backlight Mode</a>
3	LCD Gamma Value	0 ... 255	<a href="#">Set LCD Gamma Value</a>

and the (optional) PCAP touch screen

Location	Contents	Range of Values	Reference
4	PCAP Number of Fingers	1 ... 5	<a href="#">Set Number of Touch Fingers</a>
5	PCAP Threshold	0 ... 255	iLCD Manager XE
6	PCAP Gain	0 ... 255	iLCD Manager XE
7	PCAP Offset	0 ... 255	iLCD Manager XE
8	PCAP Checksum	0 ... 255	iLCD Manager XE

Although any value can be written to any location, the iLCD Controller will automatically restrict values to the corresponding range when retrieving data from the LCD (first 4) values of the EEPROM.

It is recommended to modify the PCAP values in the "Properties" window found on iLCD Manager XE's "Device" page, the checksum is then automatically calculated and written. If the checksum is not valid, factory default values are loaded at startup and written to EEPROM. To set the factory defaults manually, refer to [Set PCAP Configuration to Factory Default](#). It is also possible to set the PCAP values with the [Write EEPROM](#) command. Therefore the PCAP Checksum is calculated with: Checksum = (byte) (NumberOfFingers + Threshold + Gain + Offset). All values have to be unsigned byte.

The 9 values mentioned above are restored from the EEPROM and the corresponding internal commands (e.g. for setting the backlight intensity) are carried out at startup (also when a [Reboot Panel Controller](#) is issued) and when a [Reset All](#) or a [Reset All and Show Startup Graphic](#) command is sent to the controller.

Please note that writing to the 9 special EEPROM locations via the [Write EEPROM](#) command does not call the associated command. The new settings will only become active upon startup. When you want to set something like the backlight intensity, use the appropriate command instead of – or in addition to – the [Write EEPROM](#) command.

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Get EEPROM Size](#)
- [Erase EEPROM](#)
- [Read EEPROM](#)
- [Write EEPROM](#)
- [Set PCAP Configuration to Factory Default](#)

---

## Get EEPROM Size

`\i E ?`

Retrieves the size of the EEPROM in bytes.

**Response:** [ACK] eeprom\_size [ACK]

Parameter	Type	Range	Description
eeprom_size	<a href="#">word</a>	EEPROM	size of the EEPROM emulation

		size	
--	--	------	--

### Response Example

```
[ACK] [01] [F0] [ACK]
```

This response means that the EEPROM of the controller has a size of 496 (0x1F0) bytes.

#### See also:

[EEPROM Related Commands](#)

[Write EEPROM](#)

[Read EEPROM](#)

### Erase EEPROM

```
\i E E =
```

Erases the EEPROM.

#### Note

- When the EEPROM is erased (all values are written to 0xFF) the 4 special locations (see [EEPROM Related Commands](#)) are loaded from the Flash data after the EEPROM has been formatted.
- The appropriate values in the Flash memory are set via iLCD Manager XE.

**Response:** [ACK]

#### See also:

[EEPROM Related Commands](#)

[Write EEPROM](#)

[Get EEPROM Size](#)

### Read EEPROM

```
\i E R word::address
```

Parameter	Type	Range	Description
address	<a href="#">word</a>	0 ... EEPROM size - 1	address of the byte to read

Reads the contents of the EEPROM at the location specified by *address*.

**Note**

- The value is returned as *[ACK] value [ACK]*. Any location not previously written to will return 0xFF.

**Response:** [ACK]

**Response Example**

```
[ACK] [FF] [ACK]
```

This means that the EEPROM is written to its default value 0xFF and is therefore empty.

**See also:**

[EEPROM Related Commands](#)

[Write EEPROM](#)

[Get EEPROM Size](#)

**Write EEPROM**

```
\i E W word::address byte::data
```

Parameter	Type	Range	Description
address	<a href="#">word</a>	0 ... EEPROM size - 1	address of the byte to write to
data	<a href="#">byte</a>	0 ... 255	data to write

Writes data to the EEPROM at the location specified by address.

**Note**

- If there is a write error, an *[EERR]* (0x10) response instead of *[ACK]* is sent back. If you get this message, the controller usually cannot be used anymore due to excessive write accesses.
- The EEPROM section has a suggested life time of more than 1,000,000 write cycles.

**Response:** [ACK]

**Example**

```
\iEW\D23\5
```

Writes the value of 5 (*data*) into the EEPROM at location 23 (*address*).

**See also:**

[EEPROM Related Commands](#)  
[Read EEPROM](#)  
[Get EEPROM Size](#)

---

**Set PCAP Configuration to Factory Default**

`\i E P D`

Loads the factory default values for PCAP touch screens and writes them to the according EEPROM locations.

**Note**

- After this command is sent, the default values will be loaded at startup until modified (refer to [EEPROM Related Commands](#)).

**Response:** [ACK]

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[EEPROM Related Commands](#)

---

**Power/Watchdog Related Commands****General Information About Power/Watchdog Related Inputs/Outputs**

The iLCD Controllers allow most port pins to be assigned as digital or analog inputs, outputs (pull down or push/pull) or keyboard columns via iLCD Manager XE. All commands referring to port pins below refer to the logical port name, not the physical port pin name.

The logical port names dealing with Power/Watchdog related Inputs/Outputs are as follows:

Port Name	Inputs/Outputs
ARES	watchdog reset output
APWR	PC power off output
ASPWR	disconnect PC's power switch output
APSWI	input from the PC's power switch

If a port function is not previously defined via iLCD Manager XE, the corresponding command is inactive. So, for example, when running the watchdog the ARES function must be assigned to a physical port pin via iLCD Manager XE to enable the pin to go high when the watchdog triggers.

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Set Watchdog Interval](#)
- [Feed Watchdog](#)
- [Shutdown \(Power Off\)](#)
- [Hard Shutdown \(Long Power Off\)](#)
- [Cancel Shutdown](#)
- [Get Power State](#)
- [Reset Motherboard](#)
- [Set Smart Power-Off Mode](#)
- [Set Power-Off Notification On/Off](#)

## **Set Watchdog Interval**

```
\i P W word::interval
```

Parameter	Type	Range	Description
interval	<a href="#">word</a>	0 ... 65535	interval in units of 10ms

Sets the watchdog interval according to *interval*.

### **Note**

- The maximum interval is 655.35 seconds (10.92 minutes).
- Setting the interval causes the current watchdog interval to be retriggered.
- An *interval* of 0 disables the watchdog.
- By default, the watchdog is disabled. Hence it will be automatically disabled on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).
- If the PC fails to trigger the watchdog before its time is expired, the panel controller activates the port pin assigned to the ARES function for a certain time (the "Reset impulse" is 200ms by default, but can be modified on the "Settings" page of iLCD Manager XE), disables the watchdog and shows the watchdog reset message ("Watchdog Reset" by default, but can be modified on the "Settings" page of iLCD Manager XE) on the LCD display.
- If the watchdog-reset message is empty, no watchdog reset message is displayed. If no pin is assigned to the ARES port function, no hardware activity will be carried out on watchdog reset.

**Response:** [ACK]

### **Example**

```
\iPW\D1000
```

Sets the watchdog interval to 10 seconds.

**See also:**

[Power/Watchdog Related Commands](#)  
[Feed Watchdog](#)  
[Shutdown \(Power Off\)](#)

---

**Feed Watchdog**

```
\i P w
```

Resets the watchdog timer.

**Note**

- If the watchdog is enabled, feeding allows the application to further process another watchdog interval without pulling the reset line.
- If the watchdog is disabled this command does nothing although it is acknowledged by an `[ACK]` command.

**Response:** `[ACK]`

**See also:**

[Power/Watchdog Related Commands](#)  
[Set Watchdog Interval](#)

---

**Shutdown (Power Off)**

```
\i P U =
```

Activates the port pin with function APWR assigned to for a short time. The "Power down impulse" is 200ms by default, but can be modified on the "Settings" page of iLCD Manager XE.

**Note**

- If the shutdown message is empty, no shutdown message box is displayed.
- If no pin is assigned to the APWR port function, no hardware activity will be carried out on shutdown.
- The display shows the shutdown message box ("Shutting Down.." by default, but can be changed on the "Settings" page of iLCD Manager XE).
- The shutdown sequence is aborted by the command [Reset All](#).
- The '=' (0x3D) character is used to avoid accidental triggering of the shutdown sequence.

**Response:** `[ACK]`

## Example

```
\iPU=
```

Starts the shutdown sequence of the controlling application.

### See also:

[Power/Watchdog Related Commands](#)  
[Hard Shutdown \(Long Power Off\)](#)

---

## Hard Shutdown (Long Power Off)

```
\i P u =
```

Activates the port pin with function APWR assigned to for a long period. The "Long power down" impulse is 5000ms by default, but can be modified on the "Settings" page of iLCD Manager XE.

### Note

- If the hard-shutdown message is empty, no shutdown message box is displayed.
- If no pin is assigned to the APWR port function, no hardware activity will be carried out on shutdown.
- The display shows the hard-shutdown message box ("Hard Shutdown" by default, but can be changed on the "Settings" page of iLCD Manager XE).
- The '=' (0x3D) character is used to avoid accidental triggering of the shutdown sequence.

**Response:** [ACK]

## Example

```
\iPu=
```

Triggers a hard shutdown of the controlling application.

### See also:

[Power/Watchdog Related Commands](#)  
[Shutdown \(Power Off\)](#)

---

## Cancel Shutdown

```
\i P U C
```

Cancels shutdown sequence.



**Note**

- This command deactivates the port pin with function APWR assigned to and does not wait for the PC's shutdown command anymore (when the power key was pressed before).
- The state of the LCD display and outputs remain unchanged
- Pressing the power down key after this command has been received triggers the power down sequence again.
- If no pin is assigned to the APWR port function, no hardware activity will be carried out.

**Response:** [ACK]

**See also:**

[Power/Watchdog Related Commands Shutdown \(Power Off\)](#)

**Get Power State**

```
\i P ?
```

Returns the power state of the application.

**Note**

- Bit 0 of *state* indicates the power key and bit 7 of state indicates a running power down sequence (only triggered when Smart Power was on at the time of the power key pressure). For more information about smart power mode see [Set Smart Power-Off Mode](#).

**Response:** [ACK] state [ACK]

Parameter	Type	Range	Description
state	<a href="#">bits</a>	Bit 0, 7	power state of the application

Bit	Description
0	power key state pressed
7	power down sequence is running (Smart Power is on)

**Response Example**

```
[ACK] [80] [ACK]
```

This response means that the power key is not pressed and the power down sequence is running (Smart Power is on).

**See also:**

[Power/Watchdog Related Commands](#)  
[Set Smart Power-Off Mode](#)  
[Set Power-Off Notification On/Off](#)  
[Set Watchdog Interval](#)

---

**Reset Motherboard**

```
\i P ! =
```

Resets the motherboard of the controlling application.

**Note**

- This command disables the watchdog and activates the pin assigned to the ARES function for a certain period defined via iLCD Manager XE. Outputs and LCD display remain in the previous state, which means that if the customer should see a message and/or the outputs should have a certain state while rebooting the PC, the corresponding sequences have to be sent by the PC before triggering the reset sequence.
- The '=' (0x3D) character is used to avoid accidental triggering of the reset sequence.

**Response:** [ACK]

**Example**

```
\iP!=
```

Resets the motherboard.

**See also:**

[Power/Watchdog Related Commands](#)

---

**Set Smart Power-Off Mode**

```
\i P S bool::on_off
```

Parameter	Type	Range	Description
on_off	<a href="#">bool</a>	0 ... 1	determines whether smart power-off is on (1) or off (0)

Enables or disables smart power-off functionality.

**Note**

- When smart power-off is on (*on\_off* = 1), pressing the power switch does not activate the main board's power pin directly, but sends the sequence "*\* [ACK]*" to the PC only. The PC then has to carry out the corresponding power down sequence via software.
- This sequence is also sent when the power switch is pressed and power-off notification is on (see [Set Power-Off Notification On/Off](#)).
- When smart power-off is off (*on\_off* = 0) the power switch is directly connected to the main board's power pin.
- The default value for *on\_off* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

**Example**

```
\iPS\1
```

Enables the smart power-off mode and transfers control over power management to the PC.

**See also:**

[Power/Watchdog Related Commands](#)  
[Get Power State](#)

**Set Power-Off Notification On/Off**

```
\i P N bool::on_off
```

Parameter	Type	Range	Description
<i>on_off</i>	<a href="#">bool</a>	0 ... 1	determines whether power-off notifications appear (1) or not (0)

Enables or disables power switch notification.

**Note**

- When the power switch is pressed, the sequence "*\* [ACK]*" is sent to the PC when the notification is on (*on\_off* = 1). When the notification is off (*on\_off* = 0) the PC has to poll the status by the [Get Power State](#) command.
- Even when the notification is off, the shutdown sequence is sent when smart power-off is on (see [Set Smart Power-Off Mode](#)).
- The PC's power switch has to pull low the input pin with function APSWI assigned to.
- The default value for *on\_off* is 1, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#).

**Response:** [ACK]

## Example

```
\iPN\0
```

This example deactivates power-off notifications.

### See also:

[Power/Watchdog Related Commands](#)  
[Get Power State](#)

## Extra Commands

Find following commands in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

- [Set Message Offset](#)
- [Set Graphic Offset](#)
- [Set Macro Offset](#)
- [Set Font Offset](#)
- [Set Message Name Prefix](#)
- [Set Graphic Name Prefix](#)
- [Set Macro Name Prefix](#)
- [Set Font Name Prefix](#)
- [Set Message Name Suffix](#)
- [Set Graphic Name Suffix](#)
- [Set Macro Name Suffix](#)
- [Set Font Name Suffix](#)

## Set Message Offset

```
\i X O T word::offset
```

Parameter	Type	Range	Description
offset	<a href="#">word</a>	0 ... max. text message index - 1	offset of the message index

Sets a message index offset (see [Write Text Message](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the messages for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Message Offset to 10 will automatically display the German messages instead of the corresponding English ones.
- When addressing text messages by name, a prefix or suffix can be used to realize language switching. (see [Set Message Name Prefix](#) or [Set Message Name Suffix](#)).

- The default value for *offset* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXOT\D5
\iDt\D10
```

Sets the message offset to 5. The subsequently called command to write the message with index 10 will now put out the message with index 15 instead.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Write Text Message](#)  
[Draw Touch Field Text Message](#)  
[Set Touch Field Text Message](#)  
[Set Graphic Offset](#)  
[Set Macro Offset](#)  
[Set Font Offset](#)

## Set Graphic Offset

```
\i X O G word::offset
```

Parameter	Type	Range	Description
offset	<a href="#">word</a>	0 ... max. graphic index - 1	offset of the graphic index

Sets an offset for the graphic index (see [Display Local Graphic](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the graphics for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Graphic Offset to 10 will automatically draw the German graphics instead of the corresponding English ones.
- When addressing graphics by name, a prefix or suffix can be used to realize language switching. (see [Set Graphic Name Prefix](#) or [Set Graphic Name Suffix](#)).
- The default value for *offset* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the

reset commands without loosing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXOG\D10
\iG\D1
```

Sets the graphic offset to 10. The subsequently called command to draw the graphic with index 1 will now use the graphic with index 11 instead.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Display Local Graphic](#)  
[Erase Animation Frame Area](#)  
[Load Animated Graphics](#)  
[Set Text/Graphic Orientation](#)  
[Reset All](#)  
[Set Message Offset](#)  
[Set Macro Offset](#)  
[Set Font Offset](#)

## Set Macro Offset

```
\i X O O word::offset
```

Parameter	Type	Range	Description
offset	<a href="#">word</a>	0 ... max. macro index - 1	offset of the macro index

Sets an offset for the macro index (see [Execute Macro](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the macros for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Macro Offset to 10 will automatically execute the German macros instead of the corresponding English ones.
- When addressing macros by name, a prefix or suffix can be used to realize language switching. (see [Set Macro Name Prefix](#) or [Set Macro Name Suffix](#)).
- The default value for *offset* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without loosing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXOO\D10
\iOE\D10
```

Sets the macro offset to 10. The subsequently called command to execute the macro with index 1 will now execute the macro with index 11 instead.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Draw Touch Field Text Message](#)  
[Set Touch Field Text Message](#)  
[Execute Touch Break Macro](#)  
[Execute Touch Make Macro](#)  
[Set Touch Field Break Macro](#)  
[Set Touch Field Make Macro](#)  
[Jump to Macro](#)  
[Reset All](#)  
[Set Message Offset](#)  
[Set Graphic Offset](#)  
[Set Font Offset](#)

## Set Font Offset

```
\i X O F word::offset
```

Parameter	Type	Range	Description
offset	<a href="#">word</a>	0 ... max. font index - 1	offset of the font index

Sets an offset for the font index (see [Set Font](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the fonts for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Font Offset to 10 will automatically select the German fonts instead of the corresponding English ones.
- When addressing fonts by name, a prefix or suffix can be used to realize language switching. (see [Set Font Name Prefix](#) or [Set Font Name Suffix](#)).
- The default value for *offset* is 0. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXOF\D3
\iAF\D1
\iDTHello World!\0
```

Sets the font offset to 3. The subsequently called command to select the font with index 1 will now set the font with index 4 instead.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Set Font](#)  
[Write Text Message](#)  
[Set Touch Field Text Message](#)  
[Reset All](#)  
[Set Message Offset](#)  
[Set Graphic Offset](#)  
[Set Macro Offset](#)

### See also:

[Set Message Offset](#)

## Set Message Name Prefix

```
\i X P T string::prefix
```

Parameter	Type	Range	Description
prefix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent message name (see [Write Text Message](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the message names for the English language are prefixed with "en\_" and for the German language with "de\_", the command

```
\iDt\message\0
```

would display the message "en\_message" or "de\_message", dependent on the prior defined message prefix.



- When addressing text messages by index, an offset can be used to realize language switching. (see [Set Message Offset](#)).
- Remove any previously set Message Name Prefix by entering an empty string as *prefix*:

```
\iXPT\0
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXPTen_\0
```

Set the message name prefix to "en\_" and will subsequently only call a message that start with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Draw Touch Field Text Message](#)  
[Set Touch Field Text Message](#)  
[Get Text Message Extent](#)  
[Write Text Message](#)  
[Set Message Offset](#)  
[Set Message Name Suffix](#)  
[Set Graphic Name Prefix](#)  
[Set Macro Name Prefix](#)  
[Set Font Name Prefix](#)

## Set Graphic Name Prefix

```
\i X P G string::prefix
```

Parameter	Type	Range	Description
prefix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent graphic name (see [Display Local Graphic](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the graphic names for the English language are prefixed with "en\_" and for the German language with "de\_", the command

```
\iG\mgraphic\0
```

would display the graphic "en\_graphic" or "de\_graphic", dependent on the prior defined graphic prefix.

- When addressing graphics by index, an offset can be used to realize language switching (see [Set Graphic Offset](#)).
- Remove any previously set Graphic Name Prefix by entering an empty string as *prefix*:

```
\iXPG\0
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXPGen_ \0
```

Set the graphic name prefix to "en\_" and will subsequently only set graphics that start with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Display Local Graphic](#)  
[Load Animated Graphics](#)  
[Set Message Offset](#)  
[Set Graphic Name Suffix](#)  
[Set Message Name Prefix](#)  
[Set Macro Name Prefix](#)  
[Set Font Name Prefix](#)

## Set Macro Name Prefix

```
\i X P O string::prefix
```

Parameter	Type	Range	Description
prefix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent macro name.

## Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the macro names for the English language are prefixed with "en\_" and for the German language with "de\_", the command

```
\iOE\macro\0
```

would execute macro "en\_macro" or "de\_macro", dependent on the prior defined macro prefix.

- When addressing macros by index, an offset can be used to realize language switching. (see [Set Macro Offset](#)).
- Remove any previously set Macro Name Prefix by entering an empty string as *prefix*:

```
\iXPO\0
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

## Example

```
\iXPOen_\0
```

Set the macro name prefix to "en\_" and will subsequently only consider macros that start with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

## See also:

[Draw Touch Field Text Message](#)  
[Set Touch Field Text Message](#)  
[Execute Macro](#)  
[Execute Touch Break Macro](#)  
[Execute Touch Make Macro](#)  
[Set Touch Field Break Macro](#)  
[Set Touch Field Make Macro](#)  
[Jump to Macro](#)  
[Set Macro Offset](#)  
[Set Macro Name Suffix](#)  
[Set Message Name Prefix](#)  
[Set Graphic Name Prefix](#)  
[Set Font Name Prefix](#)

## Set Font Name Prefix

```
\i X P F string::prefix
```

Parameter	Type	Range	Description
prefix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent font name (see [Set Font](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the font names for the English language are prefixed with "en\_" and for the German language with "de\_", the command

```
\iAF\mfont\0
```

would select the font "en\_font" or "de\_font", dependent on the prior defined font prefix.

- When addressing fonts by index, an offset can be used to realize language switching. (see [Set Font Offset](#)).
- Remove any previously set Font Name Prefix by entering an empty string as *prefix*:

```
\iXPF\0
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXPFen_\0
```

Set the font name prefix to "en\_" and will subsequently only select fonts that start with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Set Touch Field Text Message](#)  
[Set Font](#)  
[Get Text Message Extent](#)  
[Write Text Message](#)  
[Set Font Offset](#)  
[Set Font Name Suffix](#)  
[Set Message Name Prefix](#)

[Set Graphic Name Prefix](#)  
[Set Macro Name Prefix](#)

## **Set Message Name Suffix**

```
\i X S T string::suffix
```

Parameter	Type	Range	Description
suffix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent message name (see [Write Text Message](#)).

### **Note**

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the message names for the English language are suffixed with "\_en" and for the German language with "\_de", the command

```
\iDt\message\0
```

would display the message "message\_en" or "message\_de", dependent on the prior defined message suffix.

- When addressing text messages by index, an offset can be used to realize language switching. (see [Set Message Offset](#)).
- Remove any previously set Message Name Suffix by entering an empty string as *suffix*:

```
\iXST\0
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### **Example**

```
\iXST_en\0
```

Set the message name suffix to "\_en" and will subsequently only call a message that end with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Draw Touch Field Text Message](#)  
[Set Touch Field Text Message](#)  
[Get Text Message Extent](#)  
[Write Text Message](#)  
[Set Message Offset](#)  
[Set Message Name Prefix](#)  
[Set Graphic Name Suffix](#)  
[Set Macro Name Suffix](#)  
[Set Font Name Suffix](#)

[Set Message Name Prefix](#)  
[Set Message Offset](#)  
[Write Text Message](#)

**Set Graphic Name Suffix**

```
\i X S G string::suffix
```

Parameter	Type	Range	Description
suffix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent graphic name (see [Display Local Graphic](#)).

**Note**

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the graphic names for the English language are suffixed with "\_en" and for the German language with "\_de", the command

```
\iG\mgraphic\0
```

would display the graphic "graphic\_en" or "graphic\_de", dependent on the prior defined graphic prefix.

- When addressing graphics by index, an offset can be used to realize language switching. (see [Set Graphic Offset](#)).
- Remove any previously set Graphic Name Suffix by entering an empty string as *suffix*:

```
\iXSG\0
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXSG_en\0
```

Set the graphic name suffix to "\_en" and will subsequently only use graphics that end with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Display Local Graphic](#)  
[Load Animated Graphics](#)  
[Set Graphic Offset](#)  
[Set Graphic Name Prefix](#)  
[Set Message Name Suffix](#)  
[Set Macro Name Suffix](#)  
[Set Font Name Suffix](#)

---

## Set Macro Name Suffix

```
\i X S O string::suffix
```

Parameter	Type	Range	Description
suffix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent macro name (see [Execute Macro](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the macro names for the English language are suffixed with "\_en" and for the German language with "\_de", the command

```
\iOE\macro\0
```

would execute macro "macro\_en" or "macro\_de", dependent on the prior defined macro prefix.

- When addressing macros by index, an offset can be used to realize language switching. (see [Set Macro Offset](#)).
- Remove any previously set Macro Name Suffix by entering an empty string as *suffix*:

```
\iXSO\0
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXSO_en\0
```

Set the macro name suffix to "\_en" and will subsequently only consider macros that end with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Draw Touch Field Text Message](#)  
[Set Touch Field Text Message](#)  
[Execute Macro](#)  
[Execute Touch Break Macro](#)  
[Execute Touch Make Macro](#)  
[Set Touch Field Break Macro](#)  
[Set Touch Field Make Macro](#)  
[Jump to Macro](#)  
[Set Macro Offset](#)  
[Set Macro Name Prefix](#)  
[Set Message Name Suffix](#)  
[Set Graphic Name Suffix](#)  
[Set Font Name Suffix](#)

---

## Set Font Name Suffix

```
\i X S F string::suffix
```

Parameter	Type	Range	Description
suffix	<a href="#">string</a>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent font name (see [Set Font](#)).

### Note

- This functionality can be used to support multiple languages without conditional commands or macros. For example, if the font names for the English language are suffixed with "\_en" and for the German language with "\_de", the command



```
\iAF\mfont\0
```

would select the font "font\_en" or "font\_de", dependent on the prior defined font prefix.

- When addressing fonts by index, an offset can be used to realize language switching. (see [Set Font Offset](#)).
- Remove any previously set Font Name Suffix by entering an empty string as *suffix*:

```
\iXSF\0
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the commands [Reset All](#) or [Reboot Panel Controller](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset commands without losing e.g. the selected language. This setting is ignored in firmware versions < 3.02.

**Response:** [ACK]

### Example

```
\iXSF_en\0
```

Set the font name suffix to "\_en" and will subsequently only set fonts that end with these characters.

Not supported by: DPC3020, DPC2060, DPC10xx

### See also:

[Set Touch Field Text Message](#)  
[Set Font](#)  
[Get Text Message Extent](#)  
[Write Text Message](#)  
[Set Font Offset](#)  
[Set Font Name Prefix](#)  
[Set Message Name Suffix](#)  
[Set Graphic Name Suffix](#)  
[Set Macro Name Suffix](#)

---

## Data Sent by the iLCD Controller

The responses to commands are described in the corresponding [Command Description](#). In some cases the iLCD Controller sends spontaneous data, these events will be described in the following chapters:

- [Startup](#)
- [Power Key Pressed](#)
- [Calibrate Touch Screen Done](#)
- [Key Press/Release](#)
- [Rotary Encoder Turn](#)
- [Touch Field Press/Release](#)
- [Touch Field Press/Release + Coordinate of Event](#)
- [Moving Coordinate of Touch Field Press](#)

## Startup

```
# [ACK]
```

At startup, the panel controller sends the above message to the default communications port if sending the startup message is activated on the "Settings" page of iLCD Manager XE. This is also true if the controller has been rebooted by the [Reboot Panel Controller](#) command.

---

## Power Key Pressed

```
* [ACK]
```

This sequence is sent when the power key is pressed and the power-off notification is turned on independently if smart power is on or off. If smart power is on and the PC does not react within 5 seconds (can be overwritten via iLCD Manager XE) a reset followed by a hard power down is started. Within this 5 seconds the \* [ACK] sequence is repeated every second. When smart power is off, the sequence is sent only once per power key press.

### See also:

[Set Smart Power-Off Mode](#)  
[Set Power-Off Notification On/Off](#)

---

## Calibrate Touch Screen Done

This sequence is sent after successful touch screen calibration when it was activated via the [Calibrate Touch Screen and Report](#) command:

```
T [ACK]
```

If the calibration fails, the following sequence is sent:

```
T [NACK]
```

### See also:

[Calibrate Touch Screen and Report](#)

---

## Key Press/Release

When keyboard reporting is on (see at [Enable/Disable Keyboard Reporting](#)), the pressing of a key is indicated by

```
K key_char [ACK]
```

and the key release is reported as

```
k key_char [ACK]
```

where *key\_char* is the character assigned to the key via iLCD Manager XE.

### See also:

[Enable/Disable Keyboard](#)

---

## Rotary Encoder Turn

When an incremental rotary encoder is connected and activated in iLCD Manager XE, any turn is reported via the interface as follows

```
I detents [ACK]
```

where *detents* is a signed byte value indicating the turned detents of clockwise (positive value) or counter-clockwise (negative value) motion. The single detents are accumulated and reported after the time specified by "Report after" on the "Settings" page of iLCD Manager XE.

Reporting rotary encoder events can be disabled by the command [Enable/Disable Rotary Encoder Reporting](#).

### See also:

[Enable/Disable Rotary Encoder Reporting](#)

---

## Touch Field Press/Release

When a previously defined touch field (see [Create/Define Touch Field](#)) is pressed/released, the press of a field is indicated by

```
K key_char [ACK]
```

and the release is reported as follows

```
k key_char [ACK]
```

where *key\_char* is the non-zero character assigned to the touch field previously defined.

Reporting press/release events of touch fields with a *key* other than zero can be disabled by the command [Enable/Disable Touch Field Reporting](#).

### Multi Touch

When more than one touch points are evaluated simultaneously, a point ID is appended to the report for make

```
K key_char point_id [ACK]
```

as well as for break events:

```
k key_char point_id [ACK]
```

where *point\_id* is a unique ID automatically assigned to the touch point (finger).

The number of concurrently reported touch points can be set by the command [Set Number of Touch Points](#).

### See also:

[Create/Define Touch Field](#)  
[Enable/Disable Touch Field Reporting](#)  
[Set Number of Touch Points](#)  
[Touch Field Press/Release + Coordinate of Event](#)

---

## Touch Field Press/Release + Coordinate of Event

When a previously defined touch field (see [Create/Define Touch Field](#)) is pressed/released, and the reporting of the coordinates is switched on via [Enable/Disable Reporting Touch-Coordinates](#), the press of a field is indicated by

```
K key_char ev_coord_x ev_coord_y [ACK]
```

and the release is reported as follows

```
k key_char ev_coord_x ev_coord_y [ACK]
```

*key\_char* is the non-zero character assigned to the touch field previously defined and *ev\_coord\_x* / *ev\_coord\_y* is the coordinate where the event took place relative to the upper left corner of the corresponding touch field.

### Multi Touch

When more than one touch point are evaluated simultaneously, a point ID is inserted to the report for make

```
K key_char point_id ev_coord_x ev_coord_y [ACK]
```

as well as for break events:

```
k key_char point_id ev_coord_x ev_coord_y [ACK]
```

where *point\_id* is a unique ID automatically assigned to the touch point (finger).

The number of concurrently reported touch points can be set by the command [Set Number of Touch Points](#).

#### Note

- The coordinate value may become negative when the touch field is released. The minimum / maximum coordinate value being reported can even fall outside the physical screen by one pixel when converted to absolute coordinates.

#### See also:

[Create/Define Touch Field](#)  
[Enable/Disable Reporting Touch-Coordinates](#)

---

## Moving Coordinate of Touch Field Press

When a previously defined touch field (see [Create/Define Touch Field](#)) is pressed and the location of the press is changed, and the reporting of the movements is switched on via [Enable/Disable Reporting Movements](#), the movement is indicated by

```
M key_char ev_coord_x ev_coord_y [ACK]
```

where *key\_char* is the non-zero character assigned to the touch field previously defined and *ev\_coord\_x* / *ev\_coord\_y* is the coordinate of press relatively to the upper left corner of the corresponding touch field.

#### Multi Touch

When more than one touch point are evaluated simultaneously, a point ID is appended to the movement reports

```
M key_char point_id ev_coord_x ev_coord_y [ACK]
```

where *point\_id* is a unique ID automatically assigned to the touch point (finger).

The number of concurrently reported touch points can be set by the command [Set Number of Touch Points](#).

#### Note

- The coordinate may turn negative. The minimum / maximum coordinate values which can be reported can even fall outside the physical screen by one pixel when converted to absolute coordinates.
- The distance threshold for movement reports can be modified with the command [Set Threshold for Movement Reporting](#).

**See also:**

[Create/Define Touch Field](#)  
[Enable/Disable Reporting Movements](#)  
[Set Threshold for Movement Reporting](#)

**Error Codes**

Whenever a command returns unsuccessfully an error code can be retrieved by issuing the [Get Last Error Code](#) right after the command that returned a `[NACK]`.

**Note**

- The controller always returns the error code as Hex values which need to be calculated to decimal values.
- Keep in mind that some values are interpreted in iLCD Manager XE's "Response from iLCD" panel (see [Syntax used in iLCD Manager XE](#))

Hex Value	Decimal Value	INTERNAL	Description
[00][C9]	201	E_FF_NOT_READY	SD-Card not ready or card missing
[00][CA]	202	E_FF_NO_FILE	Could not find the file
[00][CB]	203	E_FF_NO_PATH	Could not find the path
[00][CC]	204	E_FF_INVALID_NAME	Invalid name in pathname or filename
[00][CD]	205	E_FF_INVALID_DRIVE	Invalid drive – should not occur
[00][CE]	206	E_FF_DENIED	Action denied Writing to file which is open for read mode File cannot created due to same name directory File cannot created due to directory table or disk full
[00][CF]	207	E_FF_EXIST	The file is already existing
[00][D0]	208	E_FF_RW_ERROR	The function failed due to a disk RW error or internal error
[00][D1]	209	E_FF_WRITE_PROTECTED	Open for write and disk is write protected
[00][D2]	210	E_FF_NOT_ENABLED	The drive has no working area – should not occur
[00][D3]	211	E_FF_NO_FILESYSTEM	There is no valid FAT filesystem on the disk
[00][D4]	212	E_FF_INVALID_OBJECT	The file object is invalid – should not occur
[00][D5]	213	E_FF_MKFS_ABORTED	Make directory aborted due to Disk size is too small Invalid parameter Wrong cluster size. This can occur when the number of clusters becomes around 0xFF7 or

			0xFFFF7
[00][D6]	214	E_FF_DISKFULL	Disk is full
[00][E6]	230	E_SDC_PARAM	Illegal parameter in command
[00][E7]	231	E_SDC_OPEN	The file is already open, when issuing an open command
[00][E8]	232	E_SDC_NOPEN	The file is not open
[00][E9]	233	E_SDC_NODISK	No card inserted
[00][EA]	234	E_SDC_EOF	End of file already reached when starting read command
[00][EB]	235	E_SDC_DIREND	End of directory reached when reading first or next directory entry
[00][EC]	236	E_SDC_READ_FAIL	An error occurred during read access
[03][E9]	1001	E_COMMAND	Illegal command
[03][EA]	1002	E_X_OUT_OF_RANGE	X coordinate out of range
[03][EB]	1003	E_Y_OUT_OF_RANGE	Y coordinate out of range
[03][EC]	1004	E_NOT_BOOLEAN	Parameter not boolean
[03][ED]	1005	E_VAL_OUT_OF_RANGE	Value out of range
[03][EE]	1006	E_PARNOT_ZERO	Parameter must not be zero
[03][EF]	1007	E_W_OUT_OF_RANGE	Width out of range
[03][F0]	1008	E_H_OUT_OF_RANGE	Height out of range
[03][F1]	1009	E_LCD_NOFONT	Font not found
[03][F2]	1010	E_FIXEDTEXT	Text message does not exist
[03][F3]	1011	E_GR_NOTFOUND	Local graphic not found
[03][F4]	1012	E_GR_INVTYPE	Invalid graphic type specified
[03][F5]	1013	E_GR_FRAMEIDX	Invalid frame index specified
[03][F6]	1014	E_GR_ANIMIDX	Invalid animation control index specified
[03][F7]	1015	E_GR_ANIMASS	Animation control index not assigned
[03][F8]	1016	E_IO_BAUDRATE	Invalid baudrate specified
[03][F9]	1017	E_IO_COMPORT	Invalid com port specified
[03][FA]	1018	E_IO_OUTPAR	Output number out of range
[03][FB]	1019	E_IO_OUTDEFINED	Output output not defined
[03][FC]	1020	E_IO_OUTMODE	Invalid output mode
[03][FD]	1021	E_IO_ADCCHANNEL	Adc channel out of range
[03][FE]	1022	E_IO_ADCENABLE	Adc channel not enabled
[03][FF]	1023	E_IO_FREQU	Frequency out of range
[04][00]	1024	E_IO_DUTY	Duty cycle out of range

[04][01]	1025	E_IO_REL_NUMBER	Relay number out of range
[04][02]	1026	E_IO_REL_MODE	Invalid relay mode specified
[04][03]	1027	E_IO_RTC_READ	RTC read error
[04][04]	1028	E_IO_RTC_WRITE	RTC write error
[04][05]	1029	E_IO_TM_HOUR	Invalid hour specified
[04][06]	1030	E_IO_TM_MINUTE	Invalid minute specified
[04][07]	1031	E_IO_TM_SECOND	Invalid second specified
[04][08]	1032	E_IO_DT_MONTH	Invalid month specified
[04][09]	1033	E_IO_DT_DAY	Invalid day specified
[04][0A]	1034	E_IO_DT_DAYOFWEEK	Invalid day of week specified
[04][0B]	1035	E_TCH_OUT_OF_RANGE	Touch field index out of range
[04][0C]	1036	E_TCH_NOTCALIB	Touch screen not calibrated
[04][0D]	1037	E_TCH_ACTIVE	Touch field is not active
[04][0E]	1038	E_TCH_MAXFIELD	Touch field index out of range
[04][0F]	1039	E_TCH_TEXT_NF	Text message does not exist
[04][10]	1040	E_MEM_OUT_OF_RANGE	Memory position out of range
[04][11]	1041	E_MEM_POS_EMPTY	Memory position is empty
[04][12]	1042	E_MEM_MAXCURSOR	Cursor index out of range
[04][13]	1043	E_MEM_CURSORNF	Cursor memory index not assigned
[04][14]	1044	E_MAC_DEPTH	Macro depth nesting exceeded
[04][15]	1045	E_MAC_NOT_FOUND	Macro not found
[04][16]	1046	E_FLASH_SETSECTOR	Sector not found
[04][17]	1047	E_FLASH_BLOCK	NAND block out of range
[04][18]	1048	E_FLASH_APPLADDR	Illegal flash address specified
[04][19]	1049	E_EEP_ERASE	EEPROM erase error
[04][1A]	1050	E_EEP_MAXSIZE	EEPROM address out of range
[04][1B]	1051	E_VIEWP_INVALID	Invalid viewport specified
[04][20]	1056	E_MEM_BLOCKED	Screen buffer is blocked for drawing or displaying
[04][21]	1057	E_MEM_USED	Screen buffer is not available for displaying
[04][22]	1058	E_VIEWPORT	Command not allowed in a viewport
[04][23]	1059	E_SCALEERR	Command not allowed when scaling != 1
[04][24]	1060	E_RII_MD5	Graphics header in flash does not match graphic data on SDC
[04][25]	1061	E_STR_TOO_LONG	Input string exceeds maximum length



[04][B0]	1200	E_PF_TOOBIG	File is too big to fit NAND flash
[04][B1]	1201	E_PF_INVALID_DATA	Found invalid data in raw flash file
[04][B2]	1202	E_PF_INVALID_CNTRL	Data in raw flash file not applicable for current iLCD Controller
[04][B3]	1203	E_PF_N_INDEX_ERASE	Failed to erase NAND flash index
[04][B4]	1204	E_PF_N_BLOCK_ERASE	Failed to erase NAND flash block
[04][B5]	1205	E_PF_N_WRITE	Failed to write data to NAND flash
[04][B6]	1206	E_PF_N_INDEX_WRITE	Failed to write index data to NAND flash
[04][B7]	1207	E_PF_VALIDATE	Failed to validate written flash
[04][BA]	1210	E_IO_TIMESSET	Invalid date or time specified
[04][BB]	1211	E_GR_ORIENTATION	Command not allowed when text/graphic rotated
[04][BC]	1212	E_GR_NO_CROP	Cropping of animated graphics not allowed

**See also:**

[Get Last Error Code](#)

---

## Controlling Options

iLCD panels can be controlled via RS232 (also RS422 and RS485), USB, SPI and I<sup>2</sup>C as well as TCP/IP using the optional Ethernet interface board.

However, use of the serial SPI and I<sup>2</sup>C protocols is only recommended if there is no RS232 connection available. The RS232 is the fastest of the serial interfaces and is intended for communication between two equitable devices. SPI and I<sup>2</sup>C on the other hand are master-slave protocols, hence the iLCD panel is not able to send data to the application unless actively queried. Therefore, responses and asynchronous data from the iLCD panel has to be polled or an additional wire must be used for signaling data availability.

It is possible to connect more than one of the interfaces simultaneously. Since all ports share a single input buffer, it must be ensured that the interfaces don't interrupt each other mid-command. The outgoing data is always sent to the port the last incoming byte was received from, switching is done as soon as a byte from another interface enters the input buffer.

The following chapters describe the different ways to control iLCD panels:

- [Controlling the iLCD via I<sup>2</sup>C](#)
- [Controlling the iLCD via SPI](#)
- [Controlling the iLCD via USB](#)
- [Controlling the iLCD via Ethernet](#)

## Controlling the iLCD via USB

The USB port of Color iLCD Controllers is implemented as a composited device consisting of a HID (Human Interface Device) and a WinUSB device. The HID part allows iLCD Controllers to be operated on any Windows operating system without having to install a special driver. The WinUSB driver offers a significantly increased transmission rate and is automatically installed with iLCD Manager XE.

A program - including the source code - for writing raw flash data files to an iLCD device can be found at <http://www.demmel.com/download/ilcd/iloader.zip>. This program shows how to handle the USB port's HID driver to communicate with an iLCD device. It is written in standard C and can be easily adapted to other operating systems.

---

## Controlling the iLCD via Serial Port

The iLCD Controller has connections to the serial port's RX, TX, CTS, RTS and GND pin only; the RTS pin is ignored. Please note that the controlling application has to monitor the CTS pin to avoid overflowing the controller's buffer, which has a size of 128 characters for all iLCD Controllers up to DPC3050 and 1024 characters for DPC3080 and higher. The command [Get Input Buffer Size](#) can be used to request the size of the input buffer. Normally, all commands can be sent, even without monitoring the CTS pin when the application waits for the responding `[ACK]` character, but we recommend using the hardware handshake if possible. Most commands sent to the iLCD Controller will fit into this buffer. Exceptions are commands like [Write Text](#), [Write Scan Line](#) or [Write File](#), in which the length of the parameter is not limited. If no CTS monitoring is used, the user has to take care to break the data into smaller chunks and to wait for the `[ACK]` character after each block of data to avoid overflowing the input buffer.

When using RS-422 or RS-485 no monitoring of CTS is available therefore the user must take care not to allow the input buffer to be overrun. If a buffer overflow occurs, the controller finishes any currently running output sequence and then sends an OVR flag (0x19). This is not done when the communication runs via I<sup>2</sup>C; an extra special overrun bit is set in this case.

The serial port of the controlling application has to be set to the baud rate selected via iLCD Manager XE (initially 115200 baud), 8-bit, no parity and 1, 1.5 or 2 stop bits. When using the RS-422/RS-485 mode (only available on Serial Port 1), 2 stop bits must be selected to allow the controller to switch the transmitter off in time.

---

## Controlling the iLCD via SPI

An extra application note has been setup to learn about how to communicate with an iLCD Controller via SPI.

This document can be downloaded from the demmel products web site: [SPI Application Note.pdf](#)

There is also a local *SPI Application Note.pdf* version available, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

---

## Controlling the iLCD via I<sup>2</sup>C

An extra application note has been setup to learn about how to communicate with an iLCD Controller via I<sup>2</sup>C.

This document can be downloaded from the demmel products web site: [I<sup>2</sup>C Application Note.pdf](#)

There is also a local *I<sup>2</sup>C Application Note.pdf* version available, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

---

## Controlling the iLCD via Ethernet

iLCDs with the DPC3080 controller and higher have the possibility to communicate via Ethernet with an additional Ethernet Interface-Board (DPA-TCPIP). iLCDs with DPC3050 or lower do not support Ethernet!

An extra application note has been setup to learn about how to communicate with an iLCD Controller via Ethernet.

This document can be downloaded from the demmel products web site: [Ethernet Application Note.pdf](#)

There is also a local *Ethernet Application Note.pdf* version available, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

---

## Sample Source Code

Two source code samples are available:

Sample one is a command line tool called iLoader to download raw flash data files created with iLCD Manager XE to the iLCD Controller. It is written in Standard C and shows how to generally send and receive data via the serial, USB and Ethernet port on the one hand and, on the other, can be integrated into an application to allow updating the flash data without having to use iLCD Manager XE. The sources compile under Borland's C++ Builder 5.0 and Linux but can easily be transferred to other operating systems and/or compilers. The sample code (including executables for Windows and Linux) can be found on the following location: <http://www.demmel.com/download/ilcd/iloader.zip>.

Sample two shows how to operate the iLCD's I<sup>2</sup>C port. This sample code contains a "driver" file, which covers all I<sup>2</sup>C functions described under [Controlling the iLCD via I<sup>2</sup>C](#). This file can be used without modification in most cases for your own application. The "main" file contains a very simple example to initialize the iLCD and to show text on the screen. Both files compile under Keil C for 8051 microcontroller at 12MHz but can be easily transferred to any other controller and/or compiler. The source code can be found at [http://www.demmel.com/download/ilcd/i2c\\_sample\\_code.zip](http://www.demmel.com/download/ilcd/i2c_sample_code.zip).

## Java

The integration of the *Java VM* allows to use the iLCD panel without external controller and makes it possible to take advantage of the computing power of the panel controller. See [Application Class Design](#) on how to implement a basic Java Application.

The environment for the implementation of a Java Application is fully integrated into the iLCD Manager XE. It provides all necessary features for editing, compiling and debugging a Java Program (see [Java IDE](#) for details), such as syntax highlighting, error highlighting, breakpoints, variable inspection, console I/O, call stack, source code outline, template insertion, code completion, global search.

Note that it is also possible to implement and debug a Java Application using the build in iLCD Simulator. Also check out the various Java samples via New - Browse for sample projects on the File Tab in the iLCD Manager XE.

Please note that the *Java* functionality is only available for panels with a *DPC3090* controller.

- [General Info](#)
- [Java IDE](#)

Java programs can be started from within normal iLCD macros using the following commands. This is particularly useful if the Java program shall be started automatically on power-up.

- [Set Java Main Method Arguments](#)
- [Start Java Binary](#)
- [Can Run Java](#)

## Packages

### Platform API

For programming the iLCD panel *Java 1.1* including the usual packages *java.lang*, *java.io* and *java.util* can be used. Note that these packages are proprietary and there are some differences in comparison with a standard Java implementation.

Almost the full iLCD command set is still available via the package *ilcd*.

Communication interfaces such as I2C, SPI or UART can be accessed via the package *comm*.

- [comm](#)
  - [hw](#)
  - [ilcd](#)
  - [java.io](#)
  - [java.lang](#)
  - [java.util](#)
  - [javax](#)
  - [javax.events](#)
-

## General Info

[Application Class Design](#) contains descriptions on how to implement a basic Java Application, with and without touch handling.

[ANSI Support](#) allows for cursor control, attribute settings, etc. using special ANSI sequences.

For information on how colors are represented see [Color Values](#).

To make use of MicroSD card related methods see [Class File](#).

## Overview

- [Application Class Design](#)
  - [ANSI Support](#)
  - [Color Values](#)
  - [Release Notes](#)
  - [MicroSD](#)
- 

## Application Class Design

In order to use Java on an iLCD panel it's recommended to implement at least the following two classes:

- The *Main Class* containing the *main()* method.
- The *Application Class* implementing the abstract method *OnUpdate()*.

Note that the *main()* method is the entry point of the program. The corresponding thread is called the main thread. There has to be exactly one class containing this method. If the *Java App* doesn't contain a *Main Class* yet, please mark the corresponding file in the *Java App* file tree via the right mouse click menu.

When enabling event handling (e.g. touch or keyboard events) an additional thread will be created to collect all occurring events. In this case it is required to extend the *Application Class* and call it's *run()* method. The *run()* method contains the central event loop of the Java program and makes sure that corresponding event listeners are called when an event occurs.

Find a detailed description how to design an application for the iLCD in the following chapters:

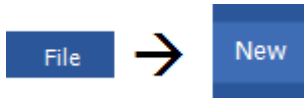
- [How To Create a Basic Java Application](#)
  - [How To Create A Java Application With Touch Handling](#)
-

## Application Class Design

### How To Create A Basic Java Application

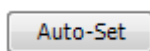
The following describes how to create a new Java application with automatically inserted class templates (also see [Application](#)).

#### Create a new empty project



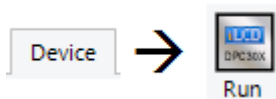
##### Create an empty project

Click on the File menu and select New in the iLCD Manager XE to open the configuration wizard.

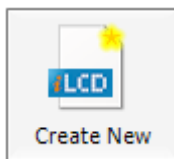


##### Select a panel

There are several ways to select a panel in the next step. When an iLCD panel is connected to the computer the Auto-Set button can be pressed to select the panel currently in use. In case there is no panel available select an iLCD panel or iLCD controller.



Furthermore, the iLCD Manager XE provides a simulator. Thus a Java project can be implemented without hardware. The Simulator is available on the "Device" ribbon. Simply click the *Run* button. You can also add this button to the *Quick Access Toolbar* in the upper left corner of the iLCD Manager XE.



##### Create the new empty project

The *Create New* button can be clicked after the desired panel is selected.

#### Add Java class files to the empty project

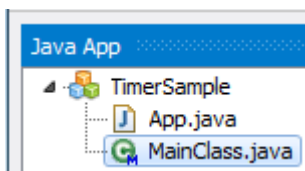


##### Add Java class files

To add a basic Java application to the iLCD project select *New* on the "Java" ribbon. Follow the dialogs to create two basic template classes, a *Main Class* and an *Application Class*. The files can be located anywhere, it's however recommended to keep all files in the same root folder.

Both java classes will be added to the newly created *Java App*. Notice that the *Main Class* has a special icon. It contains the main entry point of the Java application:

```
public static void main(String[] args)
```



There can only be one *Main Class* in an iLCD Java application. The Java application can already be build but want do anything interesting yet.

Add the following code to write simple text messages to the console every second.

**Add code to the App class:**

Put the following code in the application class, e.g. *App*.

```
private int countOnUpdateMethodCall;
```

Put the following code into the constructor *App()*.

```
try
{
    countOnUpdateMethodCall = 0;
    Console.println(" " + countOnUpdateMethodCall +
        ". onUpdate() call \r");
}
catch (Exception exc)
{
    Logger.e(exc.getMessage());
}
```

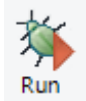
Put the following code into the *onUpdate()* method.

```
// count each call to the onUpdate() method
countOnUpdateMethodCall++;

try
{
    // draw the count of the onUpdate() calls to the screen
    Console.println(" " + countOnUpdateMethodCall +
        ". onUpdate() call \r");
}
catch (Exception exc)
{
    Logger.e(exc.getMessage());
}

// wait a second
try
{
    Thread.sleep(1000);
}
catch (InterruptedException ie)
{
    Logger.e(ie.getMessage());
}
```

**After coding is done**



### Run the program

Click the *Run* button. This will start the debugger. Notice that everything will be automatically downloaded to the iLCD panel. Feel free to add breakpoints to the *OnUpdate()* method. Now, every second a message is written to the screen. See [How To Create A Java Application With Touch Handling](#) on how to handle touch events.

## Application Class Design

### How To Create A Java Application With Touch Handling

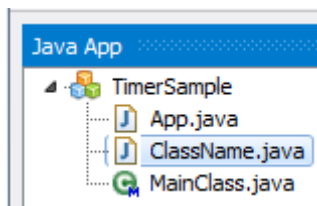
Create a new *Java App* without adding code, as explained in [How To Create A Basic Java Application](#).

#### Add a new java class to the project



#### Add a java file

Click on the *Add new* symbol in the iLCD Manager XE. Store the new file with the name *TouchButton.java* inside the iLCD project folder.



A new java class with the name *TouchButton.java* is added to the *Java App*, and will be displayed in the *Java App* tree view.

Add the following code to the project to create a simple touch button which prints a message on the screen.

#### Add code to the TouchButton class:

In order to handle touch events the class the *TouchButton* class has to implement the [OnTouchListener](#) interface. This is done by adding *implements OnTouchListener* to the class declaration.

```
public class TouchButton implements OnTouchListener
```

Import all classes from the *javax.events* and the *hw* package.

```
import javax.events.*;
import hw.*;
```

Add a counter variable to class *TouchButton* class.

```
private int countMethodCall;
```

Add a constructor to the *TouchButton* class.



```

public TouchButton() throws ILCDEException
{
    // draw a rectangle for a button on the screen
    Control.setCursorPosition(200, 30);
    Attribute.setBackgroundColor(0x00FF00);
    Draw.drawRectangle(
        Draw.RECTANGLE_FILLED_WITH_BACKGROUND_COLOR, 125, 50);

    // write a string on the button
    Control.setCursorPosition(210, 45);
    Draw.writeText("Push the button!");

    // create a touch field over the rectangle
    Control.setCursorPosition(200, 30);
    EventManagement.getTouchEventDispatcher().addListener(this);
    Touch.setTouchFieldHeight(50);
    Touch.setTouchFieldWidth(125);
    Touch.createDefineTouchField(1, 0);

    countMethodCall = 0;
}

```

Now the interface method `OnTouch()` has to be implemented.

```

public void onTouch(TouchEvent event)
{
    try
    {
        if (event.isTouchReleased())
        {
            // count each button press
            countMethodCall++;

            // write a string on the button
            Control.setCursorPosition(0, 45);
            Draw.writeText(" " + countMethodCall +
                ". onTouch() call \r");
        }
    }
    catch (ILCDEException exc)
    {
        Logger.e(exc.getMessage());
    }
}

```

**Add code to the constructor of the App class:**

```

try
{

```

```

General.resetAll();
Control.setCursorPosition(0, 45);
Draw.writeText("0. onTouch() call \r");

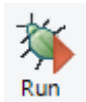
// enable touch field reporting
Touch.setTouchFieldReportingEnabled(true);

// create an instance of the button class
TouchButton button = new TouchButton();
}
catch (Exception exc)
{
    Logger.e(exc.getMessage());
}

```

Put the following code into the constructor `App()`. Note that touch field reporting has to be enabled with [setTouchFieldReportingEnabled](#).

### **After coding is done**



#### **Run the program**

Click the *Run* button after the coding is done. Now, the panel draws a first message and a button with another message to the panel. When the button on the panel is pressed the first message changes the text and the background color of this message. The first message changes the text every time when the button is pressed. Set a breakpoint in the `OnTouch()` method to verify the touch event handling.

## **ANSI Support**

All iLCD text messages can contain controlling ANSI sequences, for cursor control, attribute settings and so on. For further information about what sequences exist and how they are interpreted refer to the chapters listed under [Standard ANSI Sequences](#) and [Private ANSI Extensions](#). Please note that turning off ANSI mode causes the ANSI sequences to be interpreted as normal characters.

Using cursor control sequences from the standard ANSI sequences makes sense only if you have chosen a fixed pitch font as the iLCD controller always uses the maximum character width of the selected font (which is equal to the character width of any character on a fixed size font).

If you use a standard sequence such as `\u001B[4D` (go left 4 characters), and you have printed the text "Hill" via a proportional font, the resulting cursor position would be too far left as the "i" and "l" characters take much less space than the maximum character width used.

ANSI cursor control sequences will mainly be used on simple character output applications, which may use the iLCD's terminal mode. A typical application is showing the console output of a Linux system. In such cases there is no problem in using a fixed pitch font. More sophisticated applications can run the Java application with use of private ANSI extensions, which support many more possibilities.

Find following listings in this chapter:

- [Standard ANSI Sequences](#)

- [Private ANSI Extensions](#)

## ANSI Support

### Private ANSI Extensions

The iLCD's private ANSI extensions are not covered by an independent specification, so they will work on iLCD displays only.

Sequence	Description
\u001B { <x> ; <y> H	Go to <x> <y> <u>pixel</u> position <b>(1-based)</b>
\u001B { <x> ; <y> f	Go to <x> <y> <u>pixel</u> position <b>(1-based)</b>
\u001B { H	Go to home
\u001B { f	Go to home
\u001B { <n> F	Select font number <n> <b>(1-based, 0 = startup font)</b>
\u001B { <n> G	Paint graphic number <n> <b>(1-based, 0 = startup graphic)</b>
\u001B { <n> s	Save cursor & attributes to memory position <n> (0 ... 7)
\u001B { s	Save cursor & attributes to memory position 0
\u001B { <n> u	Restore cursor & attributes from memory position <n> (0 ... 7)
\u001B { u	Restore cursor & attributes from memory position 0

### Example

```
Draw.writeText ("\u001B{H" + "Go to home");
```

### See also:

[Standard ANSI Sequences](#)

## ANSI Support

### Standard ANSI Sequences

#### Control Characters

Sequence	Hex-Code	Description
\r	0x0D	Carriage Return (CR) - Return to beginning of line
\n	0x0A	Line Feed (LF) - Go to next line

\f	0x0C	Form Feed (FF) - Same effect as LF
\b	0x08	Backspace (BS) - Go back one character
\t	0x09	Horizontal Tab (HT) - Go to next tabulator position
\u001B	0x1B	Escape (ESC) - Start of ANSI sequence

### Example

```
Draw.writeText("Hello \t World!");
```

### Attribute Sequences

Sequence	Description
\u001B [ m	Set bold, underline and inverse mode off
\u001B [ 0 m	Set bold, underline and inverse mode off
\u001B [ 1 m	Set bold mode on
\u001B [ 4 m	Set underline mode on
\u001B [ 7 m	Set invert mode on

### Example

```
Draw.writeText("Only the expression " + "\u001B[4m" + "hello" + "\u001B[m" + " is underlined.");
```

### Display Control Sequences

Sequence	Description
\u001B [ 2 J	Clear Display
\u001B [ K	Erase to end of line
\u001B [ 0 K	Erase to end of line

### Example

```
Draw.writeText("\u001B[2J");
```

### Cursor Control Sequences

Sequence	Description
\u001B [ <n> A	Cursor up <n> rows
\u001B [ <n> B	Cursor down <n> rows
\u001B [ <n> C	Cursor right <n> columns
\u001B [ <n> D	Cursor left <n> columns

\u001B [ <x> ; <y> H	Go to <x> <y> position
\u001B [ <x> ; <y> f	Go to <x> <y> position
\u001B [ H	Go to home (set cursor to x=0 / y=0)
\u001B [ f	Go to home (set cursor to x=0 / y=0)

### Example

```
Draw.writeText ("\u001B[1A");
```

Please note, that <x> <y> is used in the opposite order to that in standard ANSI sequences.

### Save Restore Sequences

Sequence	Description
\u001B [ <n> s	Save cursor & attributes to memory position <n> (0 ... 7)
\u001B [ s	Save cursor & attributes to memory position 0
\u001B [ <n> u	Restore cursor & attributes from memory position <n> (0 ... 7)
\u001B [ u	Restore cursor & attributes from memory position 0

### Example

```
Draw.writeText ("\u001B[s");
```

### See also:

[Private ANSI Extensions](#)

## Color Values

- [16-Bit Color Values](#)
- [24-Bit Color Values](#)

## Color Values

### 16-Bit Color Values

The Color iLCD controller supports commands for setting colors such as background color or foreground color. Although the Color iLCD controller internally works with 16-bit color values, all commands except the read and write scan line commands use 24-bit color values (see at [24-Bit Color Values](#)) to allow future expansions. The [readScanLine\(int, short\[\]\)](#) and [writeScanLine\(int, short\[\]\)](#) scan line commands use the 16-bit color values only to minimize the amount of data to be read and

written, as one 320x240 color pixel screen needs  $320 \times 240 \times 2 = 153,600$  bytes to be read/written even in 16-bit color format.

In the [ilcd](#) package description, 16-bit color values are always described as type "color\_16bit" in the corresponding parameter tables.

The bit assignment of the 16-bit color values is R5G6B5, which make up one 16-bit word as follows:

*RRRR RGGG GGGB BBBB* (most significant bit is leftmost)

The pseudo formula to get a 16-bit color value from the red/green/blue parts is as follows:

$$\text{color\_16\_bit} = (\text{red} \ll 11) + (\text{green} \ll 5) + \text{blue}$$

where *red* and *blue* has a range of 0..31 and *green* has a range of 0..63, the maximum value for *red/green/blue* refers to 100% color intensity. Shifting *red* left by 11, shifting *green* left by 5 and adding the shifted red and green value and blue together delivers the 16-bit color value.

Some color values shown in binary and hex representation explain the color values further:

Color	Red (Dec. /%)	Green (Dec. /%)	Blue (Dec. /%)	Binary Value	Hex Value
Pure red	31 / 100%	0 / 0%	0 / 0%	1111100000000000	F800
Pure green	0 / 0%	63 / 100%	0 / 0%	0000011111100000	07E0
Pure blue	0 / 0%	0 / 0%	31 / 100%	0000000000011111	001F
Black	0 / 0%	0 / 0%	0 / 0%	0000000000000000	0000
White	31 / 100%	63 / 100%	31 / 100%	1111111111111111	FFFF
Yellow	31 / 100%	63 / 100%	0 / 0%	1111111111100000	FFE0
Magenta	31 / 100%	0 / 0%	31 / 100%	1111100000011111	F81F
Cyan	0 / 0%	63 / 100%	31 / 100%	0000011111111111	07FF

In the following description of these commands, the parameters of a 16-bit color value are always written as *xxx\_hb* and *xxx\_lb* where *xxx* describes the 16-bit color parameter.

This is the command for writing a (partial) scan line with noOfPixels length made up of pixels p1, p2, etc. An example showing the hex parameters for setting a red, a green and a blue pixel starting from the current cursor position is as follows:

#### Syntax in iLCD Manager XE - Java:

```
short red = 0x00;           //ranging from 0x00 to 0x1F
short green = 0x3F;        //ranging from 0x00 to 0x3F
short blue = 0x00;         //ranging from 0x00 to 0x1F
short color_16_bit = (short)((red << 11) + (green << 5) + blue);
```

## Color Values

### 24-Bit Color Values

The Color iLCD controller supports commands for setting colors such as background color or foreground color. Although the Color iLCD controller internally works with [16-Bit Color Values](#), all commands except the read and write scan line commands use 24 bit color values to allow future expansions.

In the [ilcd](#) package description, 24-bit color values are always described as type "color" in the corresponding parameter tables.

The 24-bit color value is converted to a 16-bit color value by the iLCD controller internally by using the following formula:

$$color\_16\_bit = ((red \gg 3) \ll 11) + ((green \gg 2) \ll 5) + (blue \gg 3)$$

where *red*, *green* and *blue* are 8-bit values with a range between 0 and 255.

The color values must be set in a 24-bit format, where *color\_r* describes the 8-bit red part, *color\_g* the green and *color\_b* the blue part of the 24-bit color value.

#### Syntax in iLCD Manager XE:

```
short red = 0x00;      //ranging from 0x00 to 0xFF
short green = 0xFF;   //ranging from 0x00 to 0xFF
short blue = 0x00;    //ranging from 0x00 to 0xFF
int color_24_bit = (red << 16) + (green << 8) + blue;
```

## Release Notes

Important information about JavaLib changes corresponding to different iLCD Manager XE versions.

- [iLCD Manager XE 4.0.7.0](#)
- [iLCD Manager XE 4.0.9.0](#)

### iLCD Manager XE 4.0.7.0 Release Note

With this version the Java API has been changed. This means that Java Apps created with an older version have to be modified and recompiled.

Please apply the following changes to existing projects:

☐ Class `ilcd.EventManager` has been moved to [javax.events.EventManagement](#) and made static:

- Rename *Common.EventManagement* to *EventManagement* in java source files using this class.
- Add `import javax.events.*` to java source files using this class.

☐ Class *ilcd.Application* has been moved to [javax.Application](#):

- Add `import javax.*` to java source files using this class.
- Method [Application.run\(\)](#) throws [MainThreadException](#) now. Add according catch clause where `run()` is called.

☐ The following classes have been moved from package *ilcd* to package [hw](#).  
Add `import javax.events.*` to java source files using this classes:

- *ilcd.IO* moved to *hw.IO*
- *ilcd.Date* moved to *hw.Date*
- *ilcd.DateTime* moved to *hw.DateTime*
- *ilcd.ILCDIOException* moved to *hw.IOException*
- *ilcd.Time* moved to *hw.Time*
- *ilcd.Logger* moved to *hw.Logger*

In addition to the above changes the new method [writeApplicationDataToFlash\(\)](#) has been added to class *ilcd.General*.

---

## Package comm

Contains classes for providing access to basic communication interfaces such as UART, I2C, SPI.

### Note

- See the Chapter Pin Descriptions in [ILCD Panels Specification.pdf](#) to find out which pins to use for the different interfaces.

## Event Handling

The [EventManager](#) allows for listening to Comm events i.e. being automatically notified when data is received in slave mode. To do that a `OnCommListener` has to be implemented by extending on of the three classes:

- [OnI2CListener](#)
- [OnSPIListener](#)
- [OnUARTListener](#)
- [OnUSBListener](#)
- [OnEthernetListener](#)

## Classes

- [Comm](#)
- [CommEventDispatcher](#)
- [CommStateException](#)
- [Ethernet](#)



- [EthernetEventDispatcher](#)
  - [I2C](#)
  - [I2CEventDispatcher](#)
  - [I2CException](#)
  - [InterfaceNotOpenException](#)
  - [OnCommListener](#)
  - [OnEthernetListener](#)
  - [OnI2CListener](#)
  - [OnSPIListener](#)
  - [OnUARTListener](#)
  - [OnUSBListener](#)
  - [SPI](#)
  - [SPIEventDispatcher](#)
  - [TimeoutException](#)
  - [UART](#)
  - [UARTEventDispatcher](#)
  - [USB](#)
  - [USBEventDispatcher](#)
  - [WrongCommModeException](#)
- 

## Package [comm](#)

### Class **Comm**

extends [Object](#)

This class implements a basic communication interface. Derived classes are e.g. [USB](#) and [Ethernet](#).

#### Public Methods

- [close](#)
- [getReceiveDataSize](#)
- [getTransmitDataSize](#)
- [open](#)
- [read](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
-

Package [comm](#)  
Class [Comm](#)

**Public Method close**

```
void close ()
```

**Description:**

Closes this communication interface (e.g. [USB](#) or [Ethernet](#)). Any pending data will be discarded.

---

Package [comm](#)  
Class [Comm](#)

**Public Method getReceiveDataSize**

```
int getReceiveDataSize ()
```

**Description:**

Returns the number of bytes received, i.e. the number of bytes that can be read by calling [read\(\)](#).

---

Package [comm](#)  
Class [Comm](#)

**Public Method getTransmitDataSize**

```
int getTransmitDataSize ()
```

**Description:**

Returns the number of bytes that can be transmitted, i.e. the number of bytes that can be send by calling [write\(\)](#).

---

Package [comm](#)  
Class [Comm](#)

**Public Method open**

```
boolean open ()
```

**Description:**

Opens this communication interface (e.g. [USB](#) or [Ethernet](#) ).

---

Package [comm](#)  
Class [Comm](#)

**Public Method read**

```
int read()
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

```
int read(byte[] buf)
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

```
int read(byte[] buf, int offset, int len)
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

**Description:**

**read() method:**

Returns	Description
value	a single byte

The `read()` method returns one single byte or -1 if nothing was read.

**read(byte[] buf) method:**

Parameter	Description
buf	array storing the returned data

Reads *buf.length* bytes into the array *buf*. The number of bytes read will be returned.

Returns	Description
value	number of returned bytes

**read(byte[] buf, int offset, int len) method:**

Parameter	Description
<i>buf</i>	array storing the returned data
<i>offset</i>	array offset
<i>len</i>	count of bytes

Reads *len* bytes into the array *buf* starting at *offset*. The number of bytes read will be returned.

Returns	Description
value	number of returned bytes

Package [comm](#)

Class [Comm](#)

**Public Method [write](#)**

```
int write(byte value)
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

```
int write(byte[] buf)
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

```
int write(byte[] buf, int offset, int len)
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

```
int write(String value)
```

**Throws:**

- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)
- [InterfaceNotOpenException](#)
- [CommStateException](#)

**Description:****write(byte value) method:**

Parameter	Description
value	byte that should be sent

Transmits a single byte.

**write(byte[] source, int offs, int count) method:**

Parameter	Description
source	byte array to be sent
offs	offset to the first data byte
count	number of the data byte

Transmits *count* bytes of the array *source*. The *offs* paramter specifies the offset to the first byte in the *source* byte array.

**write(String source) method:**

Parameter	Description
source	String to be sent

Transmits the string *source*.

**Package [comm](#)****Class CommEventDispatcher**

extends [AsyncEvent](#) → [Object](#)

Base class for event dispatcher classes such as:

- [I2CEventDispatcher](#)
- [SPIEventDispatcher](#)
- [UARTEventDispatcher](#)
- [USBEventDispatcher](#)
- [EthernetEventDispatcher](#)

Normally it's not necessary to instantiate this class, it should be handled by the [EventManagement](#) system.

### Public Constructors

- [CommEventDispatcher](#)

### Public Methods

- [addListener](#)
- [getListenerCount](#)
- [removeAllListeners](#)
- [removeListener](#)
- [startEventHandling](#)
- [stopAllCommEvents](#)
- [stopEventHandling](#)

### Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final int START\_EVENT\_HANDLING
- static final int STOP\_EVENT\_HANDLING

Package [comm](#)

## Class [CommEventDispatcher](#)

### **Public Constructor CommEventDispatcher**

```
CommEventDispatcher(int[] EventTypes)
```

#### **Description:**

Normally it's not necessary to instantiate this class, it should be handled by the [EventManager](#) system.

---

Package [comm](#)

## Class [CommEventDispatcher](#)

### **Public Method addListener**

```
void addListener(OnCommListener listener)
```

#### **Description:**

Add a new listener to this event dispatcher. Listeners will be notified when the corresponding event occurs.

---

Package [comm](#)

## Class [CommEventDispatcher](#)

### **Public Method getListenerCount**

```
int getListenerCount()
```

#### **Description:**

Returns the number of listeners registered with this event dispatcher.

---

Package [comm](#)

## Class [CommEventDispatcher](#)

### **Public Method removeAllListeners**

```
void removeAllListeners()
```

**Description:**

Remove all registered listeners from this event dispatcher. The listeners will not be notified any more about occurring events.

---

Package [comm](#)

**Class [CommEventDispatcher](#)****Public Method [removeListener](#)**

```
void removeListener(OnCommListener listener)
```

**Description:**

Remove the specified listener from this event dispatcher. The listener will not be notified any more about occurring events.

---

Package [comm](#)

**Class [CommEventDispatcher](#)****Public Method [startEventHandling](#)**

```
void startEventHandling()
```

**Description:**

Start event handling for the comm interface of this CommEventDispatcher, e.g. UART for the [UARTEventDispatcher](#). Typically used after [stopEventHandling](#)

---

Package [comm](#)

**Class [CommEventDispatcher](#)****Public Method [stopEventHandling](#)**

```
void stopEventHandling()
```

**Description:**

Stops event handling for the comm interface of this CommEventDispatcher, e.g. UART for the [UARTEventDispatcher](#).

---



Package [comm](#)

## Class [CommEventDispatcher](#)

### Public Method [stopAllCommEvents](#)

```
static void stopAllCommEvents ()
```

#### Description:

Stops event handling for all comm interfaces (e.g. I2C, UART, SPI).

---

Package [comm](#)

## Class [CommStateException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

Exception class to indicate an incorrect communication interface state, such as buffer overflows, frame errors etc.

### Public Constructors

- [CommStateException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

Package [comm](#)

## Class [CommStateException](#)

### Public Constructor [CommStateException](#)

```
CommStateException ()
```

```
CommStateException (String s)
```

**Description:****constructor CommStateException():**

Constructs a new CommStateException class without additional information.

**constructor CommStateException(String s):**

Constructs a new CommStateException class with additional information.

---

**Package [comm](#)****Class Ethernet**

extends [Comm](#) → [Object](#)

The iLCD panel can receive data via Ethernet as TCP client. In order to communicate with the panel a TCP socket with the IP address of the panel and the port number 54131 has to be created. After connecting to the panel it's possible to send and receive data.

See [OnEthernetListener](#) on how to receive data via the [EventManager](#) system.

**Note**

- TCP/IP has to be enabled in the iLCD project settings (Settings Tab in the iLCD Manager XE).

**Public Methods**

- [getInstance](#)

**Methods inherited from [comm.Comm](#)**

- [open](#)
- [close](#)
- [read](#)
- [write](#)
- [getReceiveDataSize](#)
- [getTransmitDataSize](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
-

**Package [comm](#)****Class [Ethernet](#)****Public Method [getInstance](#)**

```
static Ethernet getInstance ()
```

**Description:**

Returns the only instance of this class.

**Example**

Typical usage:

```
Ethernet.getInstance ().open ();
```

**Package [comm](#)****Class [EthernetEventDispatcher](#)**

extends [CommEventDispatcher](#) → [AsyncEvent](#) → [Object](#)

This class is normally used internally by the [EventManagement](#) system. Add an [OnEthernetListener](#) object to the *EthernetEventDispatcher* of the *EventManagement* to be notified when data is received.

**Example**

The class *EthernetListener* extends [OnEthernetListener](#) and implements the method [onReceive](#).

```
EthernetListener ethernetListener = new EthernetListener ();  
EventManagemen  
t.getEthernetEventDispatcher ().addListener (ethernetListener);
```

**Note**

- TCP/IP has to be enabled in the iLCD project settings (Settings Tab in the iLCD Manager XE).

**Public Constructors**

- [EthernetEventDispatcher](#)

**Public Methods**

- [addListener](#)
- [removeListener](#)

**Methods inherited from [comm.CommEventDispatcher](#)**

- [addListener](#)
- [removeListener](#)

- [removeAllListeners](#)
- [getListenerCount](#)
- [startEventHandling](#)
- [stopEventHandling](#)
- [stopAllCommEvents](#)

#### Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

#### Package [comm](#)

### Class [EthernetEventDispatcher](#)

#### Public Constructor EthernetEventDispatcher

```
EthernetEventDispatcher ()
```

#### Description:

Normally it should not be necessary to create an object of this class, it's handled internally by the [EventManagement](#) system.

---

#### Package [comm](#)

### Class [EthernetEventDispatcher](#)

#### Public Method addListener

```
void addListener (OnEthernetListener listener)
```

#### Description:

Add a new *OnEthernetListener* to this event dispatcher. Listeners will be notified when the corresponding event occurs.

---

Package [comm](#)

## Class [EthernetEventDispatcher](#)

### Public Method [removeListener](#)

```
void removeListener(OnEthernetListener listener)
```

#### Description:

Removes the specified *OnEthernetListener* from this event dispatcher. It will not receive any further event notifications.

---

Package [comm](#)

## Class [I2C](#)

extends [Object](#)

In a typical master-slave I2C configuration one single master can exchange data with multiple slave devices. The iLCD controller can either be used as I2C master or as I2C slave device. The I2C class is necessary for both of these modes. In master mode it allows to control the I2C bus exclusively, in slave mode it is needed for reading and writing data from and to the master device. Note that in order to use the iLCD controller as slave device you have to use the [EventManagement](#) system and add a [OnI2CListener](#) to the [I2CEventDispatcher](#).

#### Notes

- I2C has to be enabled in the iLCD project. To do this select the corresponding checkbox in the iLCD Manager XE on the Settings Tab under Hardware Settings.
- An object can be created for every I2C device.
- The iLCD's I2C communication follows the rules of the Philips documents [I<sup>2</sup>C bus specification.pdf](#) which describes the basics of the I2C bus.
- See the Chapter Pin Descriptions in [ILCD Panels Specification.pdf](#) to find out which pins to use for *SDA* and *SCL*.
- The iLCD controller can be configured in fast-mode with up to 400 kHz clock frequency.
- A 7-bit address space is available to add more than 100 slave devices to the I2C bus.

#### Public Constructors

- [I2C](#)

#### Public Methods

- [close](#)
- [getCurrentState](#)
- [gotoNextState](#)
- [open](#)
- [poll](#)
- [read](#)

- [reOpen](#)
- [reStart](#)
- [setTimeout](#)
- [start](#)
- [stop](#)
- [write](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final int DEFAULT\_MASTER\_ADDRESS
- static final int DEFAULT\_TIMEOUT
- static final int OK
- static final int ERROR
- static final int TIMEOUT
- static final int START\_CONDITION\_TRANSMITTED
- static final int REPEATED\_START\_CONDITION\_TRANSMITTED
- static final int SLAVE\_RECEIVER\_ADDRESSED\_ACK\_RECEIVED
- static final int SLAVE\_RECEIVER\_ADDRESSED\_NACK\_RECEIVED
- static final int BYTE\_TRANSMITTED\_AS\_MASTER\_ACK\_RECEIVED
- static final int BYTE\_TRANSMITTED\_AS\_MASTER\_NACK\_RECEIVED
- static final int ARBITRATION\_LOST
- static final int SLAVE\_TRANSMITTER\_ADDRESSED\_ACK\_RECEIVED
- static final int SLAVE\_TRANSMITTER\_ADDRESSED\_NACK\_RECEIVED
- static final int BYTE\_RECEIVED\_AS\_MASTER\_ACK\_TRANSMITTED
- static final int BYTE\_RECEIVED\_AS\_MASTER\_NACK\_TRANSMITTED
- static final int ADDRESSED\_AS\_SLAVE\_RECEIVER\_ACK\_TRANSMITTED
- static final int ADDRESSED\_AS\_SLAVE\_RECEIVER\_NACK\_TRANSMITTED
- static final int DATA\_RECEIVED\_AS\_SLAVE\_ACK\_TRANSMITTED
- static final int DATA\_RECEIVED\_AS\_SLAVE\_NACK\_TRANSMITTED
- static final int STOP\_OR\_REPEATED\_START\_RECEIVED\_AS\_SLAVE
- static final int ADDRESSED\_AS\_SLAVE\_TRANSMITTER\_ACK\_TRANSMITTED
- static final int  
ARBITRATION\_LOST\_ADDRESSED\_AS\_SLAVE\_TRANSMITTER\_ACK\_TRANSMITTED
- static final int BYTE\_TRANSMITTED\_AS\_SLAVE\_ACK\_RECEIVED
- static final int BYTE\_TRANSMITTED\_AS\_SLAVE\_NACK\_RECEIVED
- static final int NO\_RELEVANT\_STATUS\_INFORMATION

---

### Package [comm](#)

### Class [I2C](#)

### Public Constructor I2C

```
I2C(int address)
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

```
I2C(int address, boolean slave_mode)
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

```
I2C(int address, int frequency)
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

```
I2C(int address, int frequency, boolean slave_mode)
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

**Description:**

The iLCD panel might be used as I2C-master or I2C-slave device. An instance of this class represents either an external slave device with the iLCD-controller automatically set as master or the iLCD-controller as slave device used with an external master.

**constructor I2C(int addr):**

**constructor I2C(int addr, boolean slave\_mode):**

Parameter	Range	Description
addr	0 ... 127	target slave device address
[slave_mode]	true/false	I2C-mode of iLCD controller (default: false)

If *slave\_mode* is omitted or false, the *addr* parameter is used to set up the address for a specific I2C slave device to be addressed by the iLCD-controller in master mode.

If *slave\_mode* is true, the iLCD-controller will be used in slave mode controlled by an external master via the address *addr*.

**Note**

- *slave\_mode* = false: The iLCD-controller has address 0 and is used as I2C-master (a different master-address can be set with [open\(\)](#)).
- *slave\_mode* = true: The iLCD-controller has the specified address and is used as I2C-slave.
- A default frequency of 100kHz is automatically assigned.

- Some addresses are used for general purpose of the I2C bus (see [I<sup>2</sup>C bus specification.pdf](#)).

**constructor I2C(int addr, int freq):**

**constructor I2C(int addr, int freq, boolean slave\_mode):**

Parameter	Range	Description
addr	0 ... 127	target slave device address
freq	0 ... 400000	I2C bus frequency in Hz
[slave_mode]	true/false	I2C-mode of iLCD controller (default: false)

This constructor is used to adjust the frequency for the I2C-bus.

If *slave\_mode* is omitted or false, the *addr* parameter is used to set up the address for a specific I2C slave device to be addressed by the iLCD-controller in master mode.

If *slave\_mode* is true, the iLCD-controller will be used in slave mode controlled by an external master via the address *addr*.

### Notes

- I2C has to be enabled in the iLCD project. To do this select the corresponding checkbox in the iLCD Manager XE on the Settings Tab under Hardware Settings.
- *slave\_mode* = false: The iLCD-controller has address 0 and is used as I2C-master(a different master-address can be set with [open\(\)](#)).
- *slave\_mode* = true: The iLCD-controller has the specified address and is used as I2C-slave.
- Some addresses are used for general purpose of the I2C bus (see [I<sup>2</sup>C bus specification.pdf](#)).

### Example

The following code shows how to create an instance of the I2C class.

```
import comm.I2C;
...
I2C i2c;
...
i2c = new I2C(0x40);
```

The *comm* package must be imported in order to use the I2C class. A new variable with the I2C data type must be created in the next step. Afterwards, a new instance of the I2C class is created with an automatically assigned frequency of 100kHz.

### See also:

[SPI](#)  
[UART](#)



Package [comm](#)  
Class [I2C](#)

**Public Method `close`**

```
static void close()
```

**Description:**

Disable the I2C-interface of the iLCD-controller.

---

Package [comm](#)  
Class [I2C](#)

**Public Method `getCurrentState`**

```
static int getCurrentState()
```

**Description:**

This method is supported by DPM3090 controller only. Other controllers return 0x00.

Return s	Description
state	state of the I2C bus

Returns the current state of the I2C bus. See also the corresponding fields defined in class [I2C](#).

state	Description
0x00	Bus error
0x08	START condition transmitted
0x10	Repeated START condition transmitted
0x18	As master: slave receiver addressed (SLA + W), ACK received
0x20	As master: slave receiver addressed (SLA + W), NACK received
0x28	Data byte transmitted, ACK received
0x30	Data byte transmitted, NACK received
0x38	Arbitration lost
0x40	As master: slave transmitter addressed (SLA + R), ACK received
0x48	As master: slave transmitter addressed (SLA + R), NACK received
0x50	Data byte received as master, ACK transmitted
0x58	Data byte received as master, NACK transmitted
0x60	Addressed as slave receiver (SLA + W), ACK transmitted

0x68	Arbitration lost, addressed as slave receiver (SLA + W), ACK transmitted
0x70	General call address received, ACK transmitted
0x78	Arbitration lost, general call addr received, ACK transmitted
0x80	Data byte received as slave (SLA + W), ACK transmitted
0x88	Data byte received as slave (SLA + W), NACK transmitted
0x90	Data byte received after general call, ACK transmitted
0x98	Data byte received after general call, NACK transmitted
0xA0	STOP or repeated START condition received in slave mode
0xA8	Addressed as slave transmitter (SLA + R), ACK transmitted
0xB0	Arbitration lost, Addressed as slave transmitter (SLA + R), ACK transmitted
0xB8	Data byte transmitted as slave (SLA + R), ACK received
0xC0	Data byte transmitted as slave (SLA + R), NACK received
0xF8	No relevant status information
0xFD	Timeout waiting for status change
0xFF	Channel error

### Example

The following example reads the current state of the I2C bus.

```
int state = I2C.getCurrentState();
```

### See also:

[SPI](#)  
[UART](#)

## Package [comm](#)

### Class [I2C](#)

#### **Public Method gotoNextState**

```
static int gotoNextState()
```

#### **Description:**

This method is supported by DPM3090 controller only. Other controllers return 0x00.

Returns	Description
I2C-bus state	returns new I2C-bus state

Stops clock stretching and switches the I2C-bus to the next state. The returned value is the same as returned by see [getCurrentState\(\)](#).

**See also:**

[SPI](#)  
[UART](#)

---

**Package [comm](#)**

**Class [I2C](#)**

**Public Method [open](#)**

```
void open(int freq, int address)
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

**Description:**

Parameter	Range	Description
addr	0 ... 127	iLCD controller address
freq	0 ... 400000	I2C bus frequency in Hz

Opens the I2C interface. This method can be used to set the frequency and the address of the iLCD-controller.

**Notes**

- I2C has to be enabled in the iLCD project. To do this select the corresponding checkbox in the iLCD Manager XE on the Settings Tab under Hardware Settings.
- 

**Package [comm](#)**

**Class [I2C](#)**

**Public Method [poll](#)**

```
boolean poll()
```

```
boolean poll(int addr)
```

**Description:****poll() method:**

Returns	Range	Description
acknowledge	true ... false	is ACK received or not

The `poll()` method waits until an acknowledge (ACK) is received from a slave device after a busy period.

**Note**

- This method doesn't allow to wait for multiple slave devices.

**poll(int addr) method:**

Parameter	Range	Description
addr	0 ... 127	target device address

The `addr` parameter is used to set up the address of a specific I2C slave device. The `poll(int)` method waits until an acknowledge (ACK) is received from the slave device after a busy period.

Returns	Range	Description
acknowledge	true ... false	is ACK received or not

**Note**

- This method allows to wait for an ACK of another slave device which is specified by the `addr` parameter.

**Example**

```
i2c.poll();
```

The `poll()` method waits until an ACK from the slave is received.

**See also:**

[SPI](#)  
[UART](#)

Package [comm](#)

Class [I2C](#)

**Public Method read**

```
int read()
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
int read(boolean sendNACK)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
int read(byte[] buf)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
int read(byte[] buf, boolean sendNACK)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
int read(byte[] buf, int offset, int len)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
int read(byte[] buf, int offset, int len, boolean sendNACK)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

**Description:**

The read method reads data from an I2C slave device when the iLCD-controller is in master mode or from an I2C master device when the iLCD-controller is in slave mode. Slave mode can be set via the [I2C](#)-constructor.

Parameter	Description
<code>dest</code>	array storing the returned data
<code>offs</code>	offset of bytes
<code>len</code>	count of bytes
<code>sendNACK</code>	Send NACK after len bytes in slave mode

**int read() method:**

Returns	Description
<code>data</code>	returns one byte as int or -1 if no data was read

**int read(boolean sendNACK) method:**

Returns	Description
<code>data</code>	returns one byte as int or -1 if no data was read

When `sendNACK` is true in slave mode, NACK will be returned instead of ACK after the last byte received (default value is false). Note that the `sendNACK` parameter will be ignored in master mode (obligatory NACK after last received byte).

**int read(byte[] buf) method:**

Returns	Description
<code>nr_of_bytes</code>	returns number of bytes successfully received

The parameter `buf` is used to store the received data bytes. The number of data bytes to be received is specified by `buf.length`. Note that the number of actually received bytes might be smaller than `len`.

**int read(byte[] buf, boolean sendNACK) method:**

Returns	Description
<code>nr_of_bytes</code>	returns number of bytes successfully received

The first parameter `buf` is used to store the received data bytes. The number of data bytes to be received is specified by `buf.length`. Note that the number of actually received bytes might be smaller than `len`. When `sendNACK` is true in slave mode, NACK will be returned instead of ACK after

the last byte received (default value is false). Note that the `sendNACK` parameter will be ignored in master mode (obligatory NACK after last received byte).

**int read(byte[] buf, int offset, int len) method:**

Returns	Description
<code>nr_of_bytes</code>	returns number of bytes successfully received

The first parameter `buf` is used to store the received data bytes. The number of data bytes to be received is specified by the `len` parameter. Note that the number of actually received bytes might be smaller than `len`. The `offset` paramter specifies the offest in the `buf` array.

**int read(byte[] buf, int offset, int len, boolean sendNACK) method:**

Returns	Description
<code>nr_of_bytes</code>	returns number of bytes successfully received

The first parameter `buf` is used to store the received data bytes. The number of data bytes to be received is specified by the `len` parameter. Note that the number of actually received bytes might be smaller than `len`. The `offset` paramter specifies the offest in the `buf` array. When `sendNACK` is true in slave mode, NACK will be returned instead of ACK after the last byte received (default value is false). Note that the `sendNACK` parameter will be ignored in master mode (obligatory NACK after last received byte).

### Note

- In master mode the `start()` or the `reStart()` and one `write` method must be called before a `read` method is executed. The `write` method is required to address a specific slave device to comply with the I2C specification.
- In master mode the `stop()` method must be called after the last read transmission is completed.

### Example

A typical sequence for reading a slave device might look like this:

```
byte[] data = new byte[2];

try
{
    i2c.start();
    write((byte) SI7021_MEASRH_NOHOLD_CMD);

    i2c.reStart();
    delay(25);

    ret = i2c.read(data);
    i2c.stop();
}
catch (Exception exc)
{
```

```
Console.println(exc.getMessage());  
}
```

**See also:**

[SPI](#)  
[UART](#)

---

Package [comm](#)

Class [I2C](#)

**Public Method reOpen**

```
void reOpen()
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

**Description:**

Re-initialize the I2C-bus with the frequency- and address-values set in the I2C-constructor of this I2C-object.

---

Package [comm](#)

Class [I2C](#)

**Public Method reStart**

```
int reStart()
```

**Description:**

Send a repeated *START* condition for allowing combined *write/read* operations to one or more devices without releasing the bus.

---

Package [comm](#)

Class [I2C](#)

**Public Method setTimeout**

```
static void setTimeout(int ms)
```



**Throws:**

- [IllegalArgumentException](#)

**Description:**

Parameter	Description
ms	value for the timeout in milliseconds

This method is used to set up the time value for timeout. The timeout is used in *start*, *reStart*, *write*, *read*, *poll*, and *status* methods to provide a defined time frame for data processing. Connected devices should return messages within the time frame, otherwise an *error code* is returned. The default value for the timeout is 50 milliseconds and is automatically set by the constructor.

**Example**

The following example sets up a timeout of 100 milliseconds.

```
i2c.timeout(100);
```

**See also:**

[SPI](#)  
[UART](#)

Package [comm](#)

Class [I2C](#)

**Public Method start**

```
int start()
```

**Description:**

Returns	Description
error	returns 1 if start condition was successfully sent, otherwise an error has occurred

The *start()* method generates a start condition on the I2C bus. It returns 1 if the start was successful. Otherwise an error code is returned (0 for bus error or -3 for timeout).

**Example**

The following example excludes the *write*, *restart()*, and *stop()* method calls which are required for an I2C communication.

```
i2c.start();
```

**See also:**

[SPI](#)  
[UART](#)

---

**Package [comm](#)****Class [I2C](#)****Public Method stop**

```
void stop()
```

**Description:**

The `stop()` method is used to complete a transmission between the master and the slave. This method must be used after every transmission to comply with the I2C bus specification.

**Example**

The following example excludes the `start()`, `write`, `reStart()`, and `read` method calls which are required for an I2C communication.

```
i2c.stop();
```

**See also:**

[SPI](#)  
[UART](#)

---

**Package [comm](#)****Class [I2C](#)****Public Method write**

```
void write(byte value)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
void write(byte[] buf)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
void write(byte[] buf, byte register)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
void write(byte[] buf, byte register, int offset, int len)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
void write(byte[] buf, int offset, int len)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
void write(byte[] buf, short register)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

```
void write(byte[] buf, short register, int offset, int len)
```

**Throws:**

- [I2CException](#)
- [TimeoutException](#)
- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

**Description:****write(byte value) method:**

Parameter	Description
<i>value</i>	data byte that should be sent

The parameter *value* is a single byte which should be transmitted to the slave device.

**write(byte[] buf) method:**

Parameter	Description
<i>buf</i>	data bytes that should be sent

The parameter *buf* represents an array of bytes which should be transmitted to a specific slave device.

**write(byte[] buf, byte register) method:**

Parameter	Description
<i>buf</i>	data bytes that should be sent
<i>register</i>	an extra byte is sent to the slave

The parameter *buf* represents an array of bytes which should be transmitted to a specific slave device. Additionally, one extra byte is transmitted directly after the address byte. This is typically used to specify a slave device register where *buf* is to be written to.

**write(byte[] buf, byte register, int offset, int len) method:**

Parameter	Description
<i>buf</i>	data bytes that should be sent
<i>register</i>	an extra byte is sent to the slave
<i>offset</i>	offset in buf array
<i>len</i>	number of the data bytes to be written

The parameter *buf* represents an array of bytes which should be transmitted to a specific slave device. Additionally, one extra byte is transmitted directly after the address byte. This is typically used to specify a slave device register where *buf* is to be written to. The number of data bytes must be specified by the *len* parameter. The *offset* parameter specifies the offset in the *buf* array.

**write(byte[] buf, int offset, int len) method:**

Parameter	Description
-----------	-------------

<code>buf</code>	data bytes that should be sent
<code>offset</code>	offset in <code>buf</code> array
<code>len</code>	number of the data byte

The parameter `buf` represents an array of bytes which should be transmitted to a specific slave device. The number of data bytes must be specified by the `len` parameter. The `offset` parameter specifies the offset in the `buf` array.

#### **write(byte[] buf, short register) method:**

Parameter	Description
<code>buf</code>	data bytes that should be sent
<code>register</code>	two extra bytes are sent to the slave

The parameter `buf` represents an array of bytes which should be transmitted to a specific slave device. Additionally, two extra bytes are transmitted directly after the address byte. This is typically used to specify a slave device register where `buf` is to be written to.

#### **write(byte[] buf, short register, int offset, int len) method:**

Parameter	Description
<code>buf</code>	data bytes that should be sent
<code>register</code>	two extra bytes are sent to the slave
<code>offset</code>	offset to the first data byte
<code>len</code>	number of the data byte

The parameter `buf` represents an array of bytes which should be transmitted to a specific slave device. Additionally, two extra bytes are transmitted directly after the address byte. This is typically used to specify a slave device register where `buf` is to be written to. The number of data bytes must be specified by the `len` parameter. The `offset` parameter specifies the offset in the `buf` array.

#### **Note**

- In master mode the `start()` method must be called before any write method is executed.
- In master mode the `stop()` method must be called after a write transmission is completed.

#### **Example**

The following example excludes the `start()` and `stop()` method calls which are required for an I2C communication.

```
byte[] data = {0x30, 0x40};
byte register = SI7021_READRHT_REG_CMD;
```

```
try
{
    i2c.start();
    i2c.write(data, register, offset, length);
    i2c.stop();
}
catch (Exception exc)
{
    Console.WriteLine(exc.getMessage());
}
```

The code above writes one extra byte to a specific slave device register after the address byte is sent. Afterwards two bytes are written to the same slave.

#### See also:

[SPI](#)  
[UART](#)

---

## Package [comm](#)

### Class [I2CEventDispatcher](#)

extends [CommEventDispatcher](#) → [AsyncEvent](#) → [Object](#)

This class is normally used internally by the [EventManagement](#) system. Add an [OnI2CListener](#) object to the *I2CEventDispatcher* of the *EventManagement* to be notified when I2C data is received.

#### Example

```
I2CListener i2cListener = new I2CListener(8);
EventManagement.getI2CEventDispatcher().addListener(i2cListener);
```

#### Public Constructors

- [I2CEventDispatcher](#)

#### Public Methods

- [addListener](#)
- [removeListener](#)

#### Methods inherited from [comm.CommEventDispatcher](#)

- [addListener](#)
- [removeListener](#)
- [removeAllListeners](#)
- [getListenerCount](#)
- [startEventHandling](#)
- [stopEventHandling](#)
- [stopAllCommEvents](#)

**Methods inherited from [javax.events.AsyncEvent](#)**

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

Package [comm](#)

**Class [I2CEventDispatcher](#)****Public Constructor I2CEventDispatcher**

```
I2CEventDispatcher ()
```

**Description:**

Normally it should not be necessary to create an object of this class, it's handled internally by the [EventManagement](#) system.

---

Package [comm](#)

**Class [I2CEventDispatcher](#)****Public Method addListener**

```
void addListener (OnI2CListener listener)
```

**Description:**

Add a new *OnI2CListener* to this event dispatcher. Listeners will be notified when the corresponding event occurs.

---

Package [comm](#)

## Class [I2CEventDispatcher](#)

### Public Method [removeListener](#)

```
void removeListener(OnI2CListener listener)
```

#### Description:

Remove *OnI2CListener* from this event dispatcher. The listener will not be notified any more about occurring events.

---

Package [comm](#)

## Class [I2CException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

An *I2CException* is thrown when an error occurs on the I2C-Bus.

### Public Constructors

- [I2CException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

Package [comm](#)

## Class [I2CException](#)

### Public Constructor [I2CException](#)

```
I2CException()
```



**I2CException** (String s)

**Description:**

**constructor I2CException():**

Constructs a new *I2CException* class without additional information.

**constructor I2CException(String s):**

Constructs a new *I2CException* class with additional information.

---

**Package** [comm](#)

**Class** [InterfaceNotOpenException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

This Exception is thrown when a comm interface method is called without the interface being opened before.

**Public Constructors**

- [InterfaceNotOpenException](#)

**Methods inherited from** [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from** [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

**Package** [comm](#)

**Class** [InterfaceNotOpenException](#)

**Public Constructor** [InterfaceNotOpenException](#)

**InterfaceNotOpenException** ()

**InterfaceNotOpenException** (String s)**Description:****constructor InterfaceNotOpenException():**

Constructs a new *InterfaceNotOpenException* class without additional information.

**constructor InterfaceNotOpenException(String s):**

Constructs a new *InterfaceNotOpenException* class with additional information.

---

**Package** [comm](#)**Abstract Class OnCommListener**

extends [Object](#)

Abstract class used as base class for interface specific listener classes such as:

- [OnI2CListener](#)
- [OnSPIListener](#)
- [OnUARTListener](#)

**Note**

- It shouldn't be necessary to use this class directly. Extend one of the subclasses.

**Public Methods**

- [isListening](#)
- [startListening](#)
- [stopListening](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
-

Package [comm](#)

## Class [OnCommListener](#)

### Public Method `isListening`

```
boolean isListening()
```

#### Description:

Returns true if this *OnCommListener* is listening to comm events. By default listeners are listening, i.e. get notified when an event is happening. To stop them from getting notified use the [stopListening](#) method.

---

Package [comm](#)

## Class [OnCommListener](#)

### Public Method `startListening`

```
void startListening()
```

#### Description:

Used to make this *OnCommListener* (*re-*)*start* listening. By default listeners are getting notified, i.e. calling this method is usually done after invoking the [stopListening](#) method.

---

Package [comm](#)

## Class [OnCommListener](#)

### Public Method `stopListening`

```
void stopListening()
```

#### Description:

Use this method to stop this *OnCommListener* from getting notified when an event occurs.

---

Package [comm](#)

## Abstract Class `OnCommBasicListener`

extends [OnCommListener](#) → [Object](#)

Base class for basic comm listeners such as:

- [OnUSBListener](#)
- [OnEthernetListener](#)

Normally it should not be necessary to use this class directly, just extend the subclasses.

### Public Methods

- [onReceive](#)

### Methods inherited from [comm.OnCommListener](#)

- [startListening](#)
- [stopListening](#)
- [isListening](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

## Package [comm](#)

## Class [OnCommBasicListener](#)

### Public Method [onReceive](#)

```
abstract void onReceive(byte[] data)
```

### Throws:

- [Exception](#)

### Description:

This method has to be implemented by the user in order to receive data from a comm interface. It will be automatically called by the [EventManager](#). The incoming data is passed in the byte array *data*.

### Note

- This method is executed in the Event-Thread. Please make sure that data transfer between the Main-Thread and the Event-Thread is correctly synchronized. Consider making it synchronized in the implementing class.
- Due to the fact that this method is executed in the event handling loop it's important to keep it efficient and avoid unnecessary delays because they might block other events from being processed.

## Package [comm](#)

### Abstract Class [OnEthernetListener](#)

extends [OnCommBasicListener](#) → [OnCommListener](#) → [Object](#)

Base class for receiving Ethernet event notifications. Extend this class and implement the *onReceive* method in order to be notified when data arrives via Ethernet.

#### Note

- TCP/IP has to be enabled in the iLCD project settings (Settings Tab in the iLCD Manager XE).
- It's not possible to instantiate this class. To use it create a new class derived from this one.

#### Methods inherited from [comm.OnCommBasicListener](#)

- [onReceive](#)

#### Methods inherited from [comm.OnCommListener](#)

- [startListening](#)
- [stopListening](#)
- [isListening](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

## Package [comm](#)

### Abstract Class [OnI2CListener](#)

extends [OnCommListener](#) → [Object](#)

#### Description:

This class is used as base class for I2C event listeners. Extend this class and implement the following methods:

- [onReceive](#)
- [onRequest](#)

In order to receive I2C events as slave add the listener to the [I2CEventDispatcher](#) of the [EventManagement](#) system.

## Note

- It's not possible to instantiate this class. To use it create a new class derived from this one.

## Public Methods

- [onReceive](#)
- [onRequest](#)

## Methods inherited from [comm.OnCommListener](#)

- [startListening](#)
- [stopListening](#)
- [isListening](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Example

This Example shows how to implement an `OnI2CListener` class allowing to receive data with the iLCD controller in slave mode. The `onReceive()` method is called automatically when the I2C master has addressed the iLCD controller as slave receiver sending data to the slave. The `onRequest()` method is called when the I2C master has addressed the iLCD controller as slave transmitter requesting data to be send by the slave.

```
class I2CListener extends OnI2CListener
{
    private I2C i2cSlave;

    I2CListener(int address)
    {
        i2cSlave = new I2C(address, true);
    }

    public void onReceive()
    {
        if (I2C.getCurrentState() ==
I2C.DATA_RECEIVED_AS_SLAVE_ACK_TRANSMITTED)
        {
            byte[] data = new byte[2];
            try
            {
                int nr_of_bytes_received = i2cSlave.read(data, 0, 2);
            }
            catch (Exception exc)
            {
            }
        }
    }
}
```

```
        Console.println(exc.getMessage());
    }
}

public void onRequest()
{
    try
    {
        byte data = 0x41;
        i2cSlave.write(data);
        while (I2C.getCurrentState() ==
I2C.BYTE_TRANSMITTED_AS_SLAVE_ACK_RECEIVED)
        {
            i2cSlave.write(++data);
        }
    }
    catch (Exception exc)
    {
        Console.println(exc.getMessage());
    }
}
}
```

Package [comm](#)

## Class [OnI2CListener](#)

### Public Method onRequest

```
abstract void onRequest()
```

#### Throws:

- [Exception](#)

#### Description:

The *onRequest()* method is called when the I2C master has addressed the iLCD controller as slave transmitter requesting data to be send by the slave.

#### Note

- This method is executed in the Event-Thread. Please make sure that data transfer between the Main-Thread and the Event-Thread is correctly synchronized. Consider making it synchronized in the implementing class.

**Package [comm](#)****Class [OnI2CListener](#)****Public Method [onReceive](#)**

```
abstract void onReceive()
```

**Throws:**

- [Exception](#)

**Description:**

This method has to be implemented by the user in order to receive data via I2C. It will be automatically called by the [EventManager](#). The incoming data can be obtained calling the [I2C.read](#) method.

**Note**

- This method is executed in the Event-Thread. Please make sure that data transfer between the Main-Thread and the Event-Thread is correctly synchronized. Consider making it synchronized in the implementing class.
  - Due to the fact that this method is executed in the event handling loop it's important to keep it efficient and avoid unnecessary delays because they might block other events from being processed.
- 

**Package [comm](#)****Abstract Class [OnSPIListener](#)**

extends [OnCommListener](#) → [Object](#)

**Description:**

This class is used as base class for SPI event listeners. Extend this class and implement the following methods in order to receive SPI events:

- [onReceive](#)

In order to receive SPI events as slave add the listener to the [SPIEventDispatcher](#) of the [EventManager](#) system.

**Note**

- It's not possible to instantiate this class.

**Public Methods**

- [onReceive](#)



### Methods inherited from [comm.OnCommListener](#)

- [startListening](#)
- [stopListening](#)
- [isListening](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

### Package [comm](#)

### Class [OnSPIListener](#)

#### Public Method onReceive

```
abstract void onReceive(int[] data)
```

#### Throws:

- [Exception](#)

#### Description:

This method has to be implemented by the user in order to receive data from a *comm* interface. It will be automatically called by the [EventManagement](#). The incoming data is passed in the byte array *data*.

#### Note

- This method is executed in the Event-Thread. Please make sure that data transfer between the Main-Thread and the Event-Thread is correctly synchronized. Consider making it synchronized in the implementing class.
- Due to the fact that this method is executed in the event handling loop it's important to keep it efficient and avoid unnecessary delays because they might block other events from being processed.

---

### Package [comm](#)

### Abstract Class [OnUARTListener](#)

extends [OnCommListener](#) → [Object](#)

**Description:**

This class is used as base class for UART event listeners. Extend this class and implement the following methods in order to receive UART events:

- [onReceive](#)

In order to receive UART events add the listener to the [UARTEventDispatcher](#) of the [EventManagement](#) system.

**Note**

- It's not possible to instantiate this class. To use it create a new class derived from this one.

**Public Methods**

- [onReceive](#)

**Methods inherited from [comm.OnCommListener](#)**

- [startListening](#)
- [stopListening](#)
- [isListening](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

**Package [comm](#)****Class [OnUARTListener](#)****Public Method onReceive**

```
abstract void onReceive(int port, byte[] data)
```

**Throws:**

- [Exception](#)

**Description:**

This method has to be implemented by the user in order to receive data from a *comm* interface. It will be automatically called by the [EventManagement](#). The incoming data is passed in the byte array *data*.

## Note

- This method is executed in the Event-Thread. Please make sure that data transfer between the Main-Thread and the Event-Thread is correctly synchronized. Consider making it synchronized in the implementing class.
  - Due to the fact that this method is executed in the event handling loop it's important to keep it efficient and avoid unnecessary delays because they might block other events from being processed.
- 

## Package [comm](#)

### Abstract Class [OnUSBListener](#)

extends [OnCommBasicListener](#) → [OnCommListener](#) → [Object](#)

Base class for receiving USB event notifications. Extend this class and implement the *onReceive* method in order to be notified when data arrives via USB.

In order to receive USB data the *OnUSBListener* has to be added to the [USBEventDispatcher](#) of the [EventManager](#) system.

## Note

- It's not possible to instantiate this class. To use it create a new class derived from this one.

### Methods inherited from [comm.OnCommBasicListener](#)

- [onReceive](#)

### Methods inherited from [comm.OnCommListener](#)

- [startListening](#)
- [stopListening](#)
- [isListening](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

## Package [comm](#)

### Class [SPI](#)

extends [Object](#)

With this class the iLCD controller might be used as either SPI master or SPI slave device.

The Serial Peripheral Interface (*SPI*) is a synchronous and full duplex communication interface. This means that data can be transmitted and received at the same time. SPI can be used in master or in slave mode and is connected via *MISO*, *MOSI*, *SCK*, *SSEL*, *GND* pins to another device.

The frequency range is adjustable via the [open](#) method and can be adjusted from *500kHz* to *7MHz*. The number of bits transferred in each frame can also be adjusted via the [open](#) method and ranging from *8* to *16-bits* per frame.

Separate *RX* and *TX* buffers are used for data transmission. The *TX* buffer can be written with the [writeTXBuffer](#) method and the *RX* buffer can be read with the [readRXBuffer](#) method. Both methods can not initiate a data transfer. A data transfer can be initiated in master mode with the [transferData](#) method. This counts for both master and slave mode. Each buffer has a fixed size of *63* frames.

In order to receive SPI data as slave a [OnSPIListener](#) has to be added to the [SPIEventDispatcher](#) of the [EventManager](#) system.

## Notes

- SPI has to be enabled in the iLCD project. To do this select the corresponding checkbox in the iLCD Manager XE on the Settings Tab under Hardware Settings.
- It's not possible to instantiate this class. The methods in this class are static.
- The *SPI* in master and slave mode needs *4µs* for data processing after a data frame has been received.
- See the Chapter Pin Descriptions in [ILCD Panels Specification.pdf](#) to find out which pins to use for *SCK*, *MISO*, *MOSI* and *SSEL*.

## Public Methods

- [clearBuffer](#)
- [close](#)
- [getBufferCapacity](#)
- [getFreeBufferCapacity](#)
- [open](#)
- [readRXBuffer](#)
- [setTimeout](#)
- [transferData](#)
- [writeTXBuffer](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final byte MASTER
- static final byte RXBUFFER
- static final byte SLAVE
- static final byte TXBUFFER

Package [comm](#)

Class [SPI](#)

### **Public Method `clearBuffer`**

```
static void clearBuffer (byte rxtxBuffer)
```

#### **Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)

#### **Description:**

Parameter	Range	Description
<code>rxtxBuffer</code>	SPI.RXBUFFER or SPI.TXBUFFER	the according buffer that should be cleared

Clears the according buffer. All stored frames in the according buffer gets lost.

#### **Example**

```
SPI.clearBuffer (SPI.RXBUFFER) ;
```

The example above clears the *RX* buffer.

---

Package [comm](#)

Class [SPI](#)

### **Public Method `close`**

```
static void close ()
```

#### **Description:**

This method closes the *SPI* and releases all associated system resources.

#### **Example**

```
SPI.close () ;
```

---

Package [comm](#)

Class [SPI](#)

### Public Method `getBufferCapacity`

```
static int getBufferCapacity(byte rxtxBuffer)
```

Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)

Description:

Parameter	Range	Description
rxtxBuffer	SPI.RXBUFFER or SPI.TXBUFFER	the according buffer

This method is used to get the buffer capacity of the *RX* or *TX* buffer. It returns -1 if a wrong input value is selected.

Return s	Description
val	the capacity of the selected buffer

Example

```
SPI.getBufferCapacity(SPI.RXBUFFER);
```

This method returns the buffer capacity of the according buffer.

---

Package [comm](#)

Class [SPI](#)

### Public Method `getFreeBufferCapacity`

```
static int getFreeBufferCapacity(byte rxtxBuffer)
```

Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)

Description:

Parameter	Range	Description
rxtxBuffer	SPI.RXBUFFER or SPI.TXBUFFER	the according buffer

This method is used to get the current number of frames which can be stored in the according buffer. It returns -1 if a wrong input value is selected.

Returns	Description
val	the free capacity of the selected buffer

#### Note

- The return value is influenced by the [transferData](#) method when the master mode is selected.
- Otherwise, this value is influenced by another master that initiates a data transfer when the slave mode is selected.

#### Example

```
SPI.getFreeBufferCapacity(SPI.RXBUFFER);
```

This method returns the number of available frames that can be stored into the according buffer.

#### Package [comm](#)

#### Class [SPI](#)

#### Public Method [open](#)

```
static void open(int freq, byte dss, byte cpol, byte cpha, byte msm)
```

#### Throws:

- [IllegalArgumentException](#)
- [CommStateException](#)

#### Description:

Parameter	Range	Description
freq	500 ... 7000	clock frequency in kHz
dss	8 ... 16	data size selection ranging from 8 to 16-bit per frame
cpol	0 ... 1	clock polarity ( <i>cpol</i> = 0) produces a steady state low value on the <i>SCK</i> pin when data is not being transferred ( <i>cpol</i> = 1) produces a steady state high value on the <i>SCK</i> pin when data is not being transferred
cpha	0 ... 1	clock phase ( <i>cpha</i> = 0) data is captured on the first clock edge transition ( <i>cpha</i> = 1) data is captured on the second clock edge transition

msm	SPI.MASTER or SPI.SLAVE	master slave mode ( <i>msm = SPI.MASTER</i> ) master mode ( <i>msm = SPI.SLAVE</i> ) slave mode
-----	-------------------------------	---

The *SPI* communication parameter can be adjusted with the `open(int, byte, byte, byte, byte)` method. All parameters for the *SPI* communication are listed in the table above. Bit order is always MSB first.

### Example

The following code example shows how to open the *SPI* for the serial communication.

```
import comm.SPI;
...
int freq = 1000;
byte dss = 16;
byte cpol = 0;
byte cpha = 0;
SPI.open(freq, dss, cpol, cpha, SPI.MASTER);
```

The `comm` package is imported in the first line to make use of the *SPI*. Afterwards, the *SPI* is initialised with the `open` method.

### Notes

- *SPI* has to be enabled in the iLCD project. To do this select the corresponding checkbox in the iLCD Manager XE on the Settings Tab under Hardware Settings.

## Package [comm](#)

### Class [SPI](#)

#### Public Method [readRXBuffer](#)

```
static int readRXBuffer()
```

#### Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)

```
static int readRXBuffer(int[] rxBuf)
```

#### Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)



```
static int readRXBuffer(int[] rxBuf, int offs, int len)
```

**Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****readRXBuffer() method:**

This method reads from the *RX* buffer and returns the value of the stored data frame. -1 is returned if there are no new data frames available in *RX* buffer.

Return s	Description
val	frame from the <i>RX</i> buffer

**readRXBuffer(int[] rxBuf) method:**

Parameter	Description
rxBuf	destination array

The *rxBuf* contains the read data frames from the *RX* buffer. The *rxBuf* is filled with data frames from the *RX* buffer starting at the beginning of the *rxBuf* array. This method returns the number of frames read from the *RX* buffer.

Return s	Description
val	number of returned data frames

**readRXBuffer(int[] rxBuf, int offs, int nrOfFrames) method:**

Parameter	Description
rxBuf	destination array
offs	offset in <i>rxBuf</i>
nrOfFrames	number of data frames

The *rxBuf* contains the read data frames from the *RX* buffer. The *offs* parameter specifies the offset between the first frame of the *rxBuf* and the first frame read into the *rxBuf* from the *RX* buffer. This method returns the number of frames read from the *RX* buffer.

Return s	Description
-------------	-------------

val	number of returned data frames
-----	--------------------------------

### Example

```
int[] data = new int[10];
int ret;
ret = SPI.readRXBuffer(data);
```

The example above reads 10 frames into the *data* array and returns the number of read data frames.

### Note

- The received data frames are stored temporarily in the *RX* buffer in case of a data transfer.
- The *readRXBuffer* method reads from the *RX* buffer.
- *rxBuf* data are not overwritten if there are no new data frames are available.
- A data transfer is initiated by another master if this device is opened in slave mode.
- A data transfer can be initiated with the [transferData](#) method if this device is opened in master mode.
- The value of each frame ranging from 0 ... 255 with a data size select of *8-bit* to 0 ... 65535 with a data size select of *16-bits* (see [open](#)).

## Package [comm](#)

### Class [SPI](#)

#### Public Method [setTimeout](#)

```
static void setTimeout(int ms)
```

#### Throws:

- [InterfaceNotOpenException](#)

#### Description:

Parameter	Range	Description
ms	2 ... 10000	value for timeout in milliseconds

The timeout is used for [transferData](#) method to provide a defined time frame for one data frame. The default value is 2 milliseconds and is automatically set with the [open](#) method.

### Example

The following example sets up a timeout of 10 milliseconds.

```
SPI.setTimeout(10);
```

**See also:**

[I2C](#)  
[UART](#)

---

**Package** [comm](#)**Class** [SPI](#)**Public Method** `transferData`

```
static int[] transferData()
```

**Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)
- [WrongCommModeException](#)
- [CommStateException](#)

```
static int[] transferData(int nrOfFrames)
```

**Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)
- [WrongCommModeException](#)

```
static int[] transferData(int[] txBuf)
```

**Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)
- [WrongCommModeException](#)

```
static int[] transferData(int[] txBuf, int offs, int nrOfFrames)
```

**Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)
- [WrongCommModeException](#)

**Description:****transferData() method:**

This method transfers all stored data frames from the *TX* buffer to a slave device and returns the received data frames as *rxBuf* array after the transfer has been completed. The returned *rxBuf.length* value contains the number of frames that are transferred to a slave device.

Returns	Description
<i>rxBuf</i> []	array that contains the received data frames

**transferData(int nrOfFrames) method:**

Parameter	Description
<i>nrOfFrames</i>	number of the data frames that should be transferred

This method transfers *nrOfFrames* elements from the *TX* buffer to a slave device and returns the received data frames as *rxBuf* array after the transfer has been completed. The returned *rxBuf.length* value contains the number of frames that are transferred to a slave device.

Returns	Description
<i>rxBuf</i> []	array that contains the received data frames

**transferData(int[] txBuf) method:**

This method **clears** the *TX* buffer before the new *txBuf* data frames are written to the *TX* buffer.

Parameter	Description
<i>txBuf</i>	array of frames that should be written to the TX buffer

The *txBuf* contains the array of data frames that should be transmitted to a slave device and returns the received data frames as *rxBuf* array after the transfer has been completed. The returned *rxBuf.length* value contains the number of frames that are transferred to a slave device.

Returns	Description
<i>rxBuf</i> []	array that contains the received data frames

**transferData(int[] txBuf, int offs, int len) method:**

This method **clears** the *TX* buffer before the new *txBuf* data frames are written to the *TX* buffer.

Parameter	Description
<i>txBuf</i>	array of frames that should be written to the TX buffer

<code>offs</code>	offset to the first data frame
<code>nrOfFrames</code>	number of the data frames that should be transfered

The `txBuf` contains the array of data frames and writes `nrOfFrames` elements according to the offset parameter `offs` into the `TX` buffer. This method returns the received data frames as `rxBuf` array after the transfer has been completed. The returned `rxBuf.length` value contains the number of frames that are transferred to a slave device.

Returns	Description
<code>rxBuf[]</code>	array that contains the received data frames

## Notes

- A data transfer with the `transferData` method can only be used in master mode.
- The frames from the `txBuf` array are stored into the `TX` buffer.
- Some methods clear the `TX` buffer before new `txBuf` data are written into the `TX` buffer.
- Afterwards, the data transfer is initiated.
- The value of each frame ranging from `0 ... 255` with a data size select of `8-bit` to `0 ... 65535` with a data size select of `16-bits` (see [open](#)).
- Only the lower `16-bits` of an integer value are transferred.
- The frames to be transfered are taken from the `TX` buffer according to the FIFO principle.
- The received data frames from a slave device are stored in the `RX` buffer and returned in the `rxBuf` array afterwards.

## Package [comm](#)

### Class [SPI](#)

#### Public Method `writeTXBuffer`

```
static int writeTXBuffer(int txBuf)
```

#### Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)

```
static int writeTXBuffer(int[] txBuf)
```

#### Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)

```
static int writeTXBuffer(int[] txBuf, int offs, int len)
```

**Throws:**

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****writeTXBuffer(int txBuf) method:**

Parameter	Description
txBuf	one frame that should be stored in the <i>TX</i> buffer

The *txBuf* contains the data frame that should be stored in the *TX* buffer and transferred to a slave device afterwards. This method returns 1 in case of a *TX* buffer overflow. Otherwise, 0 is returned.

Returns	Description
val	1 if a <i>TX</i> buffer overflow occurs

**writeTXBuffer(int[] txBuf) method:**

Parameter	Description
txBuf	array of frames that should be stored in the <i>TX</i> buffer

The *txBuf* contains the data frames that should be stored in the *TX* buffer and transferred to a slave device afterwards. This method counts and returns the number of data frames in the case of a *TX* buffer overflow. Otherwise, 0 is returned.

Returns	Description
val	number of data frames if a <i>TX</i> buffer overflow occurs

**writeTXBuffer(int[] txBuf, int offs, int nrOfFrames) method:**

Parameter	Description
txBuf	array of frames that should be stored in the <i>TX</i> buffer
offs	offset to the first data frame in <i>txBuf</i>
nrOfFrames	number of data frames that should be stored

The *txBuf* contains the data frames that should be stored in the *TX* buffer according to the *offs* and *len* parameter. This method counts and returns the number of data frames in the case of a *TX* buffer overflow. Otherwise, 0 is returned.

Return s	Description
val	number of data frames if a <i>TX</i> buffer overflow occurs

### Note

- The data frames passed by *writeTXBuffer* method are stored temporarily in a *TX* buffer.
- A *TX* buffer overflow occurs only when a previously stored and not transferred data frame has been overwritten.
- The methods in this class count the overridden data frames and return the count value in case of a *TX* buffer overflow.
- The data frames are taken from the *TX* buffer in case of an data transfer.
- A data transfer is initiated by another master if this device is opened in slave mode.
- A data transfer can be initiated with the [transferData](#) method if this device is opened in master mode.
- The value of each frame ranging from 0 ... 255 with a data size select of 8-bit to 0 ... 65535 with a data size select of 16-bits (see [open](#)).
- Only the lower 16-bits of an integer value are transferred.

### Package [comm](#)

### Class SPIEventDispatcher

extends [CommEventDispatcher](#) → [AsyncEvent](#) → [Object](#)

This class is normally used internally by the [EventManagement](#) system. Add an [OnSPIListener](#) object to the *SPIEventDispatcher* of the *EventManagement* to be notified when SPI data is received.

### Example

```
SPIListener spiListener = new SPIListener ();
EventManagement.getSPIEventDispatcher().addListener(spiListener);
```

### Public Constructors

- [SPIEventDispatcher](#)

### Public Methods

- [addListener](#)
- [removeListener](#)

### Methods inherited from [comm.CommEventDispatcher](#)

- [addListener](#)
- [removeListener](#)
- [removeAllListeners](#)
- [getListenerCount](#)
- [startEventHandling](#)
- [stopEventHandling](#)

- [stopAllCommEvents](#)

#### Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

#### Package [comm](#)

### Class [SPIEventDispatcher](#)

#### Public Constructor SPIEventDispatcher

```
SPIEventDispatcher ()
```

#### Description:

Normally it should not be necessary to create an object of this class, it's handled internally by the [EventManagement](#) system.

---

#### Package [comm](#)

### Class [SPIEventDispatcher](#)

#### Public Method addListener

```
void addListener (OnSPIListener listener)
```

#### Description:

Add a new *OnSPIListener* to this event dispatcher. Listeners will be notified when the corresponding event occurs.

---



Package [comm](#)

## Class [SPIEventDispatcher](#)

### Public Method [removeListener](#)

```
void removeListener(OnSPIListener listener)
```

#### Description:

Remove *OnSPIListener* from this event dispatcher. The listener will not be notified any more about occurring events.

---

Package [comm](#)

## Class [TimeoutException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

An *TimeoutException* can be used to specify an exception for a blocking operation which should be thrown when a defined time expires.

### Public Constructors

- [TimeoutException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

Package [comm](#)

## Class [TimeoutException](#)

### Public Constructor [TimeoutException](#)

```
TimeoutException()
```

**TimeoutException**(String s)**Description:****constructor TimeoutException():**

The specified message is Null when a timeout occurs. It's not possible to define a detailed message with this method.

**constructor TimeoutException(String s):**

Parameter	Description
s	defined exception message

The *s* parameter is used to define a detailed message which should be thrown when this exception occurs.

**See also:**

[UART  
Logger](#)

**Package [comm](#)****Class UART**

extends [Object](#)

The Universal Aynchronous Receiver Transmitter (*UART*) is a asynchronous and full duplex communication interface. The iLCD controller has connections to the serial port's *RX*, *TX* and *GND* pin. The serial port of the controlling application can be set with the *baud rate*, *data*, *parity* and *1*, *1.5* or *2 stop bits*.

The frequency range is adjustable via the [open](#) method and can be adjusted within 12 steps up to a baud rate of *460800*. The number of bits transferred in each frame can also be adjusted via the [open](#) method and ranging from 5 to 8-bits per frame.

Separate *RX* and *TX* buffers are used for data transmission. The *TX* buffer can be written with the [write](#) method and the *RX* buffer can be read with the [read](#) method. The data bytes are transmitted immediately if data are written to the *TX* buffer. Each buffer has a fixed size of *127* bytes.

In order to receive UART data automatically using UART events [OnUARTListener](#) has to be added to the [UARTEventDispatcher](#) of the [EventManager](#) system.

**Public Constructors**

- [UART](#)

## Public Methods

- [clearBuffer](#)
- [close](#)
- [getBufferCapacity](#)
- [getFreeBufferCapacity](#)
- [getNumberOfBytesRead](#)
- [getNumberOfOverriddenBytes](#)
- [getPort](#)
- [open](#)
- [read](#)
- [write](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final int baud\_115200
- static final int baud\_1200
- static final int baud\_19200
- static final int baud\_230400
- static final int baud\_2400
- static final int baud\_300
- static final int baud\_38400
- static final int baud\_460800
- static final int baud\_4800
- static final int baud\_57600
- static final int baud\_600
- static final int baud\_9600
- static final int BITS\_5
- static final int BITS\_6
- static final int BITS\_7
- static final int BITS\_8
- static final int PAR\_EVEN
- static final int PAR\_FORCE0
- static final int PAR\_FORCE1
- static final int PAR\_NONE
- static final int PAR\_ODD
- static final byte RXBUFFER
- static final int STOP\_1
- static final int STOP\_2
- static final byte TXBUFFER
- static final int UART0
- static final int UART1

**Package [comm](#)**  
**Class [UART](#)****Public Constructor UART****UART**(int uartPort)**Throws:**

- [IllegalArgumentException](#)

**Description:**

Parameter	Range	Description
uart	UART0 UART1	UART target port

The iLCD provides two *UART* ports (*UART0* and *UART1*). Each *UART* port provides one *RX* pin for data reception and one *TX* pin to send data. The *UART1* port is selected when a decimal value of 0 or static variable `UART.UART1` is passed to the *UART* constructor. The *UART0* port is selected when a decimal value of 1 or static variable `UART.UART0` is passed to the *UART* constructor.

**Notes**

- *Serial Port 1* has to be enabled in the iLCD project when using the *UART1* port. To do this select the corresponding checkbox in the iLCD Manager XE on the Settings Tab under Hardware Settings.
- The hardware change is required in order to use the *UART1* port. Contact demmel products for details.

**Example**

The following code example shows how to select the *UART0* port for the serial communication.

```
import comm.UART;
...
UART uart;
...
uart = new UART(UART.UART0);
```

The `comm` package is imported in the first line to make use of the *UART* port. A new variable with the data type *UART* is created in the next step. Afterwards, a new instance for the *UART0* port is created.

**See also:**

[SPI](#)  
[I2C](#)

**Package [comm](#)**  
**Class [UART](#)****Public Method `clearBuffer`**

```
void clearBuffer()
```

**Throws:**

- [InterfaceNotOpenException](#)

**Description:**

All received data bytes are stored in a buffer. The `read()` methods are used to read from this buffer till the last data byte was read. Unread data bytes are lost when the `clearBuffer()` method is called. New received data bytes can be read with the `read()` methods after the `clearBuffer()` method was called.

**Example**

```
uart.clearBuffer();
```

**See also:**

[SPI](#)  
[I2C](#)

---

**Package [comm](#)**  
**Class [UART](#)****Public Method `close`**

```
void close()
```

**Description:**

The `close()` method is used when a particular `UART` port is no longer needed.

**Example**

```
uart.close();
```

**See also:**

[SPI](#)  
[I2C](#)

---

Package [comm](#)

Class [UART](#)

### Public Method [getBufferCapacity](#)

```
int getBufferCapacity(byte rxtxBuffer)
```

#### Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)

#### Description:

Parameter	Range	Description
rxtxBuffer	UART.RXBUFFER or UART.TXBUFFER	the according buffer

This method is used to get the buffer capacity of the *RX* or *TX* buffer. It returns -1 if a wrong input value is selected.

Return s	Description
val	the capacity of the selected buffer

#### Example

```
uart.getBufferCapacity(UART.RXBUFFER);
```

This method returns the buffer capacity of the according buffer.

Package [comm](#)

Class [UART](#)

### Public Method [getFreeBufferCapacity](#)

```
int getFreeBufferCapacity(byte rxtxBuffer)
```

#### Throws:

- [InterfaceNotOpenException](#)
- [IllegalArgumentException](#)

#### Description:

Parameter	Range	Description
-----------	-------	-------------

<code>rxtxBuffer</code>	UART.RXBUFFER or UART.TXBUFFER	the according buffer
-------------------------	-----------------------------------	----------------------

This method is used to get the current number of frames which can be stored in the according buffer. It returns -1 if a wrong input value is selected.

Returns	Description
<code>val</code>	the free capacity of the selected buffer

### Example

```
uart.getFreeBufferCapacity(UART.RXBUFFER);
```

This method returns the number of available frames that can be stored into the according buffer.

---

## Package [comm](#)

## Class [UART](#)

### Public Method [getNumberOfBytesRead](#)

```
int getNumberOfBytesRead()
```

#### Description:

Returns	Description
<code>value</code>	number of bytes read

Returns the number of bytes read from the *RX buffer*. This value is returned and updated by one of the read methods.

### Example

```
int numberOfBytesRead = uart.getNumberOfBytesRead();
```

The code above reads the number of bytes read.

#### See also:

[read](#)  
[getNumberOfOverriddenBytes](#)  
[SPI](#)  
[I2C](#)

---

**Package [comm](#)**  
**Class [UART](#)****Public Method [getNumberOfOverriddenBytes](#)**

```
int getNumberOfOverriddenBytes ()
```

**Description:**

Return s	Description
value	number of overridden bytes

Returns the number of overridden bytes in case of a *RX buffer* overflow has occurred. An internal counter is incremented by one every time if the *RX buffer* is full and a new byte was received. The first byte or bytes that are read by the according *read* method gets lost in the case of a buffer overflow. The internal counter is reseted to 0 if this method is called.

**Example**

```
int numberOfOverriddenBytes = uart.getNumberOfOverriddenBytes();
```

The code above reads the number of overridden bytes.

**See also:**

[read](#)  
[getNumberOfBytesRead](#)  
[SPI](#)  
[I2C](#)

---

**Package [comm](#)**  
**Class [UART](#)****Public Method [getPort](#)**

```
int getPort ()
```

**Description:**

Returns the UART port used by this UART object (either *UART.UART0* or *UART.UART1*).



**Package [comm](#)**  
**Class [UART](#)**

**Public Method open**

```
boolean open(int baud, int parity, int bits, int stop)
```

**Throws:**

- [IllegalArgumentException](#)
- [CommStateException](#)

**Description:**

Parameter	Value	Description
baud	UART.baud_300 UART.baud_600 UART.baud_1200 UART.baud_2400 UART.baud_4800 UART.baud_9600 UART.baud_19200 UART.baud_38400 UART.baud_57600 UART.baud_115200 UART.baud_230400 UART.baud_460800	baud rate of the UART port
parity	UART.PAR_NONE UART.PAR_ODD UART.PAR_EVEN UART.PAR_FORCE1 UART.PAR_FORCE0	parity check
bits	UART.BITS_5 UART.BITS_6 UART.BITS_7 UART.BITS_8	number of data bits
stop	UART.STOP_1 UART.STOP_2	number of stop bits

The *UART* communication parameter can be adjusted with the *open(int, int, int, int)* method once an instance of the *UART* class is created. All parameters for the *UART* communication are listed in the table above.

Returns	Range	Description
opened	true ... false	whether the UART port can be opened or not

The *open()* method returns true when the UART port can be opened.

**Example**

The following example opens the UART port.

```
if
(uart.open(UART.baud_115200,UART.STOP_1,UART.BITS_8,UART.PAR_NONE))
    Draw.writeText("the serial port is open");
```

**See also:**

[SPI](#)  
[I2C](#)

---

**Package [comm](#)****Class [UART](#)****Public Method read**

```
int read()
```

**Throws:**

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

```
int read(byte[] buf)
```

**Throws:**

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

```
int read(byte[] buf, int offset, int len)
```

**Throws:**

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****read() method:**

Returns	Description
value	a single byte

The `read()` method returns one single byte or -1 if nothing was read.

**read(byte[] buf) method:**

Parameter	Description
buf	array storing the returned data

The `read(byte[])` method reads data from the *UART RX buffer*. The first parameter `buf` is used to return the received data bytes in an array of bytes. The number of data bytes to be read at once are specified by the `buf.length` parameter. This method returns the number of bytes read from the *RX buffer*.

Returns	Description
value	number of returned bytes

**read(byte[] buf, int offset, int len) method:**

Parameter	Description
buf	array storing the returned data
offset	offset of bytes
len	count of bytes

The `read(byte[], int, int)` method reads data from the *UART RX buffer*. The first parameter `buf` is used to return the received data bytes in an array of bytes. The number of data bytes to be read at once are specified by the `len` parameter. The `offset` parameter specifies the offset to the first byte in the `buf` array. This method returns the number of bytes read from the *RX buffer*.

Returns	Description
value	number of returned bytes

**Example**

```
try
{
    retCnt = uart.read(rxData);
}
catch (CommStateException ise)
{
    Console.println("error: " + ise);
    int numberOfOverriiddenBytes = uart.getNumberOfOverriddenBytes();
    if (numberOfOverriiddenBytes != 0)
    {
        Console.print("Buffer overflow, number of overridden bytes: ")
    }
}
```

```
        + numberOfOverriедdenBytes + " - data: ");
    for (int i = 0; i < uart.getNumberofBytesRead(); i++)
        Console.print("" + (char)rxData[i]);
    Console.println("");
}
}
```

## Note

- The received bytes are stored temporarily in the *RX* buffer.
- The *read* method reads from the *RX* buffer.
- A buffer overflow occurs in case the *RX buffer* is full and additional data is received.
- The *getNumberofOverriедdenBytes* method should be called in case of a *RX buffer* overflow to indicate how many bytes are overridden.
- The *getNumberofBytesRead* method can be used to get the number of bytes returned in the *buf* array with the last call to the *read* method.

## See also:

[getNumberofBytesRead](#)  
[getNumberofOverriедdenBytes](#)  
[SPI](#)  
[I2C](#)

---

## Package [comm](#)

## Class [UART](#)

### Public Method write

```
int write(byte value)
```

#### Throws:

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

```
int write(byte[] buf)
```

#### Throws:

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

```
int write(byte[] buf, int offset, int len)
```

**Throws:**

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

```
int write(String value)
```

**Throws:**

- [CommStateException](#)
- [InterfaceNotOpenException](#)
- [NullPointerException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****write(byte value) method:**

Parameter	Description
value	byte that should be sent

The *value* paramter is a single byte which should be transmitted.

**write(byte[] source, int offs, int count) method:**

Parameter	Description
source	bytes that should be sent
offs	offset to the first data byte
count	number of the data byte

The *source* parameter is a byte array which should be transferred. The number of data bytes must be specified by the *count* parameter. The *offs* paramter specifies the offset to the first byte in the *source* byte array.

**write(String source) method:**

Parameter	Description
source	String that should be sent

The *source* parameter is a string which should be transmitted.

**Note**

- The bytes passed by *write* method are stored temporarily in a *TX* buffer.

- A *TX* buffer overflow occurs only when a previously stored and not transmitted byte has been overwritten.
- The methods in this class count the overridden bytes and return the count value in case of a *TX* buffer overflow.
- The bytes are taken from the *TX* buffer in case of an data transfer.
- A data transfer is initiated after the byte array is written into the *TX* buffer.

### Example

```
uart.write("text message");
```

The code above writes one whole text message over the UART port.

### See also:

[SPI](#)  
[I2C](#)

---

## Package [comm](#)

### Class **UARTEventDispatcher**

extends [CommEventDispatcher](#) → [AsyncEvent](#) → [Object](#)

This class is normally used internally by the [EventManagement](#) system. Add an [OnUARTListener](#) object to the *UARTEventDispatcher* of the *EventManagement* to be notified when UART data is received.

### Example

```
UARTListener uartListener = new UARTListener();  
EventManagement.getUARTEventDispatcher().addListener(uartListener);
```

### Public Constructors

- [UARTEventDispatcher](#)

### Public Methods

- [addListener](#)
- [removeListener](#)

### Methods inherited from [comm.CommEventDispatcher](#)

- [addListener](#)
- [removeListener](#)
- [removeAllListeners](#)
- [getListenerCount](#)
- [startEventHandling](#)
- [stopEventHandling](#)
- [stopAllCommEvents](#)

**Methods inherited from [javax.events.AsyncEvent](#)**

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

**Package [comm](#)****Class [UARTEventDispatcher](#)****Public Constructor UARTEventDispatcher**

```
UARTEventDispatcher ()
```

**Description:**

Normally it should not be necessary to create an object of this class, it's handled internally by the [EventManagement](#) system.

---

**Package [comm](#)****Class [UARTEventDispatcher](#)****Public Method addListener**

```
void addListener (OnUARTListener listener)
```

**Description:**

Add a new *OnUARTListener* to this event dispatcher. Listeners will be notified when the corresponding event occurs.

---

**Package [comm](#)****Class [UARTEventDispatcher](#)****Public Method [removeListener](#)**

```
void removeListener(OnUARTListener listener)
```

**Description:**

Remove *OnUARTListener* from this event dispatcher. The listener will not be notified any more about occurring events.

---

**Package [comm](#)****Class [USB](#)**

extends [Comm](#) → [Object](#)

The iLCD controller can be used as USB slave device in order to communicate with a USB host (typically an operating system). The iLCD panel might be identified with the following interface data:

- product string "iLCD Controller"
- manufacturer string: "demmel products"
- Vendor ID: 0x0483
- Product ID: 0x1513

See [OnUSBListener](#) on how to receive USB data via the [EventManager](#) system.

**Public Methods**

- [getInstance](#)
- [isWriteDone](#)

**Methods inherited from [comm.Comm](#)**

- [open](#)
- [close](#)
- [read](#)
- [write](#)
- [getReceiveDataSize](#)
- [getTransmitDataSize](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)



**Package [comm](#)****Class [USB](#)****Public Method getInstance**

```
static USB getInstance()
```

**Description:**

Returns the only instance of this class.

**Example**

Typical usage:

```
USB.getInstance().open();
```

---

**Package [comm](#)****Class [USB](#)****Public Method isWriteDone**

```
boolean isWriteDone()
```

**Description:**

Returns false directly after `USB.getInstance().write()` until the write operation is completed.

---

**Package [comm](#)****Class USBEventDispatcher**

extends [CommEventDispatcher](#) → [AsyncEvent](#) → [Object](#)

This class is normally used internally by the [EventManagement](#) system. Add an [OnUSBListener](#) object to the `USBEventDispatcher` of the `EventManagement` to be notified when USB data is received.

**Example**

The class `USBListener` extends [OnUSBListener](#) and implements the method [onReceive](#).

```
USBListener usbListener = new USBListener();  
EventManagement.getUSBEventDispatcher().addListener(usbListener);
```

## Public Constructors

- [USBEventDispatcher](#)

## Public Methods

- [addListener](#)
- [removeListener](#)

## Methods inherited from [comm.CommEventDispatcher](#)

- [addListener](#)
- [removeListener](#)
- [removeAllListeners](#)
- [getListenerCount](#)
- [startEventHandling](#)
- [stopEventHandling](#)
- [stopAllCommEvents](#)

## Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

## Package [comm](#)

## Class [USBEventDispatcher](#)

### Public Constructor USBEventDispatcher

```
USBEventDispatcher ()
```

#### Description:

Normally it should not be necessary to create an object of this class, it's handled internally by the [EventManagement](#) system.

Package [comm](#)

## Class [USBEventDispatcher](#)

### Public Method [addListener](#)

```
void addListener(OnUSBListener listener)
```

#### Description:

Add a new *OnUSBListener* to this event dispatcher. Listeners will be notified when the corresponding event occurs.

---

Package [comm](#)

## Class [USBEventDispatcher](#)

### Public Method [removeListener](#)

```
void removeListener(OnUSBListener listener)
```

#### Description:

Removes the specified *OnUSBListener* from this event dispatcher. It will not receive any further event notifications.

---

Package [comm](#)

## Class [WrongCommModeException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

Thrown if a method is used that is only available in Slave or Master mode.

### Public Constructors

- [WrongCommModeException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)

- [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

Package [comm](#)

## Class [WrongCommModeException](#)

### Public Constructor WrongCommModeException

```
WrongCommModeException()
```

```
WrongCommModeException(String s)
```

#### Description:

##### constructor [WrongCommModeException\(\)](#):

Constructs a new *WrongCommModeException* class without additional information.

##### constructor [WrongCommModeException\(String s\)](#):

Constructs a new *WrongCommModeException* class with additional information.

---

## Package [hw](#)

Contains classes for setting port pins, digital or analog IOs and the console class for writing messages to the iLCD Manager XE console.

It also includes classes for keyboard event management, data conversion and handling data and time information.

## Interfaces

- [OnKeyboardListener](#)
- [OnRotaryEncoderListener](#)

## Classes

- [Console](#)
- [Conversion](#)
- [Date](#)
- [DateTime](#)
- [IO](#)
- [IOException](#)
- [KeyboardEventDispatcher](#)
- [Logger](#)

- [RotaryEncoderEventDispatcher](#)
  - [Time](#)
- 

## Package [hw](#)

### Class Console

extends [Object](#)

The methods in the *Console* class provide a simple means of communication between the console window in the iLCD Manager XE *console* and the *iLCD panel*. Both characters and strings can be send to the iLCD Manager XE console window. From the iLCD Manager XE each character entered in the console window will be send to the iLCD panel immediatly.

#### Note

- It's not possible to instantiate this class. The methods in this class are static.

#### Public Methods

- [getKey](#)
- [getKeys](#)
- [keyAvail](#)
- [print](#)
- [println](#)
- [putch](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- static final char BS
  - static final char CR
  - static final char ESC
  - static final char LF
  - static final char TAB
-

Package [hw](#)  
Class [Console](#)

**Public Method `getKey`**

```
static char getKey()
```

**Description:**

Returns	Description
key	reads character wise from console

This method waits for an input from the iLCD Manager XE console. Each character entered into the console is immediately transmitted to the panel. The `getKey()` method returns a character after one character is received from the console.

**Example**

```
char c = Console.getKey();
Console.println("" + c);
```

Package [hw](#)  
Class [Console](#)

**Public Method `getKeys`**

```
static String getKeys()
```

**Description:**

Returns	Description
keys	reads multiple characters at once from the console

This method waits for an input from the iLCD Manager XE console. Each character entered into the console is immediately transmitted to the panel. The `getKeys()` method stops the processing of the Java application until the enter key on the keyboard is pressed.

**Note**

- A run-time exception occurs when an escape character is sent to the .
- A backspace erase the last character.

**Example**

```
String str = Console.getKeys();
Console.println(str);
```

Package [hw](#)  
Class [Console](#)

**Public Method `keyAvail`**

```
static boolean keyAvail()
```

**Description:**

Returns	Description
keyAvail	new character available or not

Delivers true if a character is sent from the console.

**Example**

```
if (Console.keyAvail())  
{  
    String str = Console.getKeys();  
    Console.println(str);  
}
```

Package [hw](#)  
Class [Console](#)

**Public Method `print`**

```
static void print(String s)
```

**Description:**

Parameter	Description
s	sends a String to the console

The iLCD Manager XE receives a message from the panel and prints it to the console.

**Note**

- UNICODE characters are sent to the console.

**Example**

```
Console.print(" This message was sent by the panel! ");
```

**Package [hw](#)**  
**Class [Console](#)****Public Method println**

```
static void println(String s)
```

**Description:**

Parameter	Description
s	sends a String to the console

The iLCD Manager XE receives a message from the panel and prints it to the console. The cursor of the console begins in a new line after a message is received from the panelJoC board.

**Note**

- UNICODE characters are sent to the console.

**Example**

```
Console.println(" This message was sent by the panel! ");
```

**Package [hw](#)**  
**Class [Console](#)****Public Method putch**

```
static void putch(char ch)
```

**Description:**

Parameter	Description
ch	sends a character to the console

The iLCD Manager XE receives only one character from the panel and prints it to the console.

**Note**

- UNICODE characters are sent to the console.

**Example**

```
char c = 'h';  
Console.putch(c);
```



## Package [hw](#)

### Class Conversion

extends [Object](#)

Class containing methods for basic data type conversion.

#### Public Methods

- [boolToByte](#)
- [byteToUnsignedInt](#)
- [byteToUnsignedShort](#)
- [dwordToInt](#)
- [floatToByte](#)
- [floatToString](#)
- [intArrToString](#)
- [stringToIntArr](#)
- [wordToInt](#)
- [wordToUnsignedInt](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- static boolean DONT\_SWAP\_ELEMENTS
  - static boolean SWAP\_ELEMENTS
- 

## Package [hw](#)

### Class [Conversion](#)

#### Public Method [boolToByte](#)

```
static byte boolToByte(boolean value)
```

#### Description:

Parameter	Description
value	true or false

Convert boolean to byte, resulting in 0 for false and 1 for true.

---

Package [hw](#)

## Class [Conversion](#)

### Public Method `byteToUnsignedInt`

```
static int byteToUnsignedInt(byte value)
```

Description:

Parameter	Description
value	(signed) byte value

Convert a (signed) byte value (-128 to 128) into an unsigned int value (0 to 255).

---

Package [hw](#)

## Class [Conversion](#)

### Public Method `byteToUnsignedShort`

```
static short byteToUnsignedShort(byte value)
```

Description:

Parameter	Description
value	(signed) byte value

Convert a (signed) byte value (-128 to 128) into an unsigned short value (0 to 255).

---

Package [hw](#)

## Class [Conversion](#)

### Public Method `dwordToInt`

```
static int dwordToInt(byte[] dword)
```

Throws:

- [IllegalArgumentException](#)

**Package [hw](#)****Class [Conversion](#)****Public Method [floatToByte](#)**

```
static byte[] floatToByte(float value)
```

**Description:**

Parameter	Description
value	4 byte floating point value

Convert a 4 byte floating point value in a 4 element byte array.

---

**Package [hw](#)****Class [Conversion](#)****Public Method [floatToString](#)**

```
static String floatToString(float f, int nrOfDecimalPlaces)
```

**Description:**

Parameter	Description
f	float that should convertet into a String
nrOfDecimalPlaces	number of digits after the comma that should be visible in the returned String

The parameter f that should be converted into a String with a number of decimal places that should be displayed.

Returns	Description
String	a new created string

---

**Package [hw](#)****Class [Conversion](#)****Public Method [intArrToString](#)**

```
static String intArrToString(int[] intArr)
```

```
static String intArrToString(int[] intArr, int offset)
```

**Throws:**

- [ArrayIndexOutOfBoundsException](#)

**Description:****intArrToString(int[] intArr) method:**

Parameter	Description
<code>intArr</code>	integer array that could be converted to a string

The parameter `intArr` is an array of integer values that should representing a string.

Returns	Description
<code>String</code>	a new created string

**intArrToString(int[] intArr, int offset) method:**

Parameter	Description
<code>intArr</code>	integer array that could be converted to a string
<code>offset</code>	offset in the specified integer array

The parameter `intArr` is an array of integer values that should representing a string. The `offset` parameter specifies the offset in the integer array with the result that the returned string starts at that position.

Returns	Description
<code>String</code>	a new created string

**Package [hw](#)****Class [Conversion](#)****Public Method [stringToIntArr](#)**

```
static int[] stringToIntArr(String str)
static int[] stringToIntArr(String str, int offset)
```

**Description:****stringToIntArr(String str) method:**

Parameter	Description
-----------	-------------

r	
str	string that could be converted to a integer array

The parameter *str* is a string that should be converted in an integer array.

Return s	Description
int []	a new created integer array

#### **stringToIntArr(String str, int offset) method:**

Parameter	Description
intArr	integer array that could be converted to a string
offset	offset in the specified integer array

The parameter *str* is a string that should be converted in an integer array. The *offset* parameter specifies the offset in the string with the result that the returned integer array starts at that position.

Return s	Description
int []	a new created integer array

Package [hw](#)

Class [Conversion](#)

**Public Method [wordToInt](#)**

```
static int wordToInt(byte[] word)
```

**Throws:**

- [IllegalArgumentException](#)

**Description:**

Parameter	Description
word	2-element byte array representing a word

Convert a word, represented by a 2-element byte array, into an integer value (-32768 to 32767).

**Package [hw](#)****Class [Conversion](#)****Public Method [wordToUnsignedInt](#)**

```
static int wordToUnsignedInt(byte[] word)
```

**Throws:**

- [IllegalArgumentException](#)

```
static int wordToUnsignedInt(byte[] word, boolean swapElements)
```

**Throws:**

- [IllegalArgumentException](#)

**Description:**

Parameter	Description
word	2-element byte array representing a word
boolean	swap the array elements

Convert a word, represented by a 2-element byte array, into an unsigned integer value (0 to 65535).

---

**Package [hw](#)****Class [Date](#)**

extends [Object](#)

This class provides methods for getting date information. The [IO.getDate\(\)](#) method returns a *Date* object, containing year, month, day and weekday of the date at creation time derived from the real-time clock. Use the getter methods to access the individual values.

**Note**

- To set the date please use the [IO.setDate\(int, int, int, int\)](#) method.

**Public Constructors**

- [Date](#)

**Public Methods**

- [getDay](#)
- [getMonth](#)
- [getWeekDay](#)
- [getYear](#)

- [toString](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### See also:

[hw.Time](#)  
[java.util.Date](#)

---

### Package [hw](#)

### Class [Date](#)

#### **Public Constructor Date**

```
Date(int year, int month, int day, int weekDay)
```

#### Description:

Parameter	Range	Description
year	0 ... 99	year value Date object
month	1 ... 12	month value Date object
day	1 ... 31	day value Date object
weekDay	0 ... 6	weekDay value Date object

This constructor is used to set up the current date of an *Date* object.

---

### Package [hw](#)

### Class [Date](#)

#### **Public Method getDay**

```
int getDay()
```

#### Description:

Returns	Range	Description
---------	-------	-------------

int	1 ... 31	day value of Date object
-----	----------	--------------------------

Get day value from a *Date* object. The Date object is returned from *IO.getDate()* and will contain the date at the time of the method call, derived from the real-time clock.

### Example

```
Date saveCurrentDate = IO.getDate();
Console.println("stored day: " + saveCurrentDate.getDay());
```

### See also:

[IO.getDate\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)

## Package [hw](#) Class [Date](#)

### Public Method getMonth

```
int getMonth()
```

### Description:

Returns	Range	Description
int	1...12	month value of Date object

Get month value from a *Date* object. The Date object is returned from *IO.getDate()* and will contain the date at the time of the method call, derived from the real-time clock.

### Example

```
Date saveCurrentDate = IO.getDate();
Console.println("stored month: " + saveCurrentDate.getMonth());
```

### See also:

[IO.getDate\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)



Package [hw](#)  
Class [Date](#)

**Public Method `getWeekDay`**

```
int getWeekDay()
```

**Description:**

Returns	Range	Description
int	0...6	weekday value of Date object

Get weekday value from a *Date* object. The Date object is returned from *IO.getDate()* and will contain the date at the time of the method call, derived from the real-time clock.

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Example**

```
Date saveCurrentDate = IO.getDate();
Console.println("stored weekday: " + saveCurrentDate.getWeekDay());
```

**See also:**

[IO.getDate\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)

Package [hw](#)  
Class [Date](#)

**Public Method `getYear`**

```
int getYear()
```

**Description:**

Returns	Range	Description
int	0...99	year value Date object

Get year value from a *Date* object. The Date object is returned from *IO.getDate()* and will contain the date at the time of the method call, derived from the real-time clock.

## Example

```
Date saveCurrentDate = IO.getDate();
Console.println("stored year: " + saveCurrentDate.getYear());
```

## See also:

[IO.getDate\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)

---

## Package [hw](#) Class [Date](#)

### Public Method `toString`

```
String toString()
```

### Overrides:

- [toString](#) in class [Object](#)

### Description:

Returns	Description
String	year, month, and day within a single string

Returns year, month, and day of a *Date* object in one string ("year/month/day"). The *Date* object is returned from *IO.getDate()* and will contain the date at the time of the method call, derived from the real-time clock.

### Note

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

## Example

```
Date saveCurrentDate = IO.getDate();
Console.println("stored date: " + saveCurrentDate.toString());
```

## See also:

[IO.getDate\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)

---

**Package [hw](#)****Class [DateTime](#)**extends [Object](#)

Class providing access to date and time information of a *DateTime* object. The [IO.getDateTime\(\)](#) method returns a *DateTime* object, containing date and time information from the real-time clock. Use the getter methods to access the individual values.

**Note**

- To set date and time please use the [IO.setDate\(int, int, int, int\)](#) and [IO.setTime\(int, int, int\)](#) methods.

**Public Constructors**

- [DateTime](#)

**Public Methods**

- [getDate](#)
- [getTime](#)
- [now](#)
- [toString](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [hw](#)****Class [DateTime](#)****Public Method [getDate](#)**Date [getDate\(\)](#)**Description:**

Return s	Description
<a href="#">Date</a>	date value of a <i>DateTime</i> object

Get a *Date* object from a *DateTime* object. The *DateTime* object is returned from the *IO.getDateTime()* method.

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Example**

```
DateTime myDateTime = IO.getDateTime();
Date      myDate      = myDateTime.getDate();
Console.println("stored month: " + myDate.getMonth());
```

**See also:**

[IO.getDateTime\(\)](#)  
[IO.getDate\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)  
[IO.setTime\(int, int, int\)](#)  
[Date](#)

**Package [hw](#)****Class [DateTime](#)****Public Method [getTime](#)**

```
Time getTime()
```

**Description:**

Returns	Description
<a href="#">Time</a>	time value of a DateTime object

Get a *Time* object from a *DateTime* object. The *DateTime* object is returned from the *IO.getDateTime()* method.

**Example**

```
DateTime myDateTime = IO.getDateTime();
Time      myTime      = myDateTime.getTime();
Console.println("stored minute: " + myTime.getMinute());
```

**See also:**

[IO.getDateTime\(\)](#)  
[IO.getTime\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)  
[IO.setTime\(int, int, int\)](#)  
[Time](#)

---

Package [hw](#)

## Class [DateTime](#)

### Public Method [now](#)

```
static DateTime now()
```

#### Throws:

- [IOException](#)

#### Description:

Returns	Description
<a href="#">DateTime</a>	current date and time

Returns a *DateTime* object which contains the year, month, day, and weekday, or hour, minute, and second at the time of the call.

#### Note

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

#### Example

```
DateTime myDateTime = IO.getDateTime();  
Console.println("stored DateTime: " + myDateTime.now());
```

#### See also:

[IO.getDateTime\(\)](#)

[IO.getDate\(\)](#)

[IO.getTime\(\)](#)

[IO.setDate\(int, int, int, int\)](#)

[IO.setTime\(int, int, int\)](#)

[Date](#)

[Time](#)

---

**Package [hw](#)****Class [DateTime](#)****Public Method [toString](#)**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

Returns	Description
String	year, month, day, hour, minute, second within a single string

A new created *DateTime* object can be used to save the current date derived from the real-time clock. The date and the time are assigned to a *DateTime* object when the *IO.getDateTime()* method is called. Once a *DateTime* object is created, the year, month, day, weekday, hour, minute, and second values at the objects creation time will be stored into the *DateTime* object for further use. The *toString()* method returns the year, month, day, hour, minute, and second in one string ("year/month/day hour:minute:second").

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Example**

```
DateTime saveCurrentDateTime = IO.getDateTime();
Console.println("stored date and time: " +
saveCurrentDateTime.toString());
```

**See also:**

[IO.getDateTime\(\)](#)

[IO.getDate\(\)](#)

[IO.getTime\(\)](#)

[IO.setDate\(int, int, int, int\)](#)

[IO.setTime\(int, int, int\)](#)

**Package [hw](#)****Class [DateTime](#)****Public Constructor [DateTime](#)**

```
DateTime(Date inDate, Time inTime)
```

**Description:**

Parameter	Description
Data	Date object
Time	Time object

This constructor is used to set up the current date and the time of an *DateTime* object.

---

**Package [hw](#)****Class IO**

extends [Object](#)

The iLCD controllers allow most port pins to be assigned as digital or analog inputs, outputs (pull down or push/pull) or keyboard columns via iLCD Manager XE. All port pin methods below refer to the logical port name, not the physical port pin name.

This means the physical port pin "Keyboard column 3" may have e.g. the logical name "Output #9", so turning on this pin (= setting it to high when it is defined as a push/pull pin) can be done via the [IO.setOutput\(int, int\)](#) method.

When using the same port as a pull-down output port by using the [IO.setOutput\(int, int\)](#) method pulls the pin to low (making it "active"). Note that the Color iLCD controllers can only source/sink 4mA. Please check the documentation for your particular board.

**Public Methods**

- [enableDisableCommunicationPorts](#)
- [getADCValue](#)
- [getCurrentCommunicationPort](#)
- [getDate](#)
- [getDateTime](#)
- [getEnabledCommunicationPorts](#)
- [getInputsState](#)
- [getKeyboardState](#)
- [getRotaryEncoderValue](#)
- [getTime](#)
- [relaysOneShot](#)
- [retrieveNextKeyboardEvent](#)
- [setDACValue](#)
- [setDate](#)
- [setKeyboardEnabled](#)
- [setKeyboardReportingEnabled](#)
- [setMultipleOutputs](#)
- [setOutput](#)
- [setOutputBlinkFrequency](#)
- [setPWM0](#)
- [setPWM1](#)
- [setRelaysOnOffPWM](#)
- [setRotaryEncoderEnabled](#)
- [setRotaryEncoderPeriod](#)

- [setRotaryEncoderReportingEnabled](#)
- [setTime](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final byte ANALOG\_IN\_0
- static final byte ANALOG\_IN\_1
- static final byte ANALOG\_IN\_2
- static final byte ANALOG\_IN\_3
- static final byte OUTPUT\_SET\_TO\_BLINK
- static final byte OUTPUT\_SET\_TO\_OFF
- static final byte OUTPUT\_SET\_TO\_ON
- static final byte PORT\_ETHERNET
- static final byte PORT\_I2C
- static final byte PORT\_ID\_ETHERNET
- static final byte PORT\_ID\_I2C
- static final byte PORT\_ID\_SERIAL\_0
- static final byte PORT\_ID\_SERIAL\_1
- static final byte PORT\_ID\_SERIAL\_CURRENT
- static final byte PORT\_ID\_SPI
- static final byte PORT\_ID\_USB
- static final byte PORT\_SERIAL\_0
- static final byte PORT\_SERIAL\_1
- static final byte PORT\_SPI
- static final byte PORT\_USB
- static final byte RELAY\_0
- static final byte RELAY\_1
- static final byte RELAYS\_ENABLE\_PWM\_ON\_RELAY
- static final byte RELAYS\_SET\_OFF
- static final byte RELAYS\_SET\_ON

---

### Package [hw](#)

### Class [IO](#)

### Public Method enableDisableCommunicationPorts

```
static void enableDisableCommunicationPorts(int portMask, int timeout)
```

### Throws:

- [IOException](#)



**Description:**

Parameter	Range	Description
portMask	Bits 1 ... 6	bitmask for communication-port
timeout	0 ... 65535	timeout in units of 10ms

Disables the communication-ports that are 0-bits in the *portMask*. After the *timeout* period, all ports are enabled again. The bits of are defined as follows:

portMask
PORT_SERIAL_0
PORT_SERIAL_1
PORT_USB
PORT_I2C
PORT_SPI
PORT_ETHERNET

**Note**

- Using this method again will restart the timeout.
- Unused bits of the byte are ignored. Hence, all ports are disabled with a *portMask* of 0x00 and enabled with 0xFF.

**Example**

```
IO.enableDisableCommunicationPorts (IO.PORT_SERIAL_0 |
                                     IO.PORT_SERIAL_0 |
                                     IO.PORT_I2C |
                                     IO.PORT_SPI |
                                     IO.PORT_ETHERNET,
                                     500) ;
```

Disables the USB port for 5 seconds.

**See also:**

[Controlling Options](#)

[IO.setEnabledCommunicationPorts\(\)](#)

[IO.getCurrentCommunicationPort\(\)](#)

Package [hw](#)

Class [IO](#)

### Public Method `getADCValue`

```
static int getADCValue(int port)
```

#### Throws:

- [IOException](#)

#### Description:

Parameter	Range	Description
port	0 ... 3	logical port name

Returns the value of any of the 4 ADC (Analog Digital Converter) ports defined as "Analog In" in the "I/O Settings" section of iLCD Manager XE's "Settings" page.

Returns	Description
value	value of the requested ADC

#### Note

- The DPC3090's ADCs have a resolution of 12 bits, resulting in a value range of decimal 0 ... 4095 (0xFFFF) where 0 is an input voltage of 0V and 4095 is an input voltage of 3.3V, the value of the supply voltage of the iLCD controller.

#### Example

```
int value = IO.getADCValue(2);
```

#### See also:

[IO](#)

---

Package [hw](#)

Class [IO](#)

### Public Method `getCurrentCommunicationPort`

```
static int getCurrentCommunicationPort()
```

#### Throws:

- [IOException](#)

**Description:**

Returns	Range	Description
portId	1 ... 6	communication-port in use

Returns the current communication-port in use.

portId
PORT_ID_SERIAL_0
PORT_ID_SERIAL_1
PORT_ID_USB
PORT_ID_I2C
PORT_ID_SPI
PORT_ID_ETHERNET

**Example**

```
(IO.getCurrentCommunicationPort() == IO.PORT_ID_USB)
```

If this expression evaluates to true, the current communication-port is the USB port.

**See also:**

[Controlling Options](#)

**Package [hw](#)****Class [IO](#)****Public Method [getDate](#)**

```
static Date getDate()
```

**Throws:**

- [IOException](#)

**Description:**

Returns	Description
static <a href="#">Date</a>	date of the real-time clock

The `getDate()` method returns the year, the month, and the day in one string ("year/month/day").

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Example**

```
Date date    = IO.getDate();
int  day     = date.getDay();
int  weekDay = date.getWeekDay();
```

If the weekDay is e.g. is 3 it means Wednesday (when automatically set by iLCD Manager XE). Alternatively a value can be obtained with a single call:

```
String date = IO.getDate().toString();
```

**See also:**

[IO.getDateTime\(\)](#)

[IO.getTime\(\)](#)

[IO.setDate\(int, int, int, int\)](#)

[IO.setTime\(int, int, int\)](#)

**Package [hw](#)****Class [IO](#)****Public Method [getDateTime](#)**

```
static DateTime getDateTime()
```

**Throws:**

- [IOException](#)

**Description:**

Returns	Description
static <a href="#">DateTime</a>	date and time of the real-time clock

The *getDateTime()* method returns the year, the month, the day, the hour, the minute, and the second in one string ("year/month/day hout:minute:second").

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.

- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.
- It is not possible to obtain the current weekday with the `getDateTime()` method. To obtain the current weekday please use the `IO.getDate().getWeekDay()` method.

### Example

```
DateTime date = IO.getDateTime();
String date   = date.getDate().toString();
String time   = date.getTime().toString();
```

Alternatively a value can be obtained with a single call:

```
String dateTime = IO.getDateTime().toString();
```

### See also:

[IO.getDate\(\)](#)  
[IO.getTime\(\)](#)  
[IO.setDate\(int, int, int, int\)](#)  
[IO.setTime\(int, int, int\)](#)

## Package [hw](#)

### Class [IO](#)

#### Public Method [getEnabledCommunicationPorts](#)

```
static int getEnabledCommunicationPorts()
```

#### Throws:

- [IOException](#)

#### Description:

Returns	Range	Description
portMask	Bits 1 ... 6	mask of the current communication-ports

Returns a port mask where a 0-bit means that the corresponding communication-port is currently disabled.

Description
PORT_SERIAL_0
PORT_SERIAL_1
PORT_USB

PORT_I2C
PORT_SPI
PORT_ETHERNET

**Example**

```
(IO.PORT_USB == IO.getEnabledCommunicationPorts())
```

If this expression evaluates to true, currently all communication-ports except USB are disabled.

**See also:**

[Controlling Options](#)

[enableDisableCommunicationPorts\(int, int\)](#)

[IO.getCurrentCommunicationPort\(\)](#)

Package [hw](#)

Class [IO](#)

**Public Method `getInputsState`**

```
static int getInputsState()
```

**Throws:**

- [IOException](#)

**Description:**

Returns	Range	Description
state	0x0000 ... 0xFFFF	every set bit represents an active input

The `getInputState()` method returns the current state of all inputs in a bit mask.

**Note**

- The inputs are reported in a bit orientated manner, where bit 0 of `state` refers to Input #0 and bit 15 refers to Input #15.
- Undefined inputs are reported as low, pins defined as inputs but left unconnected (=floating) can have any value.

**Example**

```
((IO.getInputsState() & (1 << 7)) == (1 << 7))
```

If this expression evaluates to true, the input #7 is currently active.

```
((IO.getInputState() & (1 << 7)) == 0)
```

If this expression evaluates to true, the input #7 is currently inactive.

**See also:**

[IO](#)

**Package [hw](#)**

**Class [IO](#)**

**Public Method `getKeyboardState`**

```
static int[] getKeyboardState()
```

**Throws:**

- [IOException](#)

**Description:**

Return s	Range	Description
state	0x00000000 ... 0xFFFFFFFF	every set bit represents an active input

The `getKeyboardState()` method returns the state of all keys asynchronously.

**Note**

- When keyboard scanning is disabled, reading the keyboard state will always return 0x00 for all columns (see [IO.enableDisableKeyboard\(boolean\)](#)).
- Keyboard columns not having assigned a KeybCol # via iLCD Manager XE report all corresponding keys as not pressed.

**Example**

```
int[] button = IO.getKeyboardState();
if ((button[3] & (1 << 4)) == (1 << 4))
```

If this expression evaluates to true, in column #3, the button connecting to row #4 is pressed.

```
if ((button[3] & (1 << 2)) == 0)
```

If this expression evaluates to true, in column #3, the button connecting to row #2 is not pressed.

**See also:**

[IO.enableDisableKeyboard\(boolean\)](#)  
[IO.enableDisableKeyboardReporting\(boolean\)](#)

---

Package [hw](#)

Class [IO](#)

### Public Method getRotaryEncoderValue

```
static byte getRotaryEncoderValue()
```

#### Throws:

- [IOException](#)

#### Description:

Return s	Range	Description
value	-128 ... 127	measured encoder values within the timeout period

A positive or negative value is returned by the `getRotaryEncoderValue()` method. The returned value depends on the *direction* and the *speed* of the encoder movement and also on the adjusted period ([setRotaryEncoderPeriod\(int\)](#)). The returned value will be greater if the encoder is turned faster. This counts for positive returned values when the encoder is moved in one direction. A negative value will be provided if the encoder is moved in the other direction. In this case the value will be smaller if the encoder is moved faster.

#### Note

- The rotary encoder must be enabled within the 'Settings' of the development environment.
- The encoder is disabled by default. It can be enabled or disabled by the [setRotaryEncoderEnabled\(boolean\)](#) method.

#### Example

```
byte value = IO.getRotaryEncoderValue();
```

#### See also:

[IO.setRotaryEncoderPeriod\(int\)](#)

[IO.setRotaryEncoderEnabled\(boolean\)](#)

[IO.setRotaryEncoderReportingEnabled\(boolean\)](#)

---

Package [hw](#)

Class [IO](#)

### Public Method getTime

```
static Time getTime()
```



**Throws:**

- [IOException](#)

**Description:**

Returns	Description
static <a href="#">Time</a>	provides information about the current time

The `getTime()` method returns the current time of the real-time clock.

**Example**

```
Time time = IO.getTime();
int hour  = time.getHour();
int min   = time.getMinute();
int sec   = time.getSecond();
```

Alternatively a value can be obtained with a single call:

```
String time = IO.getTime().toString();
```

**See also:**

[IO.setTime\(int, int, int\)](#)  
[IO.getDate\(\)](#)

**Package [hw](#)****Class [IO](#)****Public Method `relaysOneShot`**

```
static void relaysOneShot(int relayNo, int mode, int time)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
relayNo	0 ... 1	number of the relay
mode	0 ... 2	mode of the relay
time	0 ... 65535	period of the state change in units of 10ms

The `relaysOneShot()` method switches the specified relay for the given time to the specified `mode`:

mode
RELAYS_SET_OFF
RELAYS_SET_ON
RELAYS_ENABLE_PWM_ON_RELAY

#### Note

- The maximum value for time is 65535 (0xFFFF), which will give a maximum one-shot duration of close to 11 minutes.
- The corresponding relay output pulses with a *PWM* signal instead of being statically switched on or off.

#### Example

```
IO.relaysOneShot(1, IO.RELAYS_SET_ON, 300);
```

Turns relay number 1 on for 3 seconds.

#### See also:

[IO](#)

[IO.setRelaysOnOffPWM\(int, int\)](#)

---

Package [hw](#)

Class [IO](#)

#### Public Method `retrieveNextKeyboardEvent`

```
static KeyboardEvent retrieveNextKeyboardEvent()
```

---

Package [hw](#)

Class [IO](#)

#### Public Method `setDACValue`

```
static void setDACValue(int value)
```

#### Throws:

- [IOException](#)

**Description:**

Parameter	Range	Description
value	0 ... 1023	DAC output value

This method is used to adjust the DAC output voltage to the specified value. When the system starts up the DAC is adjustet to the value 0.

**Note**

- The DAC has a resolution of 10 bits, resulting in a value range of decimal 0 ... 1023 (0x03FF). The decimal value 0 responds to an output voltage of 0V. The decimal value 1023 responds to an output voltage of 3.3V .
- The DAC is available for DCP3090 only.
- An exception is thrown if the "DAC Out" is not set within the "I/O Settings" tab.

**Example**

```
IO.setDACValue(100);
```

**See also:**

[IO](#)  
[Set DAC Value](#)

**Package [hw](#)**  
**Class [IO](#)**

**Public Method [setDate](#)**

```
static void setDate(int year, int month, int day, int weekday)
```

**Throws:**

- [IllegalArgumentException](#)
- [IOException](#)

**Description:**

Parameter	Range	Description
year	0 ... 99	year 2000 ... 2099
month	1 ... 12	month
day	1 ... 31	day
weekday	0 ... 6	weekday as numerical representation

The `setDate()` method sets the date of the real-time clock.

**Note**

- There is no connection between the actual date and the value *weekday*. It is simply incremented every day, so it's up to the user to assign a weekday format.
- When automatically storing graphics on the MicroSD card, iLCD Manager XE sets the date where a *weekday* of 0 means Sunday.

**Example**

```
IO.setDate(16, 3, 15, 4);
```

Sets the date to march 15th, 2016 and assigns 4 to the weekday.

**See also:**

[IO.getDateTime\(\)](#)

[IO.getTime\(\)](#)

[IO.getDate\(\)](#)

[IO.setTime\(int, int, int\)](#)

**Package [hw](#)****Class [IO](#)****Public Method setKeyboardEnabled**

```
static void setKeyboardEnabled(boolean enabled)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
enabled	false ... true	determines whether keyboard is on (true) or off (false)

Enables or disables keyboard scanning.

**Note**

- When keyboard scanning is disabled, reading the keyboard state will return the same result as if no key is pressed (see [IO.getKeyboardState\(\)](#)).
- Enabling/Disabling the keyboard does not change keyboard reporting (see [IO.setKeyboardReportingEnabled\(boolean\)](#)), although a disabled keyboard will not report any keys.
- The keyboard is enabled by the default. This can be changed on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

## Example

```
IO.setKeyboardEnabled(false);
```

Disables the keyboard.

### See also:

[IO.setKeyboardReportingEnabled\(boolean\)](#)

---

## Package [hw](#)

## Class [IO](#)

### Public Method [setKeyboardReportingEnabled](#)

```
static void setKeyboardReportingEnabled(boolean enabled)
```

### Throws:

- [IOException](#)

### Description:

Parameter	Range	Description
enabled	false ... true	determines whether keyboard reporting is on (true) or off (false)

Enables or disables keyboard reporting. Also starts the Event Management by invoking the method [EventManagement.start\(\)](#).

### Note

- Switching keyboard reporting off (*enabled* = false) avoids stopping macros.
- Keyboard reporting is enabled by the default. This can be changed on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

## Example

```
IO.setKeyboardReportingEnabled(true);
```

Activates keyboard reporting.

### See also:

[IO.setKeyboardReportingEnabled\(boolean\)](#)

---

**Package [hw](#)****Class [IO](#)****Public Method [setMultipleOutputs](#)**

```
static void setMultipleOutputs(int outMask, int blinkMask)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
outMask	0x0000 ... 0xFFFF	every set bit represents an output turned on
blinkMask	0x0000 ... 0xFFFF	every set bit represents a blinking output

Sets multiple outputs, similar to the [IO.setOutput\(int, int\)](#) method.

**Note**

- The two masks refer to bit addressed outputs. Where bit 0 refers to output #0, bit 15 to output #15.
- If pin is not previously defined as an output via iLCD Manager XE, setting this bit has no effect.
- The startup values can be defined on the "Settings" page of iLCD Manager XE. All outputs will be automatically set to the startup value on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
IO.setMultipleOutputs(0xA, 0x500);
```

This turns on outputs #1 and #3 (0xA = Bin 0000 0000 0000 1010) and sets outputs #8 and #10 (0x500 = Bin 0000 0101 0000 0000) to blink.

**See also:**

[IO](#)  
[IO.setOutput\(int, int\)](#)  
[IO.setOutputBlinkFrequency\(int\)](#)

---

**Package [hw](#)****Class [IO](#)****Public Method [setOutput](#)**

```
static void setOutput(int outNo, int mode)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
outNo	0 ... 15	output number to be turned on/off
mode	0 ... 2	mode to set

This method is used for controlling outputs via the following modes:

mode
OUTPUT_SET_TO_OFF
OUTPUT_SET_TO_ON
OUTPUT_SET_TO_BLINK

**Note**

- Multiple outputs can be controlled with the [IO.setMultipleOutputs\(int, int\)](#) method.
- When the output is set to blink, the frequency can be adjusted with the [IO.setOutputBlinkFrequency\(int\)](#) method.
- The startup values can be defined on the "Settings" page of iLCD Manager XE. All outputs will be automatically set to the startup value on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
IO.setOutput(3, IO.OUTPUT_SET_TO_BLINK);
```

After sending this command, output number 3 will blink (periodically toggles state after the interval specified by the blink frequency).

**See also:**[IO](#)

[IO.setMultipleOutputs\(int, int\)](#)

[IO.setOutputBlinkFrequency\(int\)](#)

**Package [hw](#)**  
**Class [IO](#)****Public Method [setOutputBlinkFrequency](#)**

```
static void setOutputBlinkFrequency(int period)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
period	1 ... 255	interval of state changes in units of 10ms

Sets the blinking frequency for all outputs by defining the interval of state changes.

**Note**

- When an output is currently in blink mode (see [IO.setOutput\(int, int\)](#) and [IO.setMultipleOutputs\(int, int\)](#)), the frequency changes immediately.
- The default value for *period* is 20, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
IO.setOutputBlinkFrequency(25);
```

This will set the blinking frequency to 2 Hertz (250ms on and 250ms off).

**See also:**

[IO](#)  
[IO.setOutput\(int, int\)](#)  
[IO.setMultipleOutputs\(int, int\)](#)

---

**Package [hw](#)**  
**Class [IO](#)****Public Method [setPWM0](#)**

```
static void setPWM0(int freq, int dutyCycle)
```

**Throws:**

- [IOException](#)



**Description:**

Parameter	Range	Description
freq <sup>1</sup>	1 ... 1000000	frequency in Hertz
dutyCycle	1 ... 9999	duty cycle in units of 0.01%

Configures the PWM 0 (on relay output 0) with the specified parameters.

**Note**

- Relay output 0 has to be set to PWM mode for this setting to become relevant (see [IO.setRelaysOnOffPWM\(int, int\)](#)).
- On startup *dutyCycle* is set to 50% (5000) duty cycle automatically.
- <sup>1</sup> DPM5050 is adjustable from 20 Hz to 1 MHz. *IOException* will be thrown if a smaller or a greater value is applied.

**Example**

```
IO.setPWM0(500, 8000);
```

After calling this method, the PWM 0 on relay output 0 is running with 500 Hertz and a duty cycle of 80%. The resulting signal will have a rising edge every 2ms (1/500Hz = 0.002s) and stay high for 1.6ms (80% of 2ms).

**See also:**

[IO.setPWM1\(int\)](#)  
[IO](#)

**Package** [hw](#)  
**Class** [IO](#)

**Public Method setPWM1**

```
static void setPWM1(int dutyCycle)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
dutyCycle	1 ... 9999	duty cycle in units of 0.01%

Configures the PWM 1 (on relay output 1) with the specified duty cycle.

**Note**

- The PWM 1 has a fixed frequency of 1kHz.
- Relay output 1 has to be set to PWM mode for this setting to become relevant (see [IO.setRelaysOnOffPWM\(int, int\)](#)).
- On startup *dutyCycle* is set to 50% (5000) duty cycle automatically.

**Example**

```
IO.setPWM1(2000);
```

After calling this method, the PWM 1 on relay output 1 is running with a duty cycle of 20%. The resulting signal will have a rising edge every 1ms (1/1kHz = 1ms) and stay high for 0.2ms (20% of 1ms).

**See also:**

[IO.setPWM0\(int, int\)](#)  
[IO](#)

**Package [hw](#)****Class [IO](#)****Public Method [setRelaysOnOffPWM](#)**

```
static void setRelaysOnOffPWM(int relayNo, int mode)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
relayNo	0 ... 1	number of the relay
mode	0 ... 2	mode of the relay

The *setRelaysOnOffPWM()* method sets relay output *relayNo* to one of the following values for *mode*:

mode
RELAYS_SET_OFF
RELAYS_SET_ON
RELAYS_ENABLE_PWM_ON_RELAY

**Note**

- Setting *mode* to 2 enables the PWM 0 on relay output 0 or PWM 1 on relay output 1.
- By default, both relay outputs are off. This can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the commands [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
IO.setRelaysOnOffPWM(0, IO.RELAYS_ENABLE_PWM_ON_RELAY);
```

Enables PWM 0 on relay output 0.

**See also:**[IO](#)

[IO.relaysOneShot\(int, int, int\)](#)

**Package [hw](#)****Class [IO](#)****Public Method setRotaryEncoderEnabled**

```
static void setRotaryEncoderEnabled(boolean enabled)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
enabled	false ... true	determines whether rotary encoder is enabled (true) or disabled (false)

This method enables or disables the rotary encoder functionality.

**Note**

- The rotary encoder is disabled by default and should be enabled within the 'Settings' of the development environment.
- This method must be called with true to enable the rotary encoder reporting functionality (see [setRotaryEncoderReportingEnabled\(boolean\)](#)).

**Example**

```
IO.setRotaryEncoderEnabled(true);
```

**See also:**

[setRotaryEncoderPeriod\(int\)](#)  
[getRotaryEncoderValue\(\)](#)  
[setRotaryEncoderReportingEnabled\(boolean\)](#)

---

**Package [hw](#)****Class [IO](#)****Public Method [setRotaryEncoderReportingEnabled](#)**

```
static void setRotaryEncoderReportingEnabled(boolean enabled)
```

**Throws:**

- [IOException](#)

**Description:**

Parameter	Range	Description
enabled	false ... true	determines whether rotary encoder events are reported (true) or not (false)

This method enables or disables the reporting of rotary encoder events. The *OnRotaryEncoderListener* class must be implemented to make use of the reporting and the automatically called *onRotaryEncoderMovement()* method (see [OnRotaryEncoderListener](#)).

**Note**

- The rotary encoder is disabled by default must be enabled within the 'Settings' of the development environment.
- The [setRotaryEncoderEnabled\(\)](#) method must be called to enable the rotary encoder functionality.

**Example**

```
IO.setRotaryEncoderEnabled(true) ;  
IO.setRotaryEncoderReportingEnabled(true) ;
```

**See also:**

[setRotaryEncoderPeriod\(int\)](#)  
[getRotaryEncoderValue\(\)](#)  
[setRotaryEncoderEnabled\(boolean\)](#)  
[OnRotaryEncoderListener](#)

---

Package [hw](#)

Class [IO](#)

### Public Method `setRotaryEncoderPeriod`

```
static void setRotaryEncoderPeriod(int value)
```

#### Throws:

- [IOException](#)

#### Description:

Parameter	Range	Description
value	2 ... 25	10 millisecond steps

This method sets the measurement period. A new value is provided by the `getRotaryEncoderValue()` after this period.

#### Note

- The default value is 20 milliseconds.

#### Example

```
IO.setRotaryEncoderPeriod(20);
```

The example above sets the encoder period to 200 milliseconds.

#### See also:

[getRotaryEncoderValue\(\)](#)  
[setRotaryEncoderEnabled\(boolean\)](#)  
[setRotaryEncoderReportingEnabled\(boolean\)](#)

---

Package [hw](#)

Class [IO](#)

### Public Method `setTime`

```
static void setTime(int hour, int minute, int second)
```

#### Throws:

- [IllegalArgumentException](#)
- [IOException](#)

**Description:**

Parameter	Range	Description
hour	0 ... 23	hour of the clock
minute	0 ... 59	minutes of the clock
second	0 ... 59	seconds of the clock

The `setTime()` method sets the time of the real-time clock.

**Example**

```
IO.setTime(14, 15, 32);
```

Sets the clock to 14:15:32.

**See also:**

[IO.getDateTime\(\)](#)

[IO.getTime\(\)](#)

[IO.getDate\(\)](#)

[IO.setDate\(int, int, int, int\)](#)

**Package [hw](#)****Class [IOException](#)**

extends [Exception](#) → [Throwable](#) → [Object](#)

**Public Constructors**

- [IOException](#)

**Public Methods**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)

- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final int E\_COMMAND
- static final int E\_IO\_ADCCHANNEL
- static final int E\_IO\_ADCENABLE
- static final int E\_IO\_BAUDRATE
- static final int E\_IO\_COMPORTE
- static final int E\_IO\_CRON\_CNT
- static final int E\_IO\_CRON\_CNT\_OV
- static final int E\_IO\_CRON\_EDGE
- static final int E\_IO\_CRON\_ENABLE
- static final int E\_IO\_CRON\_NO\_VALUE
- static final int E\_IO\_CRON\_NOT\_OPEN
- static final int E\_IO\_CRON\_PIN
- static final int E\_IO\_CRON\_TURNED\_OFF
- static final int E\_IO\_DT\_DAY
- static final int E\_IO\_DT\_DAYOFWEEK
- static final int E\_IO\_DT\_MONTH
- static final int E\_IO\_DUTY
- static final int E\_IO\_FREQ
- static final int E\_IO\_LEDDEFINED
- static final int E\_IO\_LEDMODE
- static final int E\_IO\_LEDPAR
- static final int E\_IO\_PWM\_DUTY\_CYCLE
- static final int E\_IO\_PWM\_FREQ
- static final int E\_IO\_PWM\_ON\_OFF
- static final int E\_IO\_PWM\_OPEN
- static final int E\_IO\_PWM\_PIN
- static final int E\_IO\_PWM\_PORT\_PIN
- static final int E\_IO\_PWM\_RESOLUTION
- static final int E\_IO\_REL\_MODE
- static final int E\_IO\_REL\_NUMBER
- static final int E\_IO\_ROTARY\_ENABLED
- static final int E\_IO\_ROTARY\_TIMEVAL
- static final int E\_IO\_RTC\_READ
- static final int E\_IO\_RTC\_WRITE
- static final int E\_IO\_TM\_HOUR
- static final int E\_IO\_TM\_MINUTE
- static final int E\_IO\_TM\_SECOND
- static final int E\_NOT\_BOOLEAN

---

## Package [hw](#)

### Class [IOException](#)

#### **Public Constructor IOException**

```
IOException(int code)
```

**IOException** (String description)

**Description:**

**constructor IOException(int code):**

Constructs a new *IOException* class with the specified code number. The *Public Fields* (e.g. *IOException.E\_IO\_TM\_HOUR*) can be used as code number to apply to a specified exception.

**constructor IOException(String description):**

Constructs a new *IOException* class with additional string description.

**Package [hw](#)**

**Class [IOException](#)**

**Public Method [getErrorCode](#)**

`int getErrorCode ()`

**Description:**

Return s	Range	Description
int	<i>Public Fields, -1</i>	last error code

This method returns the last error code if *IOException(int)* was used. *-1* is returned if the *IOException(String)* was used.

**Package [hw](#)**

**Class [KeyboardEventDispatcher](#)**

extends [AsyncEvent](#) → [Object](#)

The iLCD API implements the commonly used observer pattern. Listeners (observers) subscribe to certain events distributed by an dispatcher object that notifies the listeners when an event occurs. The [KeyboardEventDispatcher](#) class provides this kind of dispatcher functionality for distributing keyboard events. The actual KeyboardEventDispatcher object is part of the class [EventManagement](#).

**Notes**

- In order to use keyboard event handling the abstract class [Application](#) has to be extended and the [OnKeyListener](#) interface has to be implemented.
- [EventManagement](#) is automatically started when invoking the method [setKeyboardReportingEnabled](#).



- Characters can be assigned to keys in the iLCD Manager XE on the Settings Tab under Keyboard Settings.
- See the Chapter Keyboard Port in [ILCD Panels Specification.pdf](#) to find out which pins to use on the connector board.

### Public Constructors

- [KeyboardEventDispatcher](#)

### Public Methods

- [addListener](#)
- [getListenerCount](#)
- [processEvents](#)
- [removeAllListeners](#)
- [removeListener](#)

### Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

### Package [hw](#)

## Class [KeyboardEventDispatcher](#)

### Public Constructor [KeyboardEventDispatcher](#)

```
KeyboardEventDispatcher ()
```

#### Description:

Constructor creating a keyboard event dispatcher object. Normally these objects are created internally by the [EventManager](#) system.

Package [hw](#)

## Class [KeyboardEventDispatcher](#)

### Public Method addListener

```
void addListener(OnKeyboardListener listener)
```

#### Description:

Parameter	Description
listener	Keyboard Listener object

Add the specified *OnKeyboardListener* to this *KeyboardEventDispatcher*.

---

Package [hw](#)

## Class [KeyboardEventDispatcher](#)

### Public Method getListenerCount

```
int getListenerCount()
```

#### Description:

Return the number of *OnKeyboardListener* objects listening to this *KeyboardEventDispatcher*.

---

Package [hw](#)

## Class [KeyboardEventDispatcher](#)

### Public Method processEvents

```
void processEvents()
```

#### Description:

If a keyboard event occurs the [processEvents\(\)](#) method will notify all keyboard event listeners by calling their implementation of the [onKeyDown](#) and the [onKeyUp](#) methods.

#### Note

- This method will be called from the abstract class [Application](#) in its [run\(\)](#) method. It shouldn't normally be called by the user except an *Application* class is not present.

## Example

```
EventManager.getKeyboardEventDispatcher().processEvent();
```

## See also:

[Application](#)

---

## Package [hw](#)

### Class [KeyboardEventDispatcher](#)

#### Public Method [removeAllListeners](#)

```
void removeAllListeners()
```

#### Description:

Remove all *OnKeyboardListener* objects from this *KeyboardEventDispatcher*.

---

## Package [hw](#)

### Class [KeyboardEventDispatcher](#)

#### Public Method [removeListener](#)

```
void removeListener(OnKeyboardListener listener)
```

#### Description:

Parameter	Description
listener	Keyboard Listener object

Remove the specified *OnKeyboardListener* from this *KeyboardEventDispatcher*.

---

## Package [hw](#)

### Class [Logger](#)

extends [Object](#)

The *Logger* class is used to log messages. This messages are sent to the iLCD Manager XE console window. There are three logging levels available. The logging level with the highest priority is the *error* logging level. One level below is the *warn* level. The lowest level is the *debug* level.

All messages will be written to the console window by default. It's possible to switch off lower logging levels to ensure only error messages are written to the console, or write all logging messages without debug informations. Finally, it's also possible to switch off all logging levels.

### Public Methods

- [d](#)
- [e](#)
- [log](#)
- [w](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final int DEBUG
- static final int ERROR
- static int LOG\_LEVEL
- static final int NO\_LOGGING
- static final int WARN

---

### Package [hw](#)

### Class [Logger](#)

#### Public Method [d](#)

```
static void d(String msg)
```

#### Description:

Parameter	Description
msg	message to the console

The *d* method is used to log debugging information only. The parameter *msg* provides an additional and user defined logging message which will be written to the iLCD Manager XE console window.

#### Note

- ☞ It's possible to switch off this message level with the static variable *LOG\_LEVEL*.
- ☞ No debug messages will be written to the console window if the value of the *LOG\_LEVEL* variable is equal to the value of the final static variable *WARN*, *ERROR* or *NO\_LOGGING*.

## Example

```
Logger.LOG_LEVEL = Logger.DEBUG;
Logger.d("this is a debug infromation");
```

The first line ensures the logger debug level. The following line writes the message to the console window.

---

## Package [hw](#) Class [Logger](#)

### Public Method e

```
static void e(String msg)
```

#### Description:

Parameter	Description
msg	message to the console

The `e` method is used to log error information only. The parameter `msg` provides an additional and user defined logging message which will be written to the iLCD Manager XE console window.

#### Note

- ☞ It's possible to switch off this message level with the static variable `LOG_LEVEL`.
- ☞ No error messages will be written to the console window if the value of the `LOG_LEVEL` variable is equal to the value of the final static variable `NO_LOGGING`.

## Example

```
Logger.LOG_LEVEL = Logger.ERROR;
Logger.e("this is a error information");
```

The first line ensures the logger error level. The following line writes the defined message to the console window.

---

## Package [hw](#) Class [Logger](#)

### Public Method log

```
static void log(String msg)
```

**Description:**

Parameter	Description
<code>msg</code>	message to the console

The `log` method has the same functionality as the [Console.println\(String\)](#) method. The parameter `msg` provides a user defined message which is written to the iLCD Manager XE console window.

**Example**

```
Logger.log("This message is written to the console window!");
```

**See also:**

[Console](#)

---

**Package [hw](#)****Class [Logger](#)****Public Method [w](#)**

```
static void w(String msg)
```

**Description:**

Parameter	Description
<code>msg</code>	message to the console

The `w` method is used to log warning information only. The parameter `msg` provides an additional and user defined logging message which will be written to the iLCD Manager XE console window.

**Note**

- ☞ It's possible to switch off this message level with the static variable `LOG_LEVEL`.
- ☞ No warn messages will be written to the console window if the value of the `LOG_LEVEL` variable is equal to the value of the final static variable `ERROR`, or `NO_LOGGING`.

**Example**

```
Logger.LOG_LEVEL = Logger.WARN;
Logger.w("this is a warning information");
```

The first line ensures the logger warn level. The following line writes the defined message to the console window.

---

**Package [hw](#)****Interface [OnKeyboardListener](#)**

This interface is used for receiving keyboard events. Keyboard event listeners (classes implementing this interface) are required to implement the `onKeyDown()` and `onKeyUp()` methods in order to receive information about keyboard events. The two methods will be called automatically by the event management system when a keyboard event occurs after a listener object has been added to the [EventManager](#).

To receive keyboard events, keyboard reporting has to be enabled. This is done using the method [IO.setKeyboardReportingEnabled\(boolean\)](#). Also a keyboard event listener object has to be added to the event system by calling [addListener\(OnKeyboardListenerObject\)](#).

**Note**

- In order to use keyboard event handling the abstract class [Application](#) has to be extended.

**Public Methods**

- [onKeyDown](#)
- [onKeyUp](#)

**Package [hw](#)****Interface [OnKeyboardListener](#)****Public Method [onKeyDown](#)**

```
abstract void onKeyDown(char key)
```

**Throws:**

- [Exception](#)

**Description:**

Parameter	Description
key	keyboard character

This method is called automatically when a keyboard event arrives.

**Note**

- ☞ Characters can be assigned to keys in the iLCD Manager XE on the Settings Tab under Keyboard Settings.

**See also:**

[KeyboardEventDispatcher](#)

Package [hw](#)

## Interface [OnKeyboardListener](#)

### Public Method `onKeyUp`

```
abstract void onKeyUp(char key)
```

#### Throws:

- [Exception](#)

#### Description:

Parameter	Description
key	keyboard character

This method is called automatically when a keyboard event arrives.

#### Note

☞ Characters can be assigned to keys in the iLCD Manager XE on the Settings Tab under Keyboard Settings.

#### See also:

[KeyboardEventDispatcher](#)

---

Package [hw](#)

## Interface `OnRotaryEncoderListener`

This interface is used to get information about the rotary encoder movement. Rotary encoder event listeners (classes implementing this interface) requires the implementation of the `onRotaryEncoderMovement()` method in order to receive information about rotary encoder movement. This method will be called automatically by the event management system when a movement of the rotary encoder is detected.

### Public Methods

- [onRotaryEncoderMovement](#)
-



Package [hw](#)

## Interface [OnRotaryEncoderListener](#)

### Public Method [onRotaryEncoderMovement](#)

```
abstract void onRotaryEncoderMovement(byte value)
```

#### Throws:

- [Exception](#)

#### Description:

Parameter	Description
value	state of all selected input pins

This method is called automatically when the input state change e.g. from high to low level.

---

Package [hw](#)

## Class [RotaryEncoderEventDispatcher](#)

extends [AsyncEvent](#) → [Object](#)

The iLCD API implements the commonly used observer pattern. Listeners (observers) subscribe to certain events distributed by an dispatcher object that notifies the listeners when an event occurs. The [RotaryEncoderEventDispatcher](#) class provides this kind of dispatcher functionality for distributing rotary encoder events.

#### Public Constructors

- [RotaryEncoderEventDispatcher](#)

#### Public Methods

- [addListener](#)
- [getListenerCount](#)
- [removeAllListeners](#)
- [removeListener](#)

#### Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)
- [fire](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

## Package [hw](#)

### Class [RotaryEncoderEventDispatcher](#)

#### Public Method `addListener`

```
void addListener (OnRotaryEncoderListener listener)
```

#### Description:

Parameter	Description
listener	rotary encoder Listener object

Add the specified *OnRotaryEncoderListener* to this *RotaryEncoderEventDispatcher*.

---

## Package [hw](#)

### Class [RotaryEncoderEventDispatcher](#)

#### Public Method `getListenerCount`

```
int getListenerCount ()
```

#### Description:

Return the number of *OnRotaryEncoderListener* objects listening to this *RotaryEncoderEventDispatcher*.

---

## Package [hw](#)

### Class [RotaryEncoderEventDispatcher](#)

#### Public Method `removeAllListeners`

```
void removeAllListeners ()
```

**Description:**

Remove all *OnRotaryEncoderListener* objects from this *RotaryEncoderEventDispatcher*.

---

**Package [hw](#)****Class [RotaryEncoderEventDispatcher](#)****Public Method removeListener**

```
void removeListener(OnRotaryEncoderListener listener)
```

**Description:**

Parameter	Description
listener	rotary encoder listener object

Remove the specified *OnRotaryEncoderListener* from this *RotaryEncoderEventDispatcher*.

---

**Package [hw](#)****Class [RotaryEncoderEventDispatcher](#)****Public Constructor RotaryEncoderEventDispatcher**

```
RotaryEncoderEventDispatcher()
```

**Description:**

Constructor creating a rotary encoder event dispatcher object.

---

**Package [hw](#)****Class Time**

extends [Object](#)

The class *Time* is used to store time information. Call [IO.getTime\(\)](#) to get a *Time* object, containing the actual time of the method call. The time is derived from the real-time clock of the controller. To set the time please use the [IO.setTime\(int, int, int\)](#) method.

## Public Constructors

- [Time](#)

## Public Methods

- [delay](#)
- [getHour](#)
- [getMinute](#)
- [getSecond](#)
- [setHour](#)
- [setMinute](#)
- [setSecond](#)
- [toString](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## See also:

[java.lang.System.currentTimeMillis](#)

[hw.Date](#)

[java.util.Date](#)

## Package [hw](#)

## Class [Time](#)

### Public Constructor Time

```
Time(int hour, int minute, int second)
```

#### Description:

Parameter	Range	Description
hour	0 ... 23	hour value Date object
minute	0 ... 59	minute value Date object
second	0 ... 59	second value Date object

This constructor is used to set up the current time of an *Time* object.

**Package [hw](#)**  
**Class [Time](#)****Public Method delay**

```
static void delay(int ms)
```

**Description:**

Returns	Range	Description
ms	int	wait time in ms

Pause current thread (typically main thread) execution for *ms* milliseconds.

---

**Package [hw](#)**  
**Class [Time](#)****Public Method getHour**

```
int getHour()
```

**Description:**

Returns	Range	Description
int	0...23	the stored hour within a new created <i>Time</i> object

A new created *Time* object can be used to save the current time derived from the real-time clock. A new time is assigned to a *Time* object when the [IO.getTime\(\)](#) method is called. Once a *Time* object is created, the hour value at the objects creation time will be stored into the *Time* object for further use. The [getHour\(\)](#) method returns the hour as an integer value.

**Example**

```
Time saveCurrentTime = IO.getTime();  
Console.println("stored hour: " + saveCurrentTime.getHour());
```

**See also:**

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

## Package [hw](#)

### Class [Time](#)

#### Public Method [getMinute](#)

```
int getMinute()
```

##### Description:

Returns	Range	Description
int	0...59	the stored minute within a new created <i>Time</i> object

A new created *Time* object can be used to save the current time derived from the real-time clock. A new time is assigned to a *Time* object when the [IO.getTime\(\)](#) method is called. Once a *Time* object is created, the minute value at the objects creation time will be stored into the *Time* object for further use. The [getMinute\(\)](#) method returns the minute as an integer value.

##### Example

```
Time saveCurrentTime = IO.getTime();  
Console.println("stored minute: " + saveCurrentTime.getMinute());
```

##### See also:

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

## Package [hw](#)

### Class [Time](#)

#### Public Method [getSecond](#)

```
int getSecond()
```

##### Description:

Returns	Range	Description
int	0...59	the stored second within a new created <i>Time</i> object

A new created *Time* object can be used to save the current time derived from the real-time clock. A new time is assigned to a *Time* object when the [IO.getTime\(\)](#) method is called. Once a *Time* object is created, the second value at the objects creation time will be stored into the *Time* object for further use. The [getSecond\(\)](#) method returns the second as an integer value.

##### Example

```
Time saveCurrentTime = IO.getTime();  
Console.println("stored second: " + saveCurrentTime.getSecond());
```

**See also:**

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

**Package [hw](#)**  
**Class [Time](#)****Public Method [toString](#)**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

Returns	Description
String	hour, minute, and second within a single string

The [toString\(\)](#) method returns the hour, minute, and second in one string ("hour:minute:second").

**Example**

```
Time saveCurrentTime = IO.getTime();  
Console.println("stored time: " + saveCurrentTime.toString());
```

**See also:**

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

**Package [hw](#)**  
**Class [Time](#)****Public Method [setHour](#)**

```
void setHour(int hour)
```

**Description:**

Parameter	Range	Description
hour	0 ... 23	hour value of a time object

This method is used to set up the current hour of an *Time* object.

### Example

```
time.setHour(2);
```

### See also:

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

## Package [hw](#)

## Class [Time](#)

### Public Method **setMinute**

```
void setMinute(int minute)
```

#### Description:

Parameter	Range	Description
minute	0 ... 59	minute value of a time object

This method is used to set up the current minute of an *Time* object.

### Example

```
time.setMinute(2);
```

### See also:

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

## Package [hw](#)

## Class [Time](#)

### Public Method **setSecond**

```
void setSecond(int second)
```

#### Description:

Parameter	Rang	Description
-----------	------	-------------



	e	
second	0 ... 59	second value of a time object

This method is used to set up the current second of an *Time* object.

### Example

```
time.setSecond(2);
```

### See also:

[IO.getTime\(\)](#)  
[IO.setTime\(int, int, int\)](#)

---

## Package ilcd

Contains classes for providing access to the iLCD command set. Most of the methods in these classes are in close correspondance to the non-Java iLCD commands regarding command names and parameters.

Note that most methods are static and should be used without instanciating class objects.

## Interfaces

- [OnTouchListener](#)

## Classes

- [Attribute](#)
- [Common](#)
- [Control](#)
- [DeviceInfo](#)
- [Draw](#)
- [EEPROM](#)
- [Extra](#)
- [General](#)
- [Graphic](#)
- [GraphicInfo](#)
- [ILCDEEPROMException](#)
- [ILCDException](#)
- [ILCDExceptionFactory](#)
- [ILCDFileSystemException](#)
- [ILCDFlashException](#)
- [ILCDGeneralException](#)
- [ILCDGraphException](#)
- [ILCDMacroException](#)
- [ILCDMemoryException](#)
- [ILCDSDCException](#)
- [ILCDTouchException](#)
- [ILCDUnknownRuntimeException](#)

- [Memory](#)
  - [NetworkStatus](#)
  - [Position](#)
  - [Power](#)
  - [ProjectInfo](#)
  - [ScreenParameters](#)
  - [Size](#)
  - [Touch](#)
  - [TouchEvent](#)
  - [TouchEventDispatcher](#)
- 

## Package [ilcd](#)

### Class Attribute

extends [Object](#)

### Public Methods

- [setAdjustmentForGraphics](#)
- [setAlpha](#)
- [setBackground-color](#)
- [setBoldModeEnabled](#)
- [setBorderColor](#)
- [setBorderShadowColor](#)
- [setBrightnessAdjustment](#)
- [setContrastAdjustment](#)
- [setFillingColor](#)
- [setFillingGradient](#)
- [setFillingTile](#)
- [setFont](#)
- [setFontSpacing](#)
- [setForegroundColor](#)
- [setHueAdjustment](#)
- [setInverseModeEnabled](#)
- [setLineCapsStyle](#)
- [setLineEndingMode](#)
- [setLineStyle](#)
- [setLineThickness](#)
- [setRectangleCornerRadius](#)
- [setSaturationAdjustment](#)
- [setShadowOffset](#)
- [setSymbolFontEnabled](#)
- [setTransparentModeEnabled](#)
- [setUnderlineModeEnabled](#)
- [setUnderlinePosition](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)

- [notifyAll](#)

## Public Fields

- static final byte ADJUST\_BRIGHTNESS
- static final byte ADJUST\_CONTRAST
- static final byte ADJUST\_HUE
- static final byte ADJUST\_INVERT
- static final byte ADJUST\_SATURATION
- static final byte GRADIENT\_BUTTONIZED
- static final byte GRADIENT\_HORIZONTAL
- static final byte GRADIENT\_HORIZONTAL\_SYMMETRICAL
- static final byte GRADIENT\_VERTICAL
- static final byte GRADIENT\_VERTICAL\_SYMMETRICAL
- static final byte LINE\_CAP\_BUTT
- static final byte LINE\_CAP\_ROUND
- static final byte LINE\_CAP\_SQUARE
- static final byte LINE\_DASH\_DOTTED
- static final byte LINE\_DASHED\_2\_2
- static final byte LINE\_DASHED\_4\_4
- static final byte LINE\_DOTTED
- static final byte LINE\_END\_ADJUST
- static final byte LINE\_END\_ADJUST\_DOT
- static final byte LINE\_END\_PRESERVE
- static final byte LINE\_SOLID

## Package [ilcd](#)

### Class [Attribute](#)

#### Public Method [setAdjustmentForGraphics](#)

```
static void setAdjustmentForGraphics(int mode)
```

#### Description:

Parameter	Range	Description
mode	Bit 0 ... 4	adjustable mode (see below)

Sets the adjustments used for all graphics drawn subsequently. The value for any selected modification has to be set before the actual drawing method is called. Only the modifications set in the *mode* bitmask will be carried out:

mode
ADJUST_BRIGHTNESS (see <a href="#">Attribute.setBrightnessAdjustment(int)</a> )
ADJUST_CONTRAST (see <a href="#">Attribute.setContrastAdjustment(int)</a> )
ADJUST_HUE (see <a href="#">Attribute.setHueAdjustment(int)</a> )
ADJUST_SATURATION (see <a href="#">Attribute.setSaturationAdjustment(int)</a> )

ADJUST_INVERT
---------------

**Note**

- All adjustment modifications can be combined and carried out simultaneously with this method.
- The modifications are processed in the following order: invert - hue/saturation - brightness/contrast.
- After setting graphic adjustment, tile filling will also be adjusted accordingly (refer to [Attribute.setFillingTile\(int\)](#)).
- To restore the default settings, set the *mode* parameter to 0.

**Example**

```
Attribute.setAdjustmentForGraphics (Attribute.ADJUST_INVERT) ;
```

All future graphic drawing will have the color values inverted. To restore the default settings:

```
Attribute.setAdjustmentForGraphics (0) ;
```

**See also:**

[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.displayGraphicArea\(int, int, int, int, int\)](#)  
[Draw.fillDisplay\(\)](#)  
[Draw.fillDisplayArea\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)

**Package [ilcd](#)****Class [Attribute](#)****Public Method [setAlpha](#)**

```
static void setAlpha(int alpha)
```

**Description:**

Parameter	Range	Description
alpha	0 ... 255	opacity value

Sets an opacity value used for subsequent graphic drawing and filling. If set to 0, no drawing will be done, an *alpha* of 255 means full opacity.

**Note**

- The set alpha value is taken into account by all graphic ([Graphic.displayLocalGraphic\(int\)](#), [Graphic.displayGraphicArea\(int, int, int, int, int\)](#)) and filling ([Draw.fillDisplay\(\)](#)),

[Draw.fillDisplayArea\(int, int\)](#)) methods as well as shape drawing when filled ([Draw.drawRectangle\(int, int, int\)](#), [Draw.drawStyledCircle\(int, int\)](#)).

- The default value for *alpha* is 255. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Attribute.setAlpha(127);
```

All future drawing and filling will be done with half opacity.

### See also:

[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.displayGraphicArea\(int, int, int, int, int\)](#)  
[Draw.fillDisplay\(\)](#)  
[Draw.fillDisplayArea\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)

## Package **ilcd**

### Class **Attribute**

#### Public Method **setBackground-color**

```
static void setBackground-color(int colorValue)
```

#### Description:

Parameter	Range	Description
colorValue	0x000000 ... 0xFFFFFFFF	color value for the background color

Sets the current background color.

#### Note

- The default value for *colorValue* is 0xFFFFFFFF (white). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- If the transparent mode (see [Attribute.setTransparentModeEnabled\(boolean\)](#)) is switched off (set to false), the background of all text characters is drawn with the current background color.
- Using the [Draw.eraseDisplay\(\)](#) or the [Draw.eraseDisplayArea\(int, int\)](#) method erases the current display/area and fill it with the actual background color.

### Example

```
Attribute.setBackground-color(0x0000FF);
```

This example will set the background color to blue.

**See also:**[24-Bit Color Values](#)[Attribute.setForegroundColor\(int\)](#)[Attribute.setInverseMode\(boolean\)](#)[Attribute.setTransparentModeEnabled\(boolean\)](#)Package [ilcd](#)Class [Attribute](#)**Public Method [setBoldModeEnabled](#)**

```
static void setBoldModeEnabled(boolean enabled)
```

Package [ilcd](#)Class [Attribute](#)**Public Method [setBorderColor](#)**

```
static void setBorderColor(int colorValue)
```

**Description:**

Parameter	Range	Description
colorValue	0x000000 ... 0xFFFFFFFF	color value for the borders

Sets the color for borders.

**Note**

- The default value for *colorValue* is 0x000000 (black). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- Drawing Rectangles with the [Draw.drawRectangle\(int, int, int\)](#) method will make use of this border color.
- The shadows of borders have their own value and can be set via [Attribute.setBorderShadowColor\(int\)](#).

**Example**

```
Attribute.setBorderColor(0xFFFF00);
```

This example will set the border color to yellow.

**See also:**[24-Bit Color Values](#)[Attribute.setBorderShadowColor\(int\)](#)[Attribute.setBackgroundColor\(int\)](#)[Draw.drawRectangle\(int, int, int\)](#)**Package [ilcd](#)****Class [Attribute](#)****Public Method [setBorderShadowColor](#)**

```
static void setBorderShadowColor(int colorValue)
```

**Description:**

Parameter	Range	Description
colorValue	0x000000 ... 0xFFFFFFFF	color value for the shadow of borders

Sets the shadow color for rectangle borders.

**Note**

- The default value for *colorValue* is 0x7B7D7B (grey). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- Drawing Rectangles with the [Draw.drawRectangle\(int, int, int\)](#) method will make use of this border shadow color when shadow is activated.

**Example**

```
Attribute.setBorderShadowColor(0x333335);
```

This example will set border shadow color to a dark grey.

**See also:**[24-Bit Color Values](#)[Attribute.setBorderColor\(int\)](#)[Attribute.setBackgroundColor\(int\)](#)[Draw.drawRectangle\(int, int, int\)](#)

**Package [ilcd](#)****Class [Attribute](#)****Public Method [setBrightnessAdjustment](#)**

```
static void setBrightnessAdjustment(int value)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
value	-255 ... 255	value for brightness adjustment

Sets the *value* for brightness adjustment used by the methods [Draw.adjustDisplay\(int\)](#) and [Draw.adjustDisplayArea\(int, int, int\)](#) as well as graphic drawing methods when [Attribute.setAdjustmentForGraphics\(int\)](#) is used.

**Note**

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Attribute.setBrightnessAdjustment(-50);
```

Sets the value for brightness adjustments to -50.

**See also:**

[Draw.adjustDisplay\(int\)](#)  
[Draw.adjustDisplayArea\(int, int, int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Attribute.setContrastAdjustment\(int\)](#)  
[Attribute.setHueAdjustment\(int\)](#)  
[Attribute.setSaturationAdjustment\(int\)](#)

**Package [ilcd](#)****Class [Attribute](#)****Public Method [setContrastAdjustment](#)**

```
static void setContrastAdjustment(int value)
```



**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
value	-255 ... 255	value for contrast adjustment

Sets the *value* for contrast adjustment used by the methods [Draw.adjustDisplay\(int\)](#) and [Draw.adjustDisplayArea\(int, int, int\)](#) as well as graphic drawing methods when [Attribute.setAdjustmentForGraphics\(int\)](#) is used.

**Note**

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Attribute.setContrastAdjustment(100);
```

Sets the value for contrast adjustments to 100.

**See also:**

[Draw.adjustDisplay\(int\)](#)  
[Draw.adjustDisplayArea\(int, int, int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Attribute.setBrightnessAdjustment\(int\)](#)  
[Attribute.setHueAdjustment\(int\)](#)  
[Attribute.setSaturationAdjustment\(int\)](#)

**Package [ilcd](#)****Class [Attribute](#)****Public Method [setFillingColor](#)**

```
static void setFillingColor(int colorValue)
```

**Description:**

Parameter	Range	Description
colorValue	0x000000 ... 0xFFFFFFFF	color value for solid color filling

Sets the active fill pattern to the solid color *colorValue*. All subsequent filling will be done with this color.

**Note**

- The default fill pattern is a solid color with a *colorValue* of 0x000000 (black). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Attribute.setFillingColor(0x00FF00);
```

All future filling will be done with the solid color green.

**See also:**

[24-Bit Color Values](#)

[Draw.fillDisplay\(\)](#)

[Draw.fillDisplayArea\(int, int\)](#)

[Attribute.setFillingGradient\(int, int, int, int\)](#)

[Attribute.setFillingTile\(int\)](#)

[Draw.drawRectangle\(int, int, int\)](#)

[Draw.drawStyledCircle\(int, int\)](#)

**Package [ilcd](#)****Class [Attribute](#)****Public Method [setFillingGradient](#)**

```
static void setFillingGradient(int mode, int ramp, int fromColor, int toColor)
```

**Throws:**




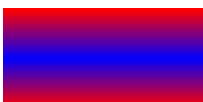

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
mode	0 ... 4	gradient mode (see below)
ramp	0 ... max[width, height]	pixels until <i>toColor</i> is reached (0 = maximum)
fromColor	0x000000 ... 0xFFFFFFFF	color value at beginning of gradient
toColor	0x000000 ... 0xFFFFFFFF	color value at end of gradient

Sets the active fill pattern to a gradient going from *fromColor* to *toColor*. All subsequent filling will be done with a gradient dependent on *mode*, which can be:

Mode	from red to blue	Description
------	------------------	-------------

GRADIENT_HORIZONTAL		ramp from left
GRADIENT_VERTICAL		ramp from top
GRADIENT_HORIZONTAL_SYMMETRICAL		ramp from left and right
GRADIENT_VERTICAL_SYMMETRICAL		ramp from top and bottom
GRADIENT_BUTTONIZED		ramp from all directions

### Note

- If *ramp* is set to 0 or the area to fill is smaller than *ramp* in the according direction, the gradient fades over the complete area disregarding the set *ramp* value.
- In any other case, the gradient starts from *fromColor* and gradually changes to *toColor* for *ramp* pixels. The rest of the according area is filled with the solid *toColor*.
- The *ramp* is scaled by the corresponding coordinate scale factor (refer to [Control.setRowCoordinatesScaling\(int, int\)](#) for horizontal and [Control.setColumnCoordinatesScaling\(int, int\)](#) for vertical gradients). For buttonized gradients, the smaller of those two scale factor is taken into account.
- The default fill pattern is a solid color (see [Attribute.setFillingColor\(int\)](#)) with a *color\_value* of 0x000000 (black). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Attribute.setFillingGradient(Attribute.GRAIDENT_BUTTONIZED, 20,
0xFFFFFFFF, 0x00FF00);
```

All future filling will be done with a gradient that fades from white to blue for 20 pixels in all directions. Filling an area with this pattern will give it a three-dimensional "buttonized" look.

### See also:

[Draw.fillDisplay\(\)](#)  
[Draw.fillDisplayArea\(int, int\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingTile\(int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)

## Package [ilcd](#)

### Class [Attribute](#)

#### Public Method [setFillingTile](#)

```
static void setFillingTile(int graphicIndex)
```

##### Throws:

- [ILCDEException](#)

```
static void setFillingTile(String graphicNameOrFilename)
```

##### Throws:

- [ILCDEException](#)

##### Description:

##### by index:

Parameter	Range	Description
graphicIndex	0 ... max. graphic index	index of tile graphic

##### by name or filename:

Parameter	Range	Description
graphicNameOrFilename	ASCII chars (0x01 .. 0xFF) or DOS filename (8.3 format)	name of the graphic or name and path of the graphics file

Sets the active fill pattern to tiling using the graphic specified by *graphicIndex* or *graphicNameOrFilename* as tiles. All subsequent filling will be done with this tile graphic.

##### Note

- When addressing by index, the graphic offset (see [Extra.setGraphicOffset\(int\)](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Extra.setGraphicNamePrefix\(String\)](#)) and suffix (refer to [Extra.setGraphicNameSuffix\(String\)](#)) are taken into account.
- Display and area filling can be done with scaled tiles (refer to [Control.setGraphicsScaling\(int\)](#)). Scaling will be ignored when drawing filled shapes (e.g. [Draw.drawRectangle\(int, int, int\)](#)) though.
- When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)) at the time filled drawing is done.
- The text/graphic orientation is ignored when filling with tiles (refer to [Control.setTextGraphicOrientation\(int\)](#)).

##### Example

```
Attribute.setFillingTile(1);
Attribute.setFillingTile("GRAPHIC");
Attribute.setFillingTile("DIR/FILE.RII");
```

Sets the graphic with index 1, graphic "GRAPHIC" from the on-board flash and graphic "FILE.RII" from the SD card's "DIR" folder as the fill pattern. Only the last successfully set tile graphic is used for future filling.

**See also:**

[Draw.fillDisplay\(\)](#)  
[Draw.fillDisplayArea\(int, int\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingGradient\(int, int, int, int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)

**Package [ilcd](#)**

**Class [Attribute](#)**

**Public Method [setFont](#)**

```
static void setFont(int fontIndex)
```

**Throws:**

- [ILCDEException](#)

```
static void setFont(String fontName)
```

**Throws:**

- [ILCDEException](#)

**Description:**

**by index:**

Parameter	Range	Description
fontIndex	0 ... max. available font	index of the font

**by name:**

Parameter	Range	Description
fontName	ASCII chars (0x01 .. 0xFF)	name of the font

Sets a font for the subsequent text outputs.

## Note

- The default value for *fontIndex* is 0, but can be modified on the "Fonts" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- The font with *fontIndex* of 0 is always available (even if the flash memory is blank).
- When addressing by index, the font offset (see [Extra.setFontOffset\(int\)](#)) is taken into account.
- When addressing by name, the font prefix (refer to [Extra.setFontNamePrefix\(String\)](#)) and suffix (refer to [Extra.setFontNameSuffix\(String\)](#)) are taken into account.
- Setting a font also resets the underline position to 0 and sets the font spacing (see [Attribute.setFontSpacing\(int, int\)](#)) and symbol font (see [Attribute.setSymbolFontEnabled\(boolean\)](#)) mode to the way defined via iLCD Manager XE.
- There is also a private ANSI extension which allows setting the font via an escape sequence (see [Private ANSI Extensions](#)).

## Example

```
Attribute.setFont(3);
Attribute.setFont("Times New Roman 18pt");
```

All subsequently outputted texts are drawn with the font with index 3.

## See also:

[Attribute.setUnderlinePosition\(int\)](#)  
[Attribute.setFontSpacing\(int, int\)](#)  
[Attribute.setSymbolFontEnabled\(boolean\)](#)  
[Extra.setFontOffset\(int\)](#)  
[Extra.setFontNameSuffix\(String\)](#)  
[Extra.setFontNamePrefix\(String\)](#)  
[Draw.writeText\(String\)](#)  
[Private ANSI Extensions](#)

## Package [ilcd](#) Class [Attribute](#)

### Public Method [setFontSpacing](#)

```
static void setFontSpacing(int xSpacing, int ySpacing)
```

#### Throws:

- [ILCDException](#)

#### Description:

Parameter	Range	Description
xSpacing	0 ... 15	horizontal spacing (number of blank pixels between consecutive characters)
ySpacing	0 ... 15	vertical spacing (number of blank pixels between consecutive lines)

This method allows to overwrite the default font spacing defined via iLCD Manager XE for the currently selected font.

### Note

- The default value for *xSpacing* and *ySpacing* is defined for every font on the "Fonts" page of iLCD Manager XE. They will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- The default font spacing is automatically scaled by font scale factor (refer to [Control.setFontsScaling\(int\)](#)). When setting the spacing via this method, the coordinate scale factors are taken into account (refer to [Control.setRowCoordinatesScaling\(int, int\)](#) for *xSpacing* and [Control.setColumnCoordinatesScaling\(int, int\)](#) for *ySpacing*).
- If the current font is set to symbol mode (see [Attribute.setSymbolFontEnabled\(boolean\)](#)) the setting of *xSpacing* and *ySpacing* is ignored and no blank space is used between consecutive characters and lines.

### Example

```
Attribute.setFontSpacing(15, 10);
```

This will overwrite the default font spacing to 15 pixels horizontally and to 10 pixels vertically.

### See also:

[Attribute.setFont\(int\)](#)  
[Attribute.setSymbolFontEnabled\(boolean\)](#)  
[Draw.writeText\(String\)](#)  
[Control.setFontsScaling\(int\)](#)  
[Control.setRowCoordinatesScaling\(int, int\)](#)  
[Control.setColumnCoordinatesScaling\(int, int\)](#)

## Package [ilcd](#)

### Class [Attribute](#)

#### Public Method [setForegroundColor](#)

```
static void setForegroundColor(int colorValue)
```

#### Description:

Parameter	Range	Description
colorValue	0x000000 ... 0xFFFFFFFF	color value for the foreground color

Sets the current foreground color all characters and lines are drawn with.

### Note

- The default value for *colorValue* is 0x000000 (black). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

- Monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.

### Example

```
Attribute.setForegroundColor(0xFF0000);
```

This example sets the foreground color to red.

### See also:

[24-Bit Color Values](#)

[Attribute.setBackgroundColor\(int\)](#)

[Attribute.setInverseMode\(boolean\)](#)

## Package [ilcd](#)

## Class [Attribute](#)

### Public Method [setHueAdjustment](#)

```
static void setHueAdjustment(int value)
```

#### Description:

Parameter	Range	Description
value	-255 ... 255	value for hue adjustment

Sets the *value* for hue adjustment used by the methods [Draw.adjustDisplay\(int\)](#) and [Draw.adjustDisplayArea\(int, int, int\)](#) as well as graphic drawing methods when [Attribute.setAdjustmentForGraphics\(int\)](#) is used.

#### Note

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Attribute.setHueAdjustment(-100);
```

Sets the value for hue adjustments to -100.

### See also:

[Draw.adjustDisplay\(int\)](#)

[Draw.adjustDisplayArea\(int, int, int\)](#)

[Attribute.setAdjustmentForGraphics\(int\)](#)

[Attribute.setBrightnessAdjustment\(int\)](#)



[Attribute.setContrastAdjustment\(int\)](#)  
[Attribute.setSaturationAdjustment\(int\)](#)

Package [ilcd](#)

Class [Attribute](#)

### **Public Method setInverseModeEnabled**

```
static void setInverseModeEnabled(boolean enabled)
```

#### **Description:**

Parameter	Range	Description
enabled	false ... true	determines whether inverse mode is on (true) or off (false)

Sets inverse mode on or off. The inverse mode swaps foreground and background colors.

#### **Note**

- All drawing, text and monochrome graphic methods take the inverse mode into account (for example [Draw.writeText\(String\)](#), [Draw.drawLine\(int, int\)](#), [Draw.eraseDisplayArea\(int, int\)](#) etc.)
- The default value for *enabled* is false. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

#### **Example**

```
Attribute.setInverseModeEnabled(true);
```

The foreground and background colors are swapped.

#### **See also:**

[Attribute.setBackgroundColor\(int\)](#)  
[Attribute.setForegroundColor\(int\)](#)  
[Draw.writeText\(String\)](#)  
[Draw.drawLine\(int, int\)](#)  
[Draw.eraseDisplayArea\(int, int\)](#)

Package [ilcd](#)

Class [Attribute](#)

### **Public Method setLineCapsStyle**

```
static void setLineCapsStyle(int style)
```




**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
style	0 ... 2	style of the line caps

Sets the *style* of the line caps according to the following styles (the red dot represents the specified coordinates for the line ending):

Style	Description
LINE_CAP_ROUND	
LINE_CAP_BUTT	
LINE_CAP_SQUARE	

**Note**

- The default value for *style* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Attribute.setLineCapsStyle (Attribute.LINE_CAP_BUTT) ;
```

After issuing this method, all subsequently drawn lines will have rectangular line caps and will start and end precisely at the specified coordinates (no overlapping).

**See also:**

[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Attribute.setLineThickness\(int\)](#)  
[Attribute.setLineEndingMode\(int\)](#)

## Package [ilcd](#)

### Class [Attribute](#)

#### Public Method [setLineEndingMode](#)

```
static void setLineEndingMode(int mode)
```







#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
mode	0 ... 2	mode of the line endings

Sets the line ending mode according to the following values for *mode* (the red dot represents the specified coordinates for the line ending resp. rectangle corner):

Mode	Single Line	Connected Lines	Description
LINE_END_PRESERVE			line is drawn up to the last complete dash or dot
LINE_END_ADJUST			line is draw exactly to the specified end point
LINE_END_ADJUST_DOT			line is drawn to the last multiple of line thickness

#### Note

- This value is only taken into account when drawing dashed or dotted lines (see [Attribute.setLineStyle\(int\)](#)).
- The default value for *mode* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

#### Example

```
Attribute.setLineEndingMode(Attribute.LINE_END_ADJUST_DOT);
```

This example will set the line ending mode to *mode* 1 (e.g. for drawing dashed rectangles without unexpectedly cropped dashes).

#### See also:

[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Attribute.setLineCapsStyle\(int\)](#)  
[Attribute.setLineThickness\(int\)](#)  
[Attribute.setLineStyle\(int\)](#)

Package [ilcd](#)

## Class [Attribute](#)

### Public Method [setLineStyle](#)

```
static void setLineStyle(int style)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
style	0x01 ... 0xFF	style of the line

This method allows a definition for a line style used by [Draw.drawLine\(int, int\)](#) and [Draw.drawRectangle\(int, int, int\)](#). The parameter *style* represents a bit mask where a 1 represents a pixel to be written and a 0 a pixel to be omitted. The following table shows some examples for line styles:

style
LINE_SOLID
LINE_DOTTED
LINE_DASHED_4_4 (4 pixel black, 4 pixel white)
LINE_DASHED_2_2 (2 pixel black, 2 pixel white)
LINE_DASH_DOTTED

#### Note

- The default value for *style* is LINE\_SOLID (solid line). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- The least significant bit of style describes the first pixel painted when drawing a line from left to right.
- When line thickness is greater than 1 (refer to [Attribute.setLineThickness\(int\)](#)), every bit represents a dot with a diameter of line thickness. The line ending mode (refer to [Attribute.setLineEndingMode\(int\)](#)) is then taken into account as well.
- The current line style is used when a rectangle is painted allowing the drawing of various styles of rectangles as well. When using a line style other than LINE\_SOLID, the two rectangular line caps styles lead to the same result (refer to [Attribute.setLineCapsStyle\(int\)](#)).

#### Example

```
Control.setLineStyle(Attribute.LINE_DASHED_2_2);
Attribute.setLineStyle(Attribute.LINE_DASHED_2_2);
```

These two methods are equivalent and will set the line style to a dashed line with 2 pixels black and 2 pixels white.

**See also:**

[Attribute.setLineThickness\(int\)](#)  
[Attribute.setLineEndingMode\(int\)](#)  
[Attribute.setLineCapsStyle\(int\)](#)  
[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)

---

**Package [ilcd](#)****Class [Attribute](#)****Public Method [setLineThickness](#)**

```
static void setLineThickness(int thickness)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>thickness</code>	1 ... 64	thickness of a line

Sets the current line thickness.

**Note**

- The default value for `thickness` is 1. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Attribute.setLineThickness(5);
```

This example will set the line thickness to 5.

**See also:**

[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Draw.drawEllipse\(int, int, int\)](#)  
[Attribute.setLineCapsStyle\(int\)](#)  
[Attribute.setLineEndingMode\(int\)](#)

Package [ilcd](#)

Class [Attribute](#)

### Public Method [setRectangleCornerRadius](#)

```
static void setRectangleCornerRadius(int radius)
```

Throws:

- [ILCDEException](#)

Description:

Parameter	Range	Description
radius	1 ... max[the smaller value of the rectangle (height, width) divided by 2]	radius of rectangle corners

Sets the corner radius for subsequently drawn rectangles.

Note

- The rectangle corner radius is only taken into account when the rounded corners flag (bit 0 of the *mode* parameter in [Draw.drawRectangle\(int, int, int\)](#)) is set.
- For *radius*, the column coordinates scale factor is used (see [Control.setColumnCoordinatesScaling\(int, int\)](#)).
- The default value for *radius* is 5. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

Example

```
Attribute.setRectangleCornerRadius(25);
```

All subsequently rectangle drawing will be done with a corner radius of 25 pixels if the rounded corners flag is set.

See also:

[Draw.drawRectangle\(int, int, int\)](#)  
[Attribute.setShadowOffset\(int, int\)](#)  
[Attribute.setLineThickness\(int\)](#)

Package [ilcd](#)

Class [Attribute](#)

### Public Method [setSaturationAdjustment](#)

```
static void setSaturationAdjustment(int value)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
value	-255 ... 255	value for saturation adjustment

Sets the *value* for saturation adjustment used by the methods [Draw.adjustDisplay\(int\)](#) and [Draw.adjustDisplayArea\(int, int, int\)](#) as well as graphic drawing methods when [Attribute.setAdjustmentForGraphics\(int\)](#) is used.

**Note**

- The default *value* is 0, meaning no modification. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Attribute.setSaturationAdjustment(50);
```

Sets the value for saturation adjustments to 50.

**See also:**

[Draw.adjustDisplay\(int\)](#)  
[Draw.adjustDisplayArea\(int, int, int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Attribute.setBrightnessAdjustment\(int\)](#)  
[Attribute.setContrastAdjustment\(int\)](#)  
[Attribute.setHueAdjustment\(int\)](#)

Package [ilcd](#)

**Class [Attribute](#)****Public Method [setShadowOffset](#)**

```
static void setShadowOffset(int xOffset, int yOffset)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
xOffset	-100 ...	horizontal shadow offset

	100	
yOffset	-100 ... 100	vertical shadow offset

Sets the shadow offset for the drop shadows of subsequent shape drawing (refer to [Draw.drawRectangle\(int, int, int\)](#), [Draw.drawStyledCircle\(int, int\)](#), [Draw.drawEllipse\(int, int, int\)](#)).

#### Note

- If the origin of a shadow shape has negative coordinates, the drop shadow is disabled in the corresponding shape drawing function.
- The default values for *xOffset* and *yOffset* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- If any offset is 0, the shadow of shapes will be drawn with the historic default: half of the line thickness (refer to [Attribute.setLineThickness\(int\)](#)) if the frame is drawn and 1 pixel if not.

#### Example

```
Attribute.setShadowOffset(20, -10);
```

All subsequent shapes are drawn with a drop shadow 20 pixels to the right (positive x) and 10 pixels above (negative y) the original shape.

#### See also:

[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Draw.drawEllipse\(int, int, int\)](#)  
[Attribute.setBorderShadowColor\(int\)](#)

## Package [ilcd](#)

### Class [Attribute](#)

#### Public Method [setSymbolFontEnabled](#)

```
static void setSymbolFontEnabled(boolean enabled)
```

#### Description:

Parameter	Range	Description
enabled	false ... true	determines whether the currently selected font is a symbol font or not

Sets the current font to a symbol font, meaning that there will be no blank pixels between consecutive characters and lines. This is especially useful when using fonts which contain border characters.

#### Note

- Selecting a font which was defined as a symbol font via iLCD Manager XE does not require the issue of this method.



- If *enabled* is false, symbol mode is turned off and the default font spacing is used or when previously overwritten via the [Attribute.setFontSpacing\(int, int\)](#), the last selected font spacing is used.

### Example

```
Attribute.setSymbolFontEnabled(true);
```

Causes every subsequent text to be outputted with no blank pixels similar to a symbol font, until the method is repeated and the parameter *enabled* is set to false again.

### See also:

[Attribute.setFont\(int\)](#)  
[Draw.writeText\(String\)](#)

## Package [ilcd](#)

## Class [Attribute](#)

### Public Method [setTransparentModeEnabled](#)

```
static void setTransparentModeEnabled(boolean enabled)
```

### Description:

Parameter	Range	Description
enabled	false ... true	determines whether transparent mode is on (true) or off (false)

Activates or deactivates transparent mode for text outputs.

### Note

- Normally, drawing text will fill the background of the character with the current background color. With transparent mode on, it is possible to output text over an existing graphic.
- This method will not affect graphics with transparencies, these are always displayed transparent.
- The default value for *enabled* is false (transparent mode off). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Attribute.setTransparentModeEnabled(true);
```

From now on, text output will be transparent.

### See also:

[Attribute.setFont\(int\)](#)  
[Draw.writeText\(String\)](#)

Package [ilcd](#)

Class [Attribute](#)

### Public Method `setUnderlineModeEnabled`

```
static void setUnderlineModeEnabled(boolean enabled)
```

Throws:

- [ILCDEException](#)

Description:

Parameter	Range	Description
<code>enabled</code>	false ... true	text is outputted underlined (true) or not (false)

Activates or deactivates an underline feature for any subsequent text outputs.

Note

- By default the line is drawn at the lowest pixel position, which is within the character area. The vertical position of the underlining can be set via the [Attribute.setUnderlinePosition\(int\)](#) method.
- The default value for `enabled` is false. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

Example

```
Attribute.setUnderlineModeEnabled(true);
```

Causes every subsequent text to be underlined, until the method is repeated and the parameter `enabled` is set to false again.

See also:

[Attribute.setFont\(int\)](#)

[Attribute.setUnderlinePosition\(int\)](#)

[Draw.writeText\(String\)](#)

Package [ilcd](#)

Class [Attribute](#)

### Public Method `setUnderlinePosition`

```
static void setUnderlinePosition(int position)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>position</code>	-32 ... 31	underline position

Sets the vertical position for the underline.

**Note**

- Underline Position is only relevant when Underline Mode is activated (see [Attribute.setUnderlineModeEnabled\(boolean\)](#)).
- Setting a font (see [Attribute.setFont\(int\)](#)) causes `position` automatically to be reset to 0.
- The default value for `position` is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- If the current vertical font spacing is 1 (see [Attribute.setFontSpacing\(int, int\)](#)), any value greater than 0 for `position` causes the underline not drawn anymore as it would overwrite parts of next line.
- Using a negative value for `position` will cause the line to be drawn within the character area (thus crossing out instead of underlining the character).
- If `position` would cause overwriting parts of the previous line (depending on the font height), underlining is not carried out anymore.

**Example**

```
Attribute.setUnderlinePosition(2);
```

This example sets the underline position to 2 pixels below the text.

**See also:**

[Attribute.setFont\(int\)](#)  
[Attribute.setFontSpacing\(int, int\)](#)  
[Draw.writeText\(String\)](#)

**Package [ilcd](#)****Class Common**

extends [Object](#)

The *Common* class defines error codes returned by the firmware. These codes are used by the class *ILCDEExceptionFactory* for creating corresponding Java exceptions. The according description text is defined in the class *ILCDEException*. Also an instance of the class *EventManager* is created.

## Note

- It's not possible to instantiate this class.

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final int E\_COMMAND
- static final int E\_EEP\_ERASE
- static final int E\_EEP\_MAXSIZE
- static final int E\_FF\_DENIED
- static final int E\_FF\_DISKFULL
- static final int E\_FF\_EXIST
- static final int E\_FF\_INVALID\_DRIVE
- static final int E\_FF\_INVALID\_NAME
- static final int E\_FF\_INVALID\_OBJECT
- static final int E\_FF\_MKFS\_ABORTED
- static final int E\_FF\_NO\_FILE
- static final int E\_FF\_NO\_FILESYSTEM
- static final int E\_FF\_NO\_PATH
- static final int E\_FF\_NOT\_ENABLED
- static final int E\_FF\_NOT\_READY
- static final int E\_FF\_READ1
- static final int E\_FF\_READ2
- static final int E\_FF\_READEND
- static final int E\_FF\_READWR
- static final int E\_FF\_RW\_ERROR
- static final int E\_FF\_RW\_ERROR1
- static final int E\_FF\_RW\_ERROR10
- static final int E\_FF\_RW\_ERROR11
- static final int E\_FF\_RW\_ERROR12
- static final int E\_FF\_RW\_ERROR13
- static final int E\_FF\_RW\_ERROR14
- static final int E\_FF\_RW\_ERROR15
- static final int E\_FF\_RW\_ERROR16
- static final int E\_FF\_RW\_ERROR17
- static final int E\_FF\_RW\_ERROR18
- static final int E\_FF\_RW\_ERROR19
- static final int E\_FF\_RW\_ERROR2
- static final int E\_FF\_RW\_ERROR20
- static final int E\_FF\_RW\_ERROR21
- static final int E\_FF\_RW\_ERROR22
- static final int E\_FF\_RW\_ERROR23
- static final int E\_FF\_RW\_ERROR24
- static final int E\_FF\_RW\_ERROR25
- static final int E\_FF\_RW\_ERROR26
- static final int E\_FF\_RW\_ERROR27
- static final int E\_FF\_RW\_ERROR28
- static final int E\_FF\_RW\_ERROR29

- static final int E\_FF\_RW\_ERROR3
- static final int E\_FF\_RW\_ERROR30
- static final int E\_FF\_RW\_ERROR31
- static final int E\_FF\_RW\_ERROR4
- static final int E\_FF\_RW\_ERROR5
- static final int E\_FF\_RW\_ERROR6
- static final int E\_FF\_RW\_ERROR7
- static final int E\_FF\_RW\_ERROR8
- static final int E\_FF\_RW\_ERROR9
- static final int E\_FF\_WRITE\_PROTECTED
- static final int E\_FF\_WRITE1
- static final int E\_FF\_WRITE2
- static final int E\_FF\_WRITERD
- static final int E\_FIXEDTEXT
- static final int E\_FLSH\_APPLADDR
- static final int E\_FLSH\_BLOCK
- static final int E\_FLSH\_SETSECTOR
- static final int E\_GR\_ANIMASS
- static final int E\_GR\_ANIMIDX
- static final int E\_GR\_COLORMODE
- static final int E\_GR\_FRAMEIDX
- static final int E\_GR\_INVTYPE
- static final int E\_GR\_NO\_CROP
- static final int E\_GR\_NOTFOUND
- static final int E\_GR\_ORIENTATION
- static final int E\_H\_OUT\_OF\_RANGE
- static final int E\_IO\_LEDCOM
- static final int E\_IO\_TIMESET
- static final int E\_JAVA\_INTERNAL\_ERROR
- static final int E\_JAVA\_INVALID\_ADDR
- static final int E\_JAVA\_INVALID\_BYTECODE
- static final int E\_JV\_ARR\_OUT\_OF\_BOUNDS
- static final int E\_LCD\_NOFONT
- static final int E\_MAC\_DEPTH
- static final int E\_MAC\_ERROR
- static final int E\_MAC\_NOT\_FOUND
- static final int E\_MCI\_BLEN
- static final int E\_MCI\_BLOCKS
- static final int E\_MCI\_BUSY1
- static final int E\_MCI\_BUSY2
- static final int E\_MCI\_CMD55
- static final int E\_MCI\_CMDT01
- static final int E\_MCI\_CMDT02
- static final int E\_MCI\_CRC
- static final int E\_MCI\_CSD
- static final int E\_MCI\_IDENT
- static final int E\_MCI\_INIT
- static final int E\_MCI\_MMCRCA
- static final int E\_MCI\_NOCARD
- static final int E\_MCI\_NREADY
- static final int E\_MCI\_PARAM
- static final int E\_MCI\_PROTECT
- static final int E\_MCI\_RCA
- static final int E\_MCI\_RDSTAT
- static final int E\_MCI\_READ
- static final int E\_MCI\_READST
- static final int E\_MCI\_READTRAN
- static final int E\_MCI\_SELECT

- static final int E\_MCI\_STAT1
- static final int E\_MCI\_STAT2
- static final int E\_MCI\_WIDE
- static final int E\_MCI\_WRITE
- static final int E\_MCI\_WRITEST
- static final int E\_MCI\_WRITETRAN
- static final int E\_MEM\_BLOCKED
- static final int E\_MEM\_CURSORNF
- static final int E\_MEM\_MAXCURSOR
- static final int E\_MEM\_OUT\_OF\_RANGE
- static final int E\_MEM\_POS\_EMPTY
- static final int E\_MEM\_UNUSED
- static final int E\_NOERR
- static final int E\_NOT\_BOOLEAN
- static final int E\_PARNOT\_ZERO
- static final int E\_PF\_INVALID\_CNTRL
- static final int E\_PF\_INVALID\_DATA
- static final int E\_PF\_N\_BLOCK\_ERASE
- static final int E\_PF\_N\_INDEX\_ERASE
- static final int E\_PF\_N\_INDEX\_WRITE
- static final int E\_PF\_N\_WRITE
- static final int E\_PF\_TOOBIG
- static final int E\_PF\_VALIDATE
- static final int E\_RII\_GRAPHIC
- static final int E\_RII\_MD5
- static final int E\_SCALEERR
- static final int E\_SCROLL\_INVALID
- static final int E\_SDC\_DIREND
- static final int E\_SDC\_EOF
- static final int E\_SDC\_NODISK
- static final int E\_SDC\_NOPEN
- static final int E\_SDC\_OPEN
- static final int E\_SDC\_PARAM
- static final int E\_SDC\_READ\_FAIL
- static final int E\_SETUP2
- static final int E\_SIM\_NOT\_SUPPORTED
- static final int E\_STR\_TOO\_LONG
- static final int E\_TCH\_ACTIVE
- static final int E\_TCH\_MAXFIELD
- static final int E\_TCH\_NOTCALIB
- static final int E\_TCH\_OUT\_OF\_RANGE
- static final int E\_TCH\_TEXT\_NF
- static final int E\_VAL\_OUT\_OF\_RANGE
- static final int E\_VIEWP\_INVALID
- static final int E\_VIEWPORT
- static final int E\_W\_OUT\_OF\_RANGE
- static final int E\_X\_OUT\_OF\_RANGE
- static final int E\_Y\_OUT\_OF\_RANGE

---

## Package [ilcd](#)

### Class Control

extends [Object](#)

This class provides methods for

- get and set cursor position
- handle screen orientation
- define and control viewports
- provide access to information about graphics, text extend, display size etc.

Alignment and scaling are available for graphics and text messages. The extent methods provide the possibility to get the size of a text message. To change the view of a specific area on the screen (e.g. rotate a graphic) the viewport related methods are available. It's also possible to change the complete screen orientation by 90°.

#### Note

- It's not possible to instantiate this class. The methods in this class are static.

#### Public Methods

- [copyViewport](#)
- [defineViewport](#)
- [getBacklightIntensity](#)
- [getBacklightIntensityHighRes](#)
- [getBacklightMode](#)
- [getCursorPosition](#)
- [getDisplaySize](#)
- [getGraphicInfo](#)
- [getLCDContrast](#)
- [getLCDGammaValue](#)
- [getTextExtent](#)
- [getTextMessageExtent](#)
- [getUnicodeTextExtent](#)
- [incrementDecrementColumnAddress](#)
- [incrementDecrementRowAddress](#)
- [isFixedLCDContrastGamma](#)
- [selectViewport](#)
- [setAnsiEnabled](#)
- [setAutoLinefeedEnabled](#)
- [setBacklightBlinkFrequency](#)
- [setBacklightIntensity](#)
- [setBacklightIntensityHighRes](#)
- [setBacklightMode](#)
- [setColumnAddress](#)
- [setColumnCoordinatesScaling](#)
- [setCursorPosition](#)
- [setFontScaling](#)
- [setGraphicAlignment](#)
- [setGraphicsScaling](#)
- [setLCDContrast](#)
- [setLCDGammaValue](#)
- [setLineStyle](#)
- [setRelativeCursorPosition](#)
- [setRowAddress](#)
- [setRowCoordinatesScaling](#)
- [setScreenOrientation](#)
- [setTabSpacing](#)
- [setTextAlignment](#)
- [setTextGraphicOrientation](#)

- [setWrapMode](#)
- [turnDisplayOnOff](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final byte BACKLIGHT\_BLINK
- static final byte BACKLIGHT\_FADE\_OUT
- static final byte BACKLIGHT\_OFF
- static final byte BACKLIGHT\_ON
- static final byte DISPLAY\_OFF
- static final byte DISPLAY\_ON\_DELAYED
- static final byte DISPLAY\_ON\_IMMEDIATELY
- static final byte GRAPHIC\_BOTTOM\_JUSTIFY\_
- static final byte GRAPHIC\_CENTER\_HORIZONTALLY
- static final byte GRAPHIC\_CENTER\_VERTICALLY
- static final byte GRAPHIC\_RIGHT\_JUSTIFY\_
- static final byte GRAPHIC\_TURN\_ALIGNMENT\_ON
- static final byte ORIENTATION\_LANDSCAPE
- static final byte ORIENTATION\_LANDSCAPE\_UPSIDE\_DOWN
- static final byte ORIENTATION\_PORTRAIT
- static final byte ORIENTATION\_PORTRAIT\_UPSIDE\_DOWN
- static final byte TEXT\_ADD\_HORIZONTAL\_SPACE\_OF\_BORDER
- static final byte TEXT\_ADD\_VERTICAL\_SPACE\_OF\_BORDER
- static final byte TEXT\_BOTTOM\_JUSTIFY
- static final byte TEXT\_CENTER\_HORIZONTALLY
- static final byte TEXT\_CENTER\_VERTICALLY
- static final byte TEXT\_DO\_NOT\_WORD\_WRAP
- static final byte TEXT\_RIGHT\_JUSTIFY
- static final byte TEXT\_TURN\_ALIGNMENT\_ON

---

## Viewport Related Methods

### Concept of Viewports

Viewports are user-defined screen areas which have their own coordinates, width, height and orientation. Additional to the main viewport 0 (entire main screen) for all screens, you may define up to 8 viewports (1-8) in a project. They may overlap each other, but consider that they don't have independent memory areas. Therefore, a viewport may overwrite the content of an overlapping other one without saving the underlying area.

All methods (except the Class [Memory](#)) executed within a viewport refer to the viewport's size and cursor position. So, for example, [getDisplaySize\(\)](#) returns the height and width of the current viewport, not the entire screen. Text wrapping and scrolling as well as cropping of graphics is also done in the selected viewport's area.



All defined viewports have their own set of attributes (see Class [Attribute](#)). So when switching from viewport 1 to viewport 2 and outputting text in both of them, it shows up on the cursor position and with the colors and attributes of the respective viewport.

### Note

- The methods [resetAll\(\)](#) and [resetAllAndShowStartupGraphic\(\)](#) delete all defined viewports

### Example

```
General.resetAll();

Control.setCursorPosition(30,130);           // set cursor position to
30/130
Control.defineViewport(1, 0, 65, 50);       // define viewport 1 with size
of 65x50

Control.setCursorPosition(120,130);        // set cursor position to
120/130
Control.defineViewport(2, 1, 65, 70);      // define viewport 2 with size
of 65x70

Control.setCursorPosition(280,130);        // set cursor position to
280/130
Control.defineViewport(3, 3, 78, 72);      // define viewport 3 with size
of 78x72

Control.selectViewport(1);                 // select viewport 1
Attribute.setBackgroundColor(0xFF0000);    // set background color to red
Draw.eraseDisplay();                      // fill viewport with
background color
Draw.writeText("Viewport 1: Orientation = 0\0");

Control.selectViewport(2);                 // select viewport 2
Attribute.setBackgroundColor(0x00FF00);    // set background color to
green
Draw.eraseDisplay();                      // fill viewport with
background color
Draw.writeText("Viewport 2: Orientation = 1\0");

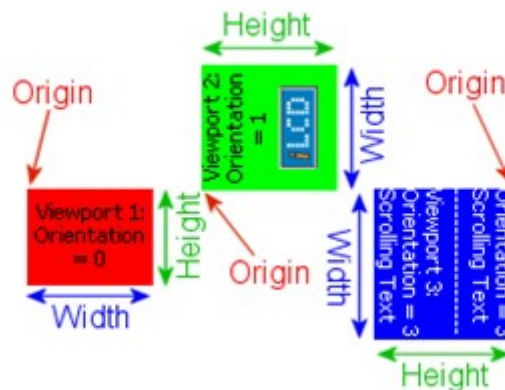
/* Cursor Position now relative to viewport Origin and Orientation: */
Control.setRelativeCursorPosition(11, 40);
Graphic.displayLocalGraphic(2);

Control.selectViewport(3);                 // select viewport 3
Attribute.setBackgroundColor(0x0000FF);    // set background color to blue
Attribute.setForegroundColor(0xFFFFFFFF);  // set foreground color to
white
Draw.eraseDisplay();                      // fill viewport with
background color
```

```

Draw.writeText("Viewport 3\r"
    + "Orientation = 3\r"
    + "Scrolling Text\r"
    + "-----\r"
    + "Viewport 3:\r"
    + "Orientation = 3\r"
    + "Scrolling Text");

```



Not supported by: DPC3020, DPC2060, DPC10xx

Find following methods in this chapter as well as in the category "LCD Control..." when using the parameter completion feature of iLCD Manager XE:

- [defineViewport](#)
- [selectViewport](#)
- [copyViewport](#)

## Package [ilcd](#)

### Class [Control](#)

#### **Public Method copyViewport**

```

static void copyViewport(int screen, int viewport, boolean copyContent,
int posX, int posY)

```

#### Throws:

- [ILCDException](#)

#### Description:

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to copied to (M = main screen)

<code>viewport</code>	1 ... 8	index of the new viewport
<code>copyContent</code>	false ... true	copy display content (true) or just the parameters (false)
<code>posX</code>	0 ... display width - 1	x coordinate for the new viewport's origin
<code>posY</code>	0 ... display height - 1	y coordinate for the new viewport's origin

Creates a new viewport with `viewport` as index at the specified position (`posX` and `posY`) of the selected `screen` and assigns all attributes of the currently selected viewport to it.

### Note

- Creating a new viewport with this method will also update the target screen's cursor position to the specified origin coordinates.
- The new viewport will inherit the dimensions (width and height) of the source viewport in any case.
- Depending on the boolean value `copyContent` the new viewport will have the same content (every pixel) as the currently active viewport. If the currently active viewport contains a running animation, only a screenshot of the currently shown frame is copied.
- `screen` can also be set to the currently active draw screen, allowing to copy a viewport to a different position on the same screen. Even if the target screen was never selected as draw screen before, it is possible to select it as view screen after copying a viewport to it.

### Example

```
Control.copyViewport(0, 2, true, 45, 60);
```

On screen #0, a new viewport with an index of 2 and the same dimensions as the previously selected viewport is created. It has the same content as the selected viewport, its origin is at the coordinates x = 45 and y = 60. The cursor position of screen #0 is also set to this position.

### See also:

[Viewports Related Methods](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method `defineViewport`

```
static void defineViewport(int viewport, int orientation, int width, int height)
```

### Throws:

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
viewport	1 ... 8	index of the viewport
orientation	0 ... 3	orientation of the viewport
width	0 ... display width	width of the viewport
height	0 ... display height	height of the viewport

Defines a viewport with origin at the current cursor position.

**Note**

- You may define and use up to 8 viewports.
- A viewport is always defined on the currently active draw screen (see [Memory.setDrawScreen\(int\)](#)).
- The viewport's orientation is interpreted relative to the screen orientation (see [Control.setScreenOrientation\(int\)](#)).
- Any other orientation than 0 will rotate the viewport around the cursor position and accordingly swap *width* and *height* (see [Viewports Related Methods](#)).
- All defined Viewports are automatically removed on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Control.defineViewport(1, 0, 30, 15);
```

This example defines a viewport with index 1, the default orientation and a size of 30x15 pixels.

**See also:**

[Viewports Related Methods](#)  
[Control.getCursorPosition\(\)](#)  
[Control.setScreenOrientation\(int\)](#)

**Package [ilcd](#)****Class [Control](#)****Public Method [getBacklightIntensity](#)**

```
static int getBacklightIntensity()
```

**Throws:**

- [ILCDEException](#)

**Description:**

Returns	Rang	Description
---------	------	-------------

	e	
intensity	0 ... 15	numerical value for the intensity

Delivers the current setting of the backlight intensity (even if the backlight is turned off).

### Example

```
int intensity = Control.getBacklightIntensity();
```

### See also:

[Control.setBacklightIntensity\(int\)](#)  
[Control.setBacklightMode\(int\)](#)  
[Control.getBacklightIntensityHighRes\(\)](#)

Package [ilcd](#)

Class [Control](#)

### Public Method [getBacklightIntensityHighRes](#)

```
static int getBacklightIntensityHighRes ()
```

### Throws:

- [ILCDEException](#)

### Description:

Returns	Range	Description
intensity	0 ... 255	numerical value for the intensity

Delivers the current setting of the backlight intensity in high-res steps from 0 to 255 (even if the backlight is turned off).

### Example

```
int intensity = Control.getBacklightIntensityHighRes();
```

### See also:

[Control.setBacklightIntensityHighRes\(int\)](#)  
[Control.setBacklightMode\(int\)](#)  
[Control.getBacklightIntensity\(\)](#)

Package [ilcd](#)  
 Class [Control](#)

**Public Method [getBacklightMode](#)**

```
static int getBacklightMode()
```

**Description:**

Returns	Range	Description
mode	0 ... 3	backlight mode

The method returns the current adjusted backlight mode.

mode
BACKLIGHT_OFF
BACKLIGHT_ON
BACKLIGHT_BLINK
BACKLIGHT_FADE_OUT

**Example**

```
Control.getBacklightMode() == Control.BACKLIGHT_ON
```

If this expression evaluates to true, the backlight is on and no other mode is set.

**See also:**

[Control.setBacklightMode\(int\)](#)  
[Control.setBacklightIntensity\(int\)](#)

Package [ilcd](#)  
 Class [Control](#)

**Public Method [getCursorPosition](#)**

```
static Position getCursorPosition()
```

**Description:**

Returns	Description
static <a href="#">Position</a>	cursor position

This method is used to obtain the current cursor position within the used display boundary. The *Position* return type delivers the *x* (horizontal axis) and *y* (vertical axis) values of the cursor position.

### Note

- The class variables will be overwritten with a new cursor position when this function is called again.
- The reference is only returned to get the value of a class variable with a single call, see below.

### Example

```
Position actualPosition = Control.getCursorPosition();
int x = actualPosition.getX();
int y = actualPosition.getY();
```

Alternatively a value can be obtained with a single call:

```
int x = Control.getCursorPosition().getX();
int y = Control.getCursorPosition().getY();
```

### See also:

[Position](#)

[Control.setCursorPosition\(int, int\)](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method `getDisplaySize`

```
static Size getDisplaySize ()
```

#### Description:

Returns	Description
static <a href="#">Size</a>	the size of the display

The `getDisplaySize()` method is used to obtain the size of the display. The *Size* return type delivers the *height* and *width* values of the display.

### Note

- Please note that `Control.Size` is a singleton object and cannot be instantiated again.
- Be careful when storing the reference to `Control.Size` because this class exists only once and will be overwritten when one of the following methods are called: [Control.getDisplaySize\(\)](#), [Control.getTextExtent\(String\)](#), [Control.getTextMessageExtent\(int\)](#), [Control.getTextMessageExtent\(String\)](#), [Control.getUnicodeTextExtent\(String\)](#).

## Example

```
Size displaySize = Control.getDisplaySize();
int height = displaySize.getHeight();
int width = displaySize.getWidth();
```

Alternatively a value can be obtained with a single call:

```
int height = Control.getDisplaySize().getHeight();
int width = Control.getDisplaySize().getWidth();
```

## See also:

[Size](#)

[Control.getDisplaySize\(\)](#)

[Control.getTextExtent\(String\)](#)

[Control.getTextMessageExtent\(int\)](#)

[Control.getTextMessageExtent\(String\)](#)

[Control.getUnicodeTextExtent\(String\)](#)

Package [ilcd](#)

## Class [Control](#)

### Public Method [getGraphicInfo](#)

```
static GraphicInfo getGraphicInfo(int graphicIndex)
```

#### Throws:

- [ILCDEException](#)

```
static GraphicInfo getGraphicInfo(String graphicNameOrFilename)
```

#### Throws:

- [ILCDEException](#)

#### Description:

#### by index:

Parameter	Range	Description
graphicIndex	0 ... max. graphic index	index of the graphic

#### by name or filename:

Parameter	Range	Description
graphicNameOrFilename	ASCII chars (0x01 .. 0xFF) or	name of the graphic or



	DOS filename (8.3 format)	name and path of the graphics file
--	---------------------------	------------------------------------

**both methods return:**

Returns	Description
static <a href="#">GraphicInfo</a>	information about the graphic

Both methods return a *GraphicInfo* object which contains information of a specific graphic.

### Note

- The *graphicIndex* range from 0xFFE0 to 0xFFFF is reserved for internal use!

### Example

```
GraphicInfo graphicA = Control.getGraphicInfo(0);
GraphicInfo graphicB = Control.getGraphicInfo("GRAPHIC");
GraphicInfo graphicC = Control.getGraphicInfo("DIR/FILE.RII");

int    graphicA_colorDepth      = graphicA.getColorDepth();
int    graphicA_graphicIndex    = graphicA.getGraphicIndex();
String graphicA_graphicName     = graphicA.getGraphicName();
Size   graphicA_size           = graphicA.getSize();
boolean graphicA_isAnimated     = graphicA.isAnimated();
boolean graphicA_isDisabled     = graphicA.isDisabled();
boolean graphicA_isTransparent  = graphicA.isTransparent();
int    graphicA_numberOfFrames = graphicA.getNumberOfFrames();
```

The first three methods will read the information of the graphic with index 0 and graphic "GRAPHIC" from the on-board flash as well as graphic "FILE.RII" from the SD card's "DIR" folder. Alternatively a value can be obtained with a single call:

```
String graphicName = Control.getGraphicInfo(0).getGraphicName();
Size   graphicSize = Control.getGraphicInfo(5).getSize();
```

### See also:

[GraphicInfo](#)  
[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)

Package [ilcd](#)  
Class [Control](#)

**Public Method getLCDContrast**

```
static int getLCDContrast()
```

**Description:**

Returns	Range	Description
value	0 ... 255	numerical value for the display contrast (255 = maximum)

Returns the current value set for the contrast of the display.

**Example**

```
int cont = Control.getLCDContrast();
```

**See also:**

[Control.getLCDGammaValue\(\)](#)  
[Control.getFixedLCDContrastGamma\(\)](#)  
[Control.setLCDContrast\(int\)](#)

---

Package [ilcd](#)  
Class [Control](#)

**Public Method getLCDGammaValue**

```
static int getLCDGammaValue()
```

**Description:**

Returns	Range	Description
value	0 ... 255	display gamma value (255 = maximum)

Returns the current value set for the gamma of the display.

**Example**

```
int LCDGammaValue = Control.getLCDGammaValue();
```

**See also:**

[Control.getLCDContrast\(\)](#)  
[Control.getFixedLCDContrastGamma\(\)](#)  
[Control.setLCDGammaValue\(int\)](#)

---

**Package** [ilcd](#)  
**Class** [Control](#)

**Public Method getTextExtent**

```
static Size getTextExtent(String textString)
```

**Description:**

Parameter	Description
textString	text string to measure

**this method returns:**

Returns	Description
static <a href="#">Size</a>	size of the text string

This method returns the extent of a text string when drawn with the current font. The *Size* return type delivers the *height* and *width* values of the *textString*.

**Note**

- Any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled.
- Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display.
- If you use a symbol font (refer to [Attribute.setSymbolFontEnabled\(boolean\)](#)) no horizontal or vertical spaces between the characters will be displayed, and this method calculates its values accordingly.
- If text alignment is on (see [Control.setTextAlignment\(int, int, int\)](#)), the reported *height* and *width* values correspond to the setting of the alignment. However, the current cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).

**Example**

```
Size sizeOfTextString = Control.getTextExtent("Hello World!");
int height = sizeOfTextString.getHeight();
int width = sizeOfTextString.getWidth();
```

Alternatively a value can be obtained with a single call:

```
int height = Control.getTextExtent("Hello World!").getHeight();
int width  = Control.getTextExtent("Hello World!").getWidth();
```

**See also:**[Size](#)[Draw.writeText\(String\)](#)[Draw.writeASCIIText\(String\)](#)[Control.getUnicodeTextExtent\(String\)](#)[Control.getTextMessageExtent\(int\)](#)Package [ilcd](#)Class [Control](#)**Public Method `getTextMessageExtent`**

```
static Size getTextMessageExtent(int messageIndex)
```

**Throws:**

- [ILCDEException](#)

```
static Size getTextMessageExtent(String name)
```

**Throws:**

- [ILCDEException](#)

**Description:****by index:**

Parameter	Range	Description
messageIndex	0 ... max. messageIndex	index of the message stored in the iLCD

**by name:**

Parameter	Range	Description
messageName	ASCII chars (0x01 .. 0xFF)	name of the message

**both methods return:**

Returns	Description
static <a href="#">Size</a>	size of the message

Both methods return the extent of a text message (stored in the iLCD via iLCD Manager XE) when drawn with the current font. The *Size* return type delivers the *height* and *width* values of the *message*.

### Note

- When addressing by *messageIndex*, the message offset (see [Extra.setMessageOffset\(int\)](#)) is taken into account.
- When addressing by *messageName*, the message prefix (refer to [Extra.setMessageNamePrefix\(String\)](#)) and suffix (refer to [Extra.setMessageNameSuffix\(String\)](#)) are taken into account.
- Any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled.
- Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display.
- If you use a symbol font (refer to [Attribute.setSymbolFontEnabled\(boolean\)](#)) no horizontal or vertical spaces between the characters will be displayed, and the this method calculates its values accordingly.
- If text alignment is on (see [Control.setTextAlignment\(int, int, int\)](#)), the reported *height* and *width* values correspond to the setting of the alignment. However, the current cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).

### Example

```
Size textMessage = Control.getTextMessageExtent(0);
int height = textMessage.getHeight();
int width = textMessage.getWidth();
```

Alternatively a value can be obtained with a single call:

```
int height = Control.getTextMessageExtent(0).getHeight();
int width = Control.getTextMessageExtent(0).getWidth();
```

### See also:

[Size](#)  
[Control.getTextExtent\(String\)](#)

---

## Package [ilcd](#) Class [Control](#)

### Public Method [getUnicodeTextExtent](#)

```
static Size getUnicodeTextExtent(String unicodeText)
```

### Throws:

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
unicodeText	Unicode chars ('/u0001' ... '/uFFFF')	unicode text to measure

**this method returns:**

Returns	Description
static <a href="#">Size</a>	size of the unicode text

This method returns the extent of a unicode text string according to the current font. The *Size* return type delivers the *height* and *width* values of the *unicodeText*.

**Note**

- Any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled.
- Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display.
- If you use a symbol font (refer to [Attribute.setSymbolFontEnabled\(boolean\)](#)) no horizontal or vertical spaces between the characters will be displayed, and the this method calculates its values accordingly.
- If text alignment is on (see [Control.setTextAlignment\(int, int, int\)](#)), the reported *height* and *width* values correspond to the setting of the alignment. However, the current cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).
- **Not supported controller types:** DPC3020, DPC2060, DPC10xx

**Example**

```
Size textUnicode = Control.getUnicodeTextExtent("Hello World!");
int height = textUnicode.getHeight();
int width = textUnicode.getWidth();
```

Alternatively a value can be obtained with a single call:

```
int height = Control.getUnicodeTextExtent("Hello World!").getHeight();
int width = Control.getUnicodeTextExtent("Hello World!").getWidth();
```

This response means that the Unicode text "Hello World!" will occupy a space of width = 81 (0x51) pixels, and height = 16 (0x10) pixels with the selected font.

**See also:**

[Draw.writeASCIIText\(String\)](#)  
[Draw.writeText\(String\)](#)  
[Control.getTextExtent\(String\)](#)  
[Control.getTextMessageExtent\(int\)](#)

Package [ilcd](#)  
 Class [Control](#)

### Public Method `incrementDecrementColumnAddress`

```
static void incrementDecrementColumnAddress (int xInc)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>xInc</code>	between the positive and negative boundary values see below	amount to increment or decrement the column address

positive boundary value of `xInc` = (display width) - (current cursor position in x direction) - 1  
 negative boundary value of `xInc` = -(current cursor position in x direction)

Increments (positive value) or decrements (negative value) the column address (horizontal cursor position) in pixel units relative to the current cursor position.

#### Note

- Take care about the boundary values of `xInc` which depends on the current cursor position. A runtime exception will occur, if the return value of [Control.getCursorPosition\(\).getX\(\)](#) plus the value of `xInc` gets higher than the display width ([Control.getDisplaySize\(\).getWidth\(\)](#)).

#### Example

```
Control.incrementDecrementColumnAddress (-40);
```

The example above moves the cursor position 40 pixels in the left direction.

#### See also:

[Control.setCursorPosition\(int, int\)](#)  
[Control.setRelativeCursorPosition\(int, int\)](#)  
[Control.setColumnAddress\(int\)](#)  
[Control.setRowAddress\(int\)](#)  
[Control.incrementDecrementRowAddress\(int\)](#)

Package [ilcd](#)  
 Class [Control](#)

**Public Method [incrementDecrementRowAddress](#)**

```
static void incrementDecrementRowAddress(int yInc)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
yInc	between the positive and negative boundary values see below	amount to increment or decrement the row address

positive boundary value of  $yInc = (\text{display height}) - (\text{current cursor position in y direction}) - 1$   
 negative boundary value of  $yInc = -(\text{current cursor position in y direction})$

Increments (positive value) or decrements (negative value) the row address (vertical cursor position) in pixel units relative to the current cursor position.

**Example**

```
Control.incrementDecrementRowAddress(40);
```

This example will move the cursor position 40 pixels down.

**See also:**

[Control.setCursorPosition\(int, int\)](#)  
[Control.setRelativeCursorPosition\(int, int\)](#)  
[Control.setRowAddress\(int\)](#)  
[Control.setColumnAddress\(int\)](#)  
[Control.incrementDecrementColumnAddress\(int\)](#)

Package [ilcd](#)  
 Class [Control](#)

**Public Method [isFixedLCDContrastGamma](#)**

```
static boolean isFixedLCDContrastGamma()
```

**Description:**

Return s	Range	Description
-------------	-------	-------------



<code>fixed</code>	<code>false ... true</code>	is the contrast/gamma fixed or not
--------------------	-----------------------------	------------------------------------

Returns whether the iLCD panel has a fixed contrast/gamma setting or not.

### Note

- Most iLCD panels do not need to have the contrast and gamma value set, as they have the optimum values already set by the TFT panel manufacturer. In this case (`fixed = 1`) although the methods for reading and setting the values can be carried out, there will be no visual effect!

### Example

```
boolean LCDContrastGamma = Control.isFixedLCDContrastGamma();
```

If `Control.isFixedLCDContrastGamma()` returns true, the iLCD panel has a fixed contrast/gamma setting.

### See also:

[Control.getLCDContrast\(\)](#)  
[Control.getLCDGammaValue\(\)](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method [selectViewport](#)

```
static void selectViewport(int viewport)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
<code>viewport</code>	<code>0 ... 8</code>	index of the viewport

Selects a specified viewport via its index `viewport`.

### Note

- A `viewport` of 0 represents the main viewport (the entire screen).
- Any other custom viewport can only be selected if it was previously defined with the [Control.defineViewport\(int, int, int, int\)](#) method.
- After selecting a viewport, the cursor position and all attributes (see class [Attribute](#)) are restored to the state when this viewport was lastly active.

- If *viewport* is other than 0 (main viewport for the active draw screen), the screen the viewport was defined on is activated as the draw screen.

### Example

```
Control.selectViewport(1);
```

This example selects the viewport defined with index 1.

### See also:

[Viewport Related Methods](#)

---

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setAnsiEnabled](#)

```
static void setAnsiEnabled(boolean enabled)
```

#### Description:

Parameter	Range	Description
enabled	false ... true	ANSI on (true) or off (false)

Enables or disables ANSI mode.

#### Note

- When ANSI support is disabled (*enabled* = false), ANSI sequences like carriage returns or escape sequences will not have any effect (see [ANSI Support](#)).
- The default value for *enabled* is true, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Control.setAnsiEnabled(true);
```

Enables the use of ANSI sequences.

### See also:

[ANSI Support](#)

---

Package [ilcd](#)  
Class [Control](#)

**Public Method `setAutoLinefeedEnabled`**

```
static void setAutoLinefeedEnabled(boolean enabled)
```

**Description:**

Parameter	Range	Description
<code>enabled</code>	false ... true	sets auto-linefeed on (true) or off (false)

Enables/Disables auto-linefeed.

**Note**

- When auto-linefeed is on, receiving a carriage return (0x0D or /r) causes the cursor to be set to the left margin and a new line (0x0A or /n) to be added automatically.
- The default value for `enabled` is true, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Control.setAutoLinefeedEnabled(true);
```

Activates auto-linefeed.

**See also:**

[Control.setTextAlignment\(int, int, int\)](#)  
[Control.setTabSpacing\(int\)](#)  
[Control.setWrapMode\(boolean, boolean\)](#)

Package [ilcd](#)  
Class [Control](#)

**Public Method `setBacklightBlinkFrequency`**

```
static void setBacklightBlinkFrequency(int period)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
-----------	-------	-------------

<code>period</code>	1 ... 255	interval of state changes in units of 10ms
---------------------	-----------	--

Sets the blinking frequency for the backlight by defining the interval of state changes.

#### Note

- The default value for *period* is 20, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- If the backlight is in blink mode (see [Control.setBacklightMode\(int\)](#)) when the method is called, the blink frequency changes immediately.

#### Example

```
Control.setBacklightBlinkFrequency(25);
```

This value of 25 (0x19) gives a frequency of 2 Hertz (250ms on and 250ms off).

#### See also:

[Control.setBacklightMode\(int\)](#)  
[Control.setBacklightIntensity\(int\)](#)

### Package [ilcd](#)

### Class [Control](#)

#### Public Method [setBacklightIntensity](#)

```
static void setBacklightIntensity(int intensity)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>intensity</code>	0 ... 15	intensity of the backlight (max. intensity = 15)

Sets the backlight of the display.

#### Note

- The default value for *intensity* is retrieved from the EEPROM emulation at location 1. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM in class [EEPROM](#).

- This method does not switch of the backlight! That can be done with [Control.setBacklightMode\(int\)](#).

### Example

```
Control.setBacklightIntensity(0);
```

Sets the backlight intensity to its lowest level.

### See also:

[Control.setBacklightMode\(int\)](#)  
[Control.setBacklightIntensityHighRes\(int\)](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setBacklightIntensityHighRes](#)

```
static void setBacklightIntensityHighRes(int fineIntensity)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
<code>fineIntensity</code>	0 ... 255	intensity of the backlight (max. intensity = 255)

Sets the backlight of the display in high-res steps from 0 to 255.

### Note

- This method is similar to [Control.setBacklightIntensity\(int\)](#) but provides a finer adjustment for the intensity (high-res).
- The default value for `fineIntensity` is retrieved from the EEPROM emulation at location 1 in the coarse range (adjustable by the [Control.setBacklightIntensity\(int\)](#) method). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM in class [EEPROM](#).
- Although an `fineIntensity` of 0 can dim the display completely. Use the method [Control.setBacklightMode\(int\)](#) to turn the display backlight off!

### Example

```
Control.setBacklightIntensityHighRes(125);
```

Sets the backlight intensity to a moderate level.

**See also:**

[Control.setBacklightMode\(int\)](#)  
[Control.setBacklightIntensity\(int\)](#)  
[Control.getBacklightIntensity\(\)](#)

Package [ilcd](#)

**Class [Control](#)**

**Public Method [setBacklightMode](#)**

```
static void setBacklightMode(int mode)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
mode	0 ... 3	mode for the backlight

Sets the backlight to the following options for *mode*:

mode
BACKLIGHT_OFF
BACKLIGHT_ON
BACKLIGHT_BLINK
BACKLIGHT_FADE_OUT (when previously on or blinking)

**Note**

- The default value for *mode* is retrieved from the EEPROM emulation at location 2. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM in class [EEPROM](#).

**Example**

```
Control.setBacklightMode(Control.BACKLIGHT_BLINK);
```

This example causes the backlight to blink.

**See also:**

[Control.getBacklightMode\(\)](#)  
[Control.setBacklightBlinkFrequency\(int\)](#)  
[Control.setBacklightIntensity\(int\)](#)  
[EEPROM](#)

---

**Package [ilcd](#)**  
**Class [Control](#)****Public Method [setColumnAddress](#)**

```
static void setColumnAddress (int address)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
address	0 ... display width - 1	value for x coordinate

Sets the horizontal cursor position (column address).

**Note**

- Using this method changes the x value of the cursor position but not the y value (see [Control.setCursorPosition\(int, int\)](#)).
- Using values greater than the available column size of the current display causes a runtime exception.
- The default value for *address* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Control.setColumnAddress (25) ;
```

This example sets the x value of the cursor position to 25. The y value is not affected.

**See also:**

[Control.setCursorPosition\(int, int\)](#)  
[Control.setRowAddress\(int\)](#)

---

## Package [ilcd](#)

### Class [Control](#)

#### Public Method [setColumnCoordinatesScaling](#)

```
static void setColumnCoordinatesScaling(int mult, int div)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
mult	1 ... 4096	multiplier factor for horizontal positions
div	1 ... 4096	divisor factor for horizontal positions

Scales column coordinates (horizontal positions) by a factor according to *mult* and *div*.

#### Note

- Rounding errors may occur due to non-integer factors.
- This scaling is valid for all screens and viewports.
- This function can be used to migrate existing applications to another iLCD with a different display resolution.
- The parameter *radius* of the method [Draw.drawCircle\(int\)](#) is scaled with this factor.
- The display size is reported unscaled, i.e. as if both *mult* and *div* were 1 (refer to [Control.getDisplaySize\(\)](#)).
- The scan line methods are disabled as long as any coordinates scale factor is other than 1 (see [Draw.writeScanLine\(int, short\[\]\)](#) and [Draw.readScanLine\(int\)](#)).
- The default value for *mult* and *div* is 1. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

#### Example

```
Control.setColumnCoordinatesScaling(1, 2);
```

This method scales subsequent horizontal positions by a factor of 0.5 ( $mult/div = 1/2$ ).

#### See also:

[Control.getDisplaySize\(\)](#)  
[Control.setFontsScaling\(int\)](#)  
[Control.setGraphicsScaling\(int\)](#)  
[Control.setRowCoordinatesScaling\(int, int\)](#)



## Package [ilcd](#)

### Class [Control](#)

#### Public Method [setCursorPosition](#)

```
static void setCursorPosition(int posX, int posY)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>posX</code>	0 ... display width - 1	horizontal cursor position
<code>posY</code>	0 ... display height - 1	vertical cursor position

Sets the cursor to the specified position according to `posX` and `posY` coordinates.

#### Note

- The cursor position is used in conjunction with most drawing methods (refer to "See also:" below).
- This method sets the column address to `posX` and the row address to `posY`.
- The default value for `posX` and `posY` is 0. They will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

#### Example

```
Control.setCursorPosition(0, 0);
```

Sets the cursor to the top-left corner.

#### See also:

[Control.setRelativeCursorPosition\(int, int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Control.setColumnAddress\(int\)](#)  
[Control.setRowAddress\(int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Draw.setClearPixel\(boolean\)](#)  
[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)  
[Touch.createDefineTouchField\(int, int\)](#)  
[Draw.drawDot\(int\)](#)  
[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawCircle\(int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Draw.drawEllipse\(int, int, int\)](#)  
[Draw.writeText\(String\)](#)  
[Memory.saveCursorAttributesToMemory\(\)](#)  
[Control.defineViewport\(int, int, int, int\)](#)

[Draw.eraseDisplayArea\(int, int\)](#)  
[Draw.writeScanLine\(int, short\[\]\)](#)

---

Package [ilcd](#)

## Class [Control](#)

### Public Method [setFontScaling](#)

```
static void setFontScaling(int factor)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>factor</code>	1 ... 16	factor for scaling fonts

Scales subsequently drawn text pixel-wise according to *factor* in horizontal and vertical direction.

#### Note

- The factor set by this method is valid for all screens and viewports.
- The default font spacing is automatically scaled by this factor. When setting the spacing via [Attribute.setFontSpacing\(int, int\)](#), the coordinate scale factors are taken into account (refer to [Control.setRowCoordinatesScaling\(int, int\)](#) and [Control.setColumnCoordinatesScaling\(int, int\)](#)).
- The default value for *factor* is 1. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

#### Example

```
Control.setFontScaling(2);
```

Text will be scaled by a factor of 2 after this method.

#### See also:

[Control.setGraphicsScaling\(int\)](#)  
[Control.setColumnCoordinatesScaling\(int, int\)](#)  
[Control.setGraphicsScaling\(int\)](#)  
[Control.setRowCoordinatesScaling\(int, int\)](#)

---

## Package [ilcd](#)

### Class [Control](#)

#### Public Method [setGraphicAlignment](#)

```
static void setGraphicAlignment(int mode, int width, int height)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
mode	Bits 0 ... 3 and 7	alignment properties
width	0 ... display width (0 = full display)	width of the alignment area
height	0 ... display height (0 = full display)	height of the alignment area

Define an area in which the next graphic will be aligned when subsequently sending the method [Graphic.displayLocalGraphic\(int\)](#) or [Graphic.displayGraphicArea\(int, int, int, int, int\)](#) as well as [Graphic.loadAnimatedGraphics\(int, int\)](#), [Graphic.setAnimationCoordinatesToXY\(int, int, int\)](#) or [Graphic.setAnimationCoordinatesToCursorPosition\(int\)](#).

mode
GRAPHIC_CENTER_HORIZONTAL Y
GRAPHIC_CENTER_VERTICALLY
GRAPHIC_RIGHT_JUSTIFY
GRAPHIC_BOTTOM_JUSTIFY
GRAPHIC_TURN_ALIGNMENT_ON

#### Note

- After executing an drawing method, the alignment is automatically cleared. For consecutive aligning of graphics, this method has to be repeated!
- If a static graphic does not fit into the specified area, it will be cropped accordingly.
- GRAPHIC\_TURN\_ALIGNMENT\_ON of *mode* must always be set to enable the alignment settings. If only this bit is set, the next graphic is top/left justified and cropped in the specified area.
- If the *width* and/or *height* parameter is set to 0, the controller automatically uses the maximum area available (starting from the current cursor position when the drawing method is called).
- If the *width* and/or *height* parameter would exceed the resulting right/bottom margin when the actual drawing method is executed, the controller automatically adjusts the *width* and/or *height* accordingly to the margins.
- Some bits of *mode* do not make sense when used together, so there are some logical precedent rules (centering always overrules justifying).

- The default value for *mode* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Control.setGraphicAlignment(Control.GRAPHIC_CENTER_HORIZONTALLY |
    Control.GRAPHIC_CENTER_VERTICALLY |
    Control.GRAPHIC_TURN_ALIGNMENT_ON,
    100, 200);
```

This will set graphic alignment to center the next drawn graphic or loaded animation horizontally and vertically in an area of 100 pixels width and 200 pixels height. To place a graphic into the centre of the display use the following command:

```
Control.setGraphicAlignment(Control.GRAPHIC_CENTER_HORIZONTALLY |
    Control.GRAPHIC_CENTER_VERTICALLY |
    Control.GRAPHIC_TURN_ALIGNMENT_ON,
    Control.getDisplaySize().getWidth(),
    Control.getDisplaySize().getHeight());
```

### See also:

- [Graphic.displayLocalGraphic\(int\)](#)
- [Graphic.loadAnimatedGraphics\(int, int\)](#)
- [Graphic.displayGraphicArea\(int, int, int, int, int\)](#)
- [Graphic.setAnimationCoordinatesToXY\(int, int, int\)](#)
- [Graphic.setAnimationCoordinatesToCursorPosition\(int\)](#)

## Package [ilcd](#)

### Class [Control](#)

#### Public Method [setGraphicsScaling](#)

```
static void setGraphicsScaling(int factor)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
factor	1 ... 16	factor for scaling graphics

Scales subsequent graphics pixel-wise according to *factor* in horizontal and vertical direction.

**Note**

- This functionality can be used to "zoom" any graphic, e.g. when adapting images used on a 320x240 display to a 640x480 (VGA) iLCD.
- The scale factor is valid for all screens and viewports. Be aware, that all running animations are rescaled in the next frame update as soon as the scale factor is changed.
- If the scaled graphic exceeds the screen or viewport dimensions, scaled pixels that can't be completely drawn are omitted. This can lead to gaps between the cropped graphic and the according margin.
- The default value for *factor* is 1. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Control.setGraphicsScaling(2);
```

Graphics will be scaled by a factor of 2.

**See also:**

[Control.setFontsScaling\(int\)](#)  
[Control.setColumnCoordinatesScaling\(int, int\)](#)  
[Control.setRowCoordinatesScaling\(int, int\)](#)

**Package [ilcd](#)****Class [Control](#)****Public Method [setLCDContrast](#)**

```
static void setLCDContrast(int value)
```

**Description:**

Parameter	Range	Description
value	0 ... 255	contrast value of the display (255 = maximum)

Sets the current contrast value of the LCD display.

**Note**

- The default value for *value* is retrieved from the EEPROM emulation at location 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM](#).
- Not all LCD panels carry out changes to the contrast setting. Please refer to [Control.getFixedLCDContrastGamma\(\)](#).

## Example

```
Control.setLCDContrast(255);
```

Sets the contrast of the display to its maximum.

### See also:

[Control.getLCDContrast\(\)](#)

[Control.getFixedLCDContrastGamma\(\)](#)

---

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setLCDGammaValue](#)

```
static void setLCDGammaValue(int value)
```

#### Description:

Parameter	Range	Description
value	0 ... 255	display gamma value

Sets the gamma value of the LCD panel.

#### Note

- The default value for *value* is retrieved from the EEPROM emulation at location 3. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- When the EEPROM is erased, the default value is obtained from the Flash data set on the "Settings" page of iLCD Manager XE. See further information about EEPROM at [EEPROM](#).
- Not all LCD panels carry out changes to the gamma setting. Please refer to [Control.getFixedLCDContrastGamma\(\)](#).

## Example

```
Control.setLCDGammaValue(255);
```

The gamma will be set to the maximum.

### See also:

[Control.getLCDGammaValue\(\)](#)

[Control.getFixedLCDContrastGamma\(\)](#)

---

## Package [ilcd](#)

### Class [Control](#)

#### Public Method [setLineStyle](#)

```
static void setLineStyle(int style)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
style	1 ... 255	style of the line

This method allows a definition for a line style used by [Draw.drawLine\(int, int\)](#) and [Draw.drawRectangle\(int, int, int\)](#). The parameter *style* represents a bit mask where a 1 represents a pixel to be written and a 0 a pixel to be omitted. The following table shows some examples for line styles:

style
LINE_SOLID
LINE_DOTTED
LINE_DASHED_4_4 (4 pixel black, 4 pixel white)
LINE_DASHED_2_2 (2 pixel black, 2 pixel white)
LINE_DASH_DOTTED

#### Note

- The default value for *style* is LINE\_SOLID (solid line). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- The least significant bit of style describes the first pixel painted when drawing a line from left to right.
- When line thickness is greater than 1 (refer to [Attribute.setLineThickness\(int\)](#)), every bit represents a dot with a diameter of line thickness. The line ending mode (refer to [Attribute.setLineEndingMode\(int\)](#)) is then taken into account as well.
- The current line style is used when a rectangle is painted allowing the drawing of various styles of rectangles as well. When using a line style other than LINE\_SOLID, the two rectangular line caps styles lead to the same result (refer to [Attribute.setLineCapsStyle\(int\)](#)).

#### Example

```
Control.setLineStyle(Attribute.LINE_DASHED_2_2);
Attribute.setLineStyle(Attribute.LINE_DASHED_2_2);
```

Both methods are equivalent and will set the line style to a dashed line with 2 pixels black and 2 pixels white.

**See also:**

[Attribute.setLineThickness\(int\)](#)  
[Attribute.setLineEndingMode\(int\)](#)  
[Attribute.setLineCapsStyle\(int\)](#)  
[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)

**Package [ilcd](#)****Class [Control](#)****Public Method [setRelativeCursorPosition](#)**

```
static void setRelativeCursorPosition(int xInc, int yInc)
```

**Throws:**

- [ILCDException](#)

**Description:**

Parameter	Range	Description
xInc	between the positive and negative boundary values see below	amount to increment or decrement the horizontal cursor position
yInc	between the positive and negative boundary values see below	amount to increment or decrement the vertical cursor position

positive boundary value of  $xInc = (\text{display width}) - (\text{current cursor position in x direction}) - 1$   
 negative boundary value of  $xInc = -(\text{current cursor position in x direction})$

positive boundary value of  $yInc = (\text{display height}) - (\text{current cursor position in y direction}) - 1$   
 negative boundary value of  $yInc = -(\text{current cursor position in y direction})$

Moves the cursor relative to the current position by the values  $xInc$  and  $yInc$ .

**Note**

- The cursor position is used in conjunction with most drawing methods (refer to "See also:" below).
- The default cursor position is 0/0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Control.setRelativeCursorPosition(-10, 20);
```

Moves the cursor 10 pixel to the left and 20 pixel down.



**See also:**

[Control.setCursorPosition\(int, int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Control.setColumnAddress\(int\)](#)  
[Control.setRowAddress\(int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Draw.setClearPixel\(boolean\)](#)  
[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)  
[Touch.createDefineTouchField\(int, int\)](#)  
[Draw.drawDot\(int\)](#)  
[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawCircle\(int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Draw.drawEllipse\(int, int, int\)](#)  
[Draw.writeText\(String\)](#)  
[Memory.saveCursorAttributesToMemory\(int\)](#)  
[Control.defineViewport\(int, int, int, int\)](#)  
[Draw.eraseDisplayArea\(int, int\)](#)  
[Draw.writeScanLine\(int, short\[\]\)](#)

---

**Package [ilcd](#)****Class [Control](#)****Public Method [setRowAddress](#)**

```
static void setRowAddress(int address)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
address	0 ... display height - 1	value for an y coordinate

Sets the vertical cursor position (row address).

**Note**

- Using this method changes the y value of the cursor position but not the x value (see [Control.setCursorPosition\(int, int\)](#)).
- Using values greater than the available raw size of the current display causes a runtime exception.
- The default value for *address* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

## Example

```
Control.setRowAddress(50);
```

This example sets the y value of the cursor position to 50. The x value is not affected.

### See also:

[Control.setCursorPosition\(int, int\)](#)

[Control.setColumnAddress\(int\)](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setRowCoordinatesScaling](#)

```
static void setRowCoordinatesScaling(int mult, int div)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
mult	1 ... 4096	multiplier for vertical positions
div	1 ... 4096	divisor for vertical positions

Scales row coordinates (vertical positions) by a factor according to *mult* and *div*.

### Note

- Rounding errors may occur due to non-integer factors.
- This scaling is valid for all screens and viewports.
- This function can be used to migrate existing applications to another iLCD with a different display resolution.
- The display size may be reported as unscaled, i.e. as if both *mult* and *div* were 1 (refer to [Control.getDisplaySize\(\)](#)).
- The scan line methods are disabled as long as any coordinates scale factor is other than 1 (see [Draw.writeScanLine\(int, short\[\]\)](#) and [Draw.readScanLine\(int\)](#)).
- The parameter *radius* of the method [Draw.drawCircle\(int\)](#) is scaled with the column coordinates scale factor (see [Control.setColumnCoordinatesScaling\(int, int\)](#)).
- The default value for *mult* and *div* is 1. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

## Example

```
Control.setRowCoordinatesScaling(3, 2);
```

This method scales subsequent vertical positions by a factor of 1.5 ( $mult/div = 3/2$ ).

### See also:

[Control.getDisplaySize\(\)](#)  
[Control.setFontsScaling\(int\)](#)  
[Control.setGraphicsScaling\(int\)](#)  
[Control.setColumnCoordinatesScaling\(int, int\)](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setScreenOrientation](#)

```
static void setScreenOrientation(int orientation)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
orientation	0 ... 3	screen orientation

Sets the screen orientation to one of the following modes:

orientation	Description
ORIENTATION_LANDSCAPE	0°
ORIENTATION_PORTRAIT	90°
ORIENTATION_LANDSCAPE_UPSIDE_DOWN	180°
ORIENTATION_PORTRAIT_UPSIDE_DOWN	270°

#### Note

- This method causes a [General.resetAll\(\)](#) to be carried out without setting the default orientation.
- The default value for *orientation* is 0, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

## Example

```
Control.setScreenOrientation(Control.ORIENTATION_PORTRAIT);
```

## See also:

[Control.setTextGraphicOrientation\(int\)](#)

---

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setTabSpacing](#)

```
static void setTabSpacing(int tabSpacing)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
tabSpacing	1 ... 16	tabulator spacing

Sets the tabulator spacing. The next available multiple of *tabSpacing* is the next character position where the cursor is placed when a TAB character is outputted.

#### Note

- The font width of the currently selected font is multiplied by *tabSpacing* for the cursor position calculation.
- When using a proportional font, the font width is not identical with the character width of single characters.
- The default value for *tabSpacing* is 8, but can be modified on the "Settings" page of iLCD Manager XE. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

## Example

```
Control.setTabSpacing(5);
```

Sets the tabulator spacing to font width times 5.

## See also:

[Control.setTextAlignment\(int, int, int\)](#)  
[Control.setAutoLinefeedEnabled\(boolean\)](#)  
[Control.setWrapMode\(boolean, boolean\)](#)

## Package [ilcd](#)

### Class [Control](#)

#### Public Method `setTextAlignment`

```
static void setTextAlignment(int mode, int width, int height)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
mode	Bit 0 ... 7	alignment properties
width	0 ... display width (0 = full display)	width of the alignment area
height	0 ... display height (0 = full display)	height of the alignment area

This powerful method helps you to align text on the screen automatically. The next invocation of the methode [Draw.writeText\(String\)](#), [Draw.writeTextMessage\(int\)](#) or [Control.getTextExtent\(String\)](#) will align, word-wrap and crop text corresponding to the *mode* set by this method in the specified area. A maximum of 1024 characters and 48 text lines can be aligned. The *mode* consists of the following bits:

mode
TEXT_CENTER_HORIZONTALLY
TEXT_CENTER_VERTICALLY
TEXT_RIGHT_JUSTIFY
TEXT_BOTTOM_JUSTIFY
TEXT_DO_NOT_WORD_WRAP
TEXT_ADD_HORIZONTAL_SPACE_OF_BORDER
TEXT_ADD_VERTICAL_SPACE_OF_BORDER
TEXT_TURN_ALIGNMENT_ON

#### Note

- After executing a method, for example [Draw.writeTextMessage\(int\)](#), the alignment is automatically cleared. For consecutive aligning of text strings, the set text alignment method has to be repeated!
- Bit 7 must always be set to enable the alignment settings. Clearing this bit allows an accidentally set alignment to be disabled before the text output.
- If alignment is turned on and none of the bits TEXT\_CENTER\_HORIZONTALLY, TEXT\_CENTER\_VERTICALLY, TEXT\_RIGHT\_JUSTIFY or TEXT\_BOTTOM\_JUSTIFY are set,

following text will be top/left justified (just as without any alignment), but also wrapped and cropped in the specified alignment area.

- Some bits do not make sense when used together, so there are some logical precedent rules (centering always overrules justifying).
- If the text does not fit into the given area, it will be truncated. If word wrap cannot be done due to words, which are longer than the available space, or word wrapping is switched off, the rest of the word will be continued on the next line.
- If the *width* and/or *height* parameter would exceed the resulting right/bottom margin when the actual method (for example [Draw.writeText\(String\)](#)) is executed, the controller automatically adjusts the *width* and/or *height* accordingly to the margins.
- If the *width* and/or *height* parameter is set to 0, the controller automatically uses the maximum area available (starting from the current cursor position when an actual method is called).
- Even ANSI sequences (such as setting the font, etc.) can be used within the text to be aligned, although using cursor control ANSI commands within text alignments does not make sense and may produce unwanted results.
- Carriage returns and linefeeds can be used to force a new line regardless of the actual horizontal space already used for the current line. Entering a CR/LF pair causes one new line, entering e.g. two consecutive CRs causes two new lines.
- The default value for *mode* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

### Example

```
Control.setTextAlignment(Control.TEXT_CENTER_HORIZONTALLY |
                        Control.TEXT_CENTER_VERTICALLY |
                        Control.TEXT_TURN_ALIGNMENT_ON,
                        200, 100);
```

This will set text alignment to center subsequently outputted text horizontally and vertically in an area of 200 pixels width and 100 pixels height.

### See also:

[Draw.writeText\(String\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Control.getTextExtent\(String\)](#)  
[Attribute.setLineStyle\(int\)](#)  
[Control.setFontsScaling\(int\)](#)  
[ANSI Support](#)

---

## Package [ilcd](#)

## Class [Control](#)

### Public Method [setTextGraphicOrientation](#)

```
static void setTextGraphicOrientation(int orientation)
```

### Throws:

- [ILCDException](#)

**Description:**

Parameter	Range	Description
orientation	0 ... 3	screen orientation

Sets the orientation of subsequent text and graphic drawing to one of the following modes:

orientation	Description
ORIENTATION_LANDSCAPE	0°
ORIENTATION_PORTRAIT	90°
ORIENTATION_LANDSCAPE_UPSIDE_DOWN	180°
ORIENTATION_PORTRAIT_UPSIDE_DOWN	270°

**Note**

- This orientation is interpreted relative to the screen orientation and, if used, viewport orientation (see [Control.setScreenOrientation\(int\)](#) and [General.defineViewport\(int, int, int, int\)](#)).
- The default value for *orientation* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- When using [Control.setTextAlignment\(int, int, int\)](#), width and height of the alignment area are interpreted according to the selected text orientation (refer to Example below).

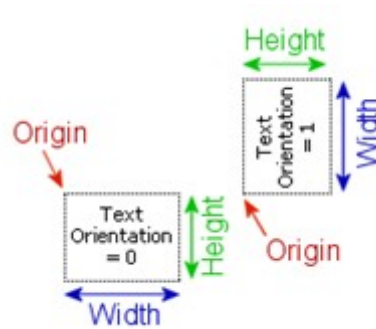
**Example**

```
General.resetAll();
Control.setLineStyle(Attribute.LINE_DOTTED);

Control.setCursorPosition(50, 130);
Control.setTextAlignment(Control.TEXT_CENTER_HORIZONTALLY |
                        Control.TEXT_CENTER_VERTICALLY |
                        Control.TEXT_TURN_ALIGNMENT_ON,
                        65, 50);
Draw.drawRectangle(0, 65, 50);
Draw.writeText("Text Orientation = 0");

Control.setCursorPosition(150, 130);
Control.setTextGraphicOrientation(ORIENTATION_PORTRAIT);
Control.setTextAlignment(Control.TEXT_CENTER_HORIZONTALLY |
                        Control.TEXT_CENTER_VERTICALLY |
                        Control.TEXT_TURN_ALIGNMENT_ON,
                        65, 50);
Draw.writeText("Text Orientation = 1");

// Draw Rectangle is not affected by Text Orientation,
// thus the point of origin is different and width are swapped
Control.setCursorPosition(150, 66);
Draw.drawRectangle(0, 65, 50);
```

**See also:**

[Control.setScreenOrientation\(int\)](#)  
[Concept of Viewports](#)  
[Control.setTextAlignment\(int, int, int\)](#)  
[Control.setCursorPosition\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)

**Package [ilcd](#)****Class [Control](#)****Public Method [setWrapMode](#)**

```
static void setWrapMode(boolean horzWrap, boolean vertWrap)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
horzWrap	false ... true	sets horizontal wrap on (true) or off (false)
vertWrap	false ... true	sets vertical wrap on (true) or off (false)

Enables/Disables character wrapping for horizontal and/or vertical character output.

**Note**

- When horizontal wrapping is on (true), characters outputted are automatically placed on a new line if there is no more space on the current line.
- When vertical wrap mode is on (true), the screen scrolls up when there is not enough space for the next line to be outputted to the LCD.
- The startup values for *horzWrap* and *vertWrap* are set via iLCD Manager XE.
- The default values for *horzWrap* and *vertWrap* are set to true, but can be modified on the "Settings" page of iLCD Manager XE. They will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).



## Example

```
Control.setWrapMode(true, false);
```

Activates horizontal wrapping but not vertical wrapping.

### See also:

[Control.setTextAlignment\(int, int, int\)](#)  
[Control.setAutoLinefeedEnabled\(boolean\)](#)  
[Control.setTabSpacing\(int\)](#)

## Package [ilcd](#)

## Class [Control](#)

### Public Method [turnDisplayOnOff](#)

```
static void turnDisplayOnOff(int mode)
```

#### Description:

Parameter	Range	Description
mode	0 ... 2	display mode

Turns the display on or off. The following modes can be set:

mode
DISPLAY_OFF
DISPLAY_ON_DELAYED
DISPLAY_ON_IMMEDIATELY

#### Note

- While the display is turned off, the iLCD still processes all commands that are sent. The results show on the screen as soon as it is turned on again.
- When using *mode DISPLAY\_ON\_IMMEDIATELY*, the display shows a blank screen for approximately 300ms. To avoid this white screen, set the backlight intensity to 0 (refer to [Control.setBacklightIntensity\(int\)](#)) before turning on the display or use mode *DISPLAY\_ON\_DELAYED* instead.

## Example

```
Control.turnDisplayOnOff(Control.DISPLAY_ON_DELAYED);
```

**See also:**

[Control.setBacklightIntensity\(int\)](#)

---

**Package [ilcd](#)****Class DeviceInfo**

extends [Object](#)

The *DeviceInfo* class provides information about the firmware and hardware currently in use. Please refer to the [General.getDeviceInfo\(\)](#) class to obtain information about the firmware and/or hardware.

**Public Methods**

- [getFirmwareInfo](#)
- [getFirmwareVersion](#)
- [getHardwareRevision](#)
- [getIdentificationInfo](#)
- [getILCDControllerName](#)
- [getSerialNumber](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

**Package [ilcd](#)****Class [DeviceInfo](#)****Public Method [getFirmwareInfo](#)**

```
String getFirmwareInfo()
```

**Description:**

Returns	Description
firmwareInfo	firmware information

The *getFirmwareInfo()* method returns a string with information about the actual firmware. Example string which is returned by this method: *iLCD Firmware V5.00 (c) by demmel products*

## Example

```
DeviceInfo deviceInfo = General.getDeviceInfo();  
Draw.writeText(deviceInfo.getFirmwareInfo());
```

### See also:

[General.getDeviceInfo\(\)](#)

---

Package [ilcd](#)

Class [DeviceInfo](#)

### Public Method [getFirmwareVersion](#)

```
String getFirmwareVersion ()
```

#### Description:

Returns	Description
firmwareVersion	firmware version

The *getFirmwareVersion()* method returns the firmware version as major and minor version separated by a dot within one single string.

Example string which is returned by this method: *5.00*

## Example

```
DeviceInfo deviceInfo = General.getDeviceInfo();  
Draw.writeText(deviceInfo.getFirmwareVersion());
```

### See also:

[General.getDeviceInfo\(\)](#)

---

Package [ilcd](#)

Class [DeviceInfo](#)

### Public Method [getHardwareRevision](#)

```
String getHardwareRevision ()
```

**Description:**

Returns	Description
hardwareRevision	hardware revision

The `getHardwareRevision()` method returns a text string containing the hardware revision of the iLCD panel.

Example string which is returned by this method: `3.0`

**Example**

```
DeviceInfo deviceInfo = General.getDeviceInfo();
Draw.writeText(deviceInfo.getHardwareRevision());
```

**See also:**

[General.getDeviceInfo\(\)](#)

---

Package [ilcd](#)

Class [DeviceInfo](#)

**Public Method `getIdentificationInfo`**

```
String getIdentificationInfo()
```

**Description:**

Returns	Description
identificationInfo	identification information

The `getIdentificationInfo()` method returns a string with the actual identification information.

Example string which is returned by this method: `iLCD Firmware`

**Example**

```
DeviceInfo deviceInfo = General.getDeviceInfo();
Draw.writeText(deviceInfo.getIdentificationInfo());
```

**See also:**

[General.getDeviceInfo\(\)](#)

---

**Package [ilcd](#)****Class [DeviceInfo](#)****Public Method [getILCDControllerName](#)**

```
String getILCDControllerName()
```

**Description:**

Returns	Description
iLCDControllerName	controller name

The `getILCDControllerName()` method returns the name of the iLCD controller of the module as a string.

Example string which is returned by this method: *DPC3090*

**Example**

```
DeviceInfo deviceInfo = General.getDeviceInfo();  
Draw.writeText(deviceInfo.getILCDControllerName());
```

**See also:**

[General.getDeviceInfo\(\)](#)

---

**Package [ilcd](#)****Class [DeviceInfo](#)****Public Method [getSerialNumber](#)**

```
String getSerialNumber()
```

**Description:**

Returns	Description
serial	serial number

The `getSerialNumber()` method returns a string containing the unique serial number of the iLCD module.

Example string which is returned by this method: *ID-CILCD-0131185698-DDD7*

**Example**

```
DeviceInfo deviceInfo = General.getDeviceInfo();  
Draw.writeText(deviceInfo.getSerialNumber());
```

**See also:**

[General.getDeviceInfo\(\)](#)

---

**Package [ilcd](#)****Class Draw**

extends [Object](#)

The *Draw* class allows the adjustment of the brightness, contrast, hue, or saturation of a specific area or the whole display. Furthermore, different shapes like rectangle, ellipse, or circle are available which can be filled with another color and can be also adjusted in different ways. The *Draw* class contains also the possibility to write a text message or a single line to the screen. The color information of a specific display area can be read and afterwards drawn to a different display position.

**Note**

- It's not possible to instantiate this class. The methods in this class are static.

**Public Methods**

- [adjustDisplay](#)
- [adjustDisplayArea](#)
- [drawCircle](#)
- [drawDot](#)
- [drawDotAtXY](#)
- [drawEllipse](#)
- [drawLine](#)
- [drawRectangle](#)
- [drawStyledCircle](#)
- [eraseDisplay](#)
- [eraseDisplayArea](#)
- [fillDisplay](#)
- [fillDisplayArea](#)
- [invertDisplay](#)
- [invertDisplayArea](#)
- [read1D2DRunLengthEncodedScanLine](#)
- [readScanLine](#)
- [scrollDown](#)
- [scrollLeft](#)
- [scrollRight](#)
- [scrollUp](#)
- [setClearPixel](#)
- [setClearPixelAtXY](#)
- [write1D2DRunLengthEncodedScanLine](#)
- [writeASCIIText](#)
- [writeScanLine](#)
- [writeScanLineAndAdvance](#)
- [writeText](#)
- [writeTextMessage](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final byte ADJUST\_BRIGHTNESS
- static final byte ADJUST\_CONTRAST
- static final byte ADJUST\_HUE
- static final byte ADJUST\_INVERT
- static final byte ADJUST\_SATURATION
- static final byte CIRCLE\_ALPHA\_BLENDING\_FOR\_SHADOW\_ONLY
- static final byte CIRCLE\_DRAW\_WITHOUT\_FRAME
- static final byte CIRCLE\_DROP\_SHADOW
- static final byte CIRCLE\_FILLED\_WITH\_BACKGROUND\_COLOR
- static final byte CIRCLE\_FILLED\_WITH\_CURRENT\_FILL\_PATTERN
- static final byte CIRCLE\_USE\_FOREGROUND\_COLOR\_FOR\_FILLING
- static final byte ELLIPSE\_ALPHA\_BLENDING\_FOR\_SHADOW\_ONLY
- static final byte ELLIPSE\_DRAW\_WITHOUT\_FRAME
- static final byte ELLIPSE\_DROP\_SHADOW
- static final byte ELLIPSE\_FILLED\_WITH\_BACKGROUND\_COLOR
- static final byte ELLIPSE\_FILLED\_WITH\_CURRENT\_FILL\_PATTERN
- static final byte ELLIPSE\_USE\_FOREGROUND\_COLOR\_FOR\_FILLING
- static final byte RECTANGLE\_ALPHA\_BLENDING\_FOR\_SHADOW\_ONLY
- static final byte RECTANGLE\_DRAW\_WITHOUT\_FRAME
- static final byte RECTANGLE\_DROP\_SHADOW
- static final byte RECTANGLE\_FILLED\_WITH\_BACKGROUND\_COLOR
- static final byte RECTANGLE\_FILLED\_WITH\_CURRENT\_FILL\_PATTERN
- static final byte RECTANGLE\_ROUNDED\_CORNERS
- static final byte RECTANGLE\_SOUROUNDED\_WITH\_BACKGROUND\_COLOR
- static final byte RECTANGLE\_USE\_FOREGROUND\_COLOR\_FOR\_FILLING

## Package [ilcd](#)

### Class [Draw](#)

#### Public Method [adjustDisplay](#)

```
static void adjustDisplay(int mode)
```

#### Description:

Parameter	Range	Description
mode	Bit 0 ... 4	adjustable mode (see below)

Adjusts the color values of all pixels on the screen. The value for any selected modification has to be set in advance and the corresponding *mode* must be set:

mode
ADJUST_BRIGHTNESS (see <a href="#">Attribute.setBrightnessAdjustment(int)</a> )
ADJUST_CONTRAST (see <a href="#">Attribute.setContrastAdjustment(int)</a> )
ADJUST_HUE (see <a href="#">Attribute.setHueAdjustment(int)</a> )
ADJUST_SATURATION (see <a href="#">Attribute.setSaturationAdjustment(int)</a> )
ADJUST_INVERT

### Note

- All adjustment modifications can be combined and changed simultaneously with this method.
- The modifications are processed in the following order: invert - hue/saturation - brightness/contrast.
- This method can be used to dim or brighten the entire screen by adjusting brightness and contrast.

### Example

```
Draw.adjustDisplay(Draw.ADJUST_BRIGHTNESS |
                  Draw.ADJUST_CONTRAST);
```

Adjusts brightness and contrast of the entire screen, using the previously set values.

### See also:

[Draw.adjustDisplayArea\(int, int, int\)](#)  
[Attribute.setBrightnessAdjustment\(int\)](#)  
[Attribute.setContrastAdjustment\(int\)](#)  
[Attribute.setHueAdjustment\(int\)](#)  
[Attribute.setSaturationAdjustment\(int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)

## Package [ilcd](#)

## Class [Draw](#)

### Public Method [adjustDisplayArea](#)

```
static void adjustDisplayArea(int mode, int width, int height)
```

### Throws:

- [ILCDEException](#)



**Description:**

Parameter	Range	Description
mode	Bit 0 ... 4	flags for possible modifications
width	0 ... display width (0 = full width)	horizontal value for the area to be adjusted
height	0 ... display height (0 = full height)	vertical value for the area to be adjusted

Adjusts the color values of all pixels in the area specified by *width* and *height*, beginning from the current cursor position. The value for any selected modification has to be set in advance and the corresponding *mode* must be set:

mode
ADJUST_BRIGHTNESS (see <a href="#">Attribute.setBrightnessAdjustment(int)</a> )
ADJUST_CONTRAST (see <a href="#">Attribute.setContrastAdjustment(int)</a> )
ADJUST_HUE (see <a href="#">Attribute.setHueAdjustment(int)</a> )
ADJUST_SATURATION (see <a href="#">Attribute.setSaturationAdjustment(int)</a> )
ADJUST_INVERT

**Note**

- All adjustment modifications can be combined and changed simultaneously with this method.
- The modifications are processed in the following order: invert - hue/saturation - brightness/contrast.
- This method can be used to dim or brighten part of the screen by adjusting brightness and contrast of an area.
- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without throwing an exception.

**Example**

```
Draw.adjustDisplayArea(Draw.ADJUST_HUE |
    Draw.ADJUST_SATURATION,
    200, 100);
```

Adjusts hue and saturation in a 200x100 pixel area starting at the cursor position, using the previously set values.

**See also:**

[Draw.adjustDisplay\(int\)](#)  
[Attribute.setBrightnessAdjustment\(int\)](#)  
[Attribute.setContrastAdjustment\(int\)](#)  
[Attribute.setHueAdjustment\(int\)](#)  
[Attribute.setSaturationAdjustment\(int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Control.setCursorPosition\(int, int\)](#)  
[Control.getCursorPosition\(\)](#)

---

Package [ilcd](#)

## Class [Draw](#)

### Public Method [drawCircle](#)

```
static void drawCircle(int radius)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
radius	1 ... max[width, height]	radius of the circle

Draws a circle with the radius *radius* and the current cursor position as center.

#### Note

- For *radius*, the column coordinates scale factor is used (see [Control.setColumnCoordinatesScaling\(int, int\)](#)).
- Drawing a circle is done with the current foreground color. When inverse mode is on, the current background color is used instead.
- This method draws an empty circle, ignoring line thickness and active fill patterns. To draw a circle with the current attributes, the method [Draw.drawStyledCircle\(int, int\)](#) can be used.

#### Example

```
Draw.drawCircle(10);
```

Draws a circle with a radius of 10 pixels to the current cursor position.

#### See also:

[Draw.drawLine\(int, int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawEllipse\(int, int, int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setForegroundColor\(int\)](#)

---

Package [ilcd](#)  
Class [Draw](#)

**Public Method drawDot**

```
static void drawDot(int colorValue)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
colorValue	0x000000 ... 0xFFFFFFFF	color of the dot

Sets the pixel at the current cursor position to *colorValue*.

**Note**

- No advancing of the current cursor position will occur.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), the pixel is drawn with the according opacity.

**Example**

```
Draw.drawDot(0xFF0000);
```

Draws a red dot at the current cursor position.

**See also:**

[Draw.drawDotAtXY\(int, int, int\)](#)  
[Draw.setClearPixel\(boolean\)](#)  
[Draw.setClearPixelAtXY\(int, int, boolean\)](#)  
[Attribute.setAlpha\(int\)](#)

---

Package [ilcd](#)  
Class [Draw](#)

**Public Method drawDotAtXY**

```
static void drawDotAtXY(int posX, int posY, int colorValue)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
posX	0 ... display width - 1	horizontal position
posY	0 ... display height - 1	vertical position
colorValue	0x000000 ... 0xFFFFFFFF	color of the dot

Sets or clears a pixel at the specified position.

**Note**

- The cursor position will be updated after this method call.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), the pixel is drawn with the according opacity.

**Example**

```
Draw.drawDotAtXY(20, 35, 0x00FF00);
```

Draws a green dot at the position x = 20 and y = 35 pixels.

**See also:**

[Draw.drawDot\(int\)](#)  
[Draw.setClearPixel\(boolean\)](#)  
[Draw.setClearPixelAtXY\(int, int, boolean\)](#)  
[Attribute.setAlpha\(int\)](#)

**Package [ilcd](#)****Class [Draw](#)****Public Method drawEllipse**

```
static void drawEllipse(int mode, int vertexA, int vertexB)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
mode	Bit 1, 3 ... 6	flags for the ellipse properties
vertexA	1 ... display width	distance from center to intersection of ellipse and x-axis
vertexB	1 ... display height	distance from center to intersection of ellipse and y-axis

Draws an ellipse with the selected line thickness, using the current cursor position as center. The vertices *vertexA* and *vertexB* specify the dimensions, the parameter *mode* allows to set several properties for the ellipse:

mode
ELLIPSE_DROP_SHADOW
ELLIPSE_FILLED_WITH_BACKGROUND_COLOR
ELLIPSE_USE_FOREGROUND_COLOR_FOR_FILLING (bit 3 must be set too)
ELLIPSE_FILLED_WITH_CURRENT_FILL_PATTERN (bits 3 and 4 ignored if set)
ELLIPSE_DRAW_WITHOUT_FRAME
ELLIPSE_ALPHA_BLENDING_FOR_SHADOW_ONLY

### Note

- Drawing the ellipse outline is done with the current border color (see [Attribute.setBorderColor\(int\)](#)). The shadow is drawn with the current border shadow color (see [Attribute.setBorderShadowColor\(int\)](#)). Outline and shadow drawing is done with alpha blending if set before (see [Attribute.setAlpha\(int\)](#)).
- By default, alpha blending is done for every activated feature (outline, filling, shadow). When *mode* ELLIPSE\_ALPHA\_BLENDING\_FOR\_SHADOW\_ONLY is set, the shadow will still be drawn with alpha blending, but outline and filling will have full opacity (refer to [Attribute.setAlpha\(int\)](#)).
- The line style attribute is ignored for ellipse drawing (see [Attribute.setLineStyle\(int\)](#)).
- If the drop shadow bit is set, but the shadow offset (see [Attribute.setShadowOffset\(int, int\)](#)) leads to a negative center of the shadow ellipse, no shadow will be drawn.
- When filling with a pattern (*mode* ELLIPSE\_FILLED\_WITH\_CURRENT\_FILL\_PATTERN), the fill pattern has to be set in advance via [Attribute.setFillingColor\(int\)](#), [Attribute.setFillingGradient\(int, int, int, int\)](#) or [Attribute.setFillingTile\(int\)](#). When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).

### Example

```
Draw.drawEllipse(Draw.ELLIPSE_DROP_SHADOW |
                Draw.ELLIPSE_FILLED_WITH_CURRENT_FILL_PATTERN,
                250, 100);
```

Draws a filled ellipse with the vertices 250 and 100 and a drop shadow.

### See also:

[Draw.drawLine\(int, int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawCircle\(int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Attribute.setBorderColor\(int\)](#)  
[Attribute.setBorderShadowColor\(int\)](#)  
[Attribute.setShadowOffset\(int, int\)](#)  
[Attribute.setLineThickness\(int\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingGradient\(int, int, int, int\)](#)

[Attribute.setFillingTile\(int\)](#)  
[Attribute.setAlpha\(int\)](#)

Package [ilcd](#)  
 Class [Draw](#)

**Public Method drawLine**

```
static void drawLine(int endX, int endY)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
endX	0 ... display width - 1	horizontal ending position of the line
endY	0 ... display height - 1	vertical ending position of the line

Draws a line with the currently selected Line Style (see [Attribute.setLineStyle\(int\)](#)) starting from the current cursor position to the specified ending point.

**Note**

- After the line is drawn the cursor position will be the ending point of the line.
- Drawing a line is done with the current foreground color. When inverse mode is on, the current background color is used instead.

**Example**

```
Draw.drawLine(799, 479);
```

Draws a line from the top left corner to the lower right corner, assuming the cursor position was 0/0 before this method was issued and the display has a resolution of 800x480 pixels.

**See also:**

[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawCircle\(int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Attribute.setForegroundColor\(int\)](#)  
[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setLineStyle\(int\)](#)

## Package [ilcd](#)

### Class [Draw](#)

#### Public Method [drawRectangle](#)

```
static void drawRectangle(int mode, int width, int height)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
mode	Bit 0 ... 7	flags for the rectangle properties
width	1 ... display width	width of the rectangle
height	1 ... display height	height of the rectangle

Draws a rectangle starting from the current cursor position to the ending point specified via *width* and *height*. The parameter *mode* allows to set several properties for the rectangle:

mode
RECTANGLE_ROUNDED_CORNERS
RECTANGLE_DROP_SHADOW
RECTANGLE_SOUROUNDED_WITH_BACKGROUND_COLOR
RECTANGLE_FILLED_WITH_BACKGROUND_COLOR
RECTANGLE_USE_FOREGROUND_COLOR_FOR_FILLING (bit 3 must be set too)
RECTANGLE_FILLED_WITH_CURRENT_FILL_PATTERN (bits 3 and 4 ignored if set)
RECTANGLE_DRAW_WITHOUT_FRAME
RECTANGLE_ALPHA_BLENDING_FOR_SHADOW_ONLY

#### Note

- If *mode* RECTANGLE\_DRAW\_WITHOUT\_FRAME is not set, the rectangle frame is drawn with the current border color (see [Attribute.setBorderColor\(int\)](#)). If the line thickness is greater than 1 (refer to [Attribute.setLineThickness\(int\)](#)), the lines of the rectangle expand to both sides of the specified corner points.
- By default, alpha blending is done for every activated feature (frame, filling, shadow). When *mode* RECTANGLE\_ALPHA\_BLENDING\_FOR\_SHADOW\_ONLY is set, the shadow will still be drawn with alpha blending, but frame and filling will have full opacity (refer to [Attribute.setAlpha\(int\)](#)).
- The rectangle frame is drawn with regard to the current line style (see [Attribute.setLineStyle\(int\)](#)) only if the corner radius and alpha are set to their default values (refer to [Attribute.setRectangleCornerRadius\(int\)](#) and [Attribute.setAlpha\(int\)](#)).
- If *mode* DROP\_SHADOW is set, the drop shadow is drawn with the border shadow color (see [Attribute.setBorderShadowColor\(int\)](#)) with regard to the current shadow offset (see

[Attribute.setShadowOffset\(int, int\)](#)). If the offset leads to a negative left/upper corner of the shadow rectangle, no shadow will be drawn.

- When filling with a pattern (*mode* `RECTANGLE_FILLED_WITH_CURRENT_FILL_PATTERN`), the fill pattern has to be set in advance via [Attribute.setFillingColor\(int\)](#), [Attribute.setFillingGradient\(int, int, int, int\)](#) or [Attribute.setFillingTile\(int\)](#). When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).
- When the rounded corners flag (*mode* `RECTANGLE_ROUNDED_CORNERS`) is set, the attribute rectangle corner radius is taken into account (see [Attribute.setRectangleCornerRadius\(int\)](#)). If the corner radius exceeds half of the rectangle *width* and/or *height*, the flag is ignored and a rectangle without rounded corners is drawn.
- The surrounding line (*mode* `RECTANGLE_SURROUNDED_WITH_BACKGROUND_COLOR`) is only drawn when corner radius and alpha are set to their default values (refer to [Attribute.setRectangleCornerRadius\(int\)](#) and [Attribute.setAlpha\(int\)](#)) and a frame is drawn (*mode* `RECTANGLE_DRAW_WITHOUT_FRAME` is not set).
- Filling with a fill pattern as well as drawing without a frame (*mode* `RECTANGLE_FILLED_WITH_CURRENT_FILL_PATTERN` and `RECTANGLE_DRAW_WITHOUT_FRAME`) are supported by firmware versions 4.20 and higher. In previous versions, those bits are ignored.

### Example

```
Draw.drawRectangle(Draw.RECTANGLE_ROUNDED_CORNERS |
                  Draw.RECTANGLE_DROP_SHADOW,
                  50, 30);
```

Draws a rectangle with a width of 50 pixels and a height of 30 pixels at the actual cursor position. Moreover, the rectangle has rounded corners and a drop shadow.

### See also:

[Draw.drawCircle\(int\)](#)  
[Draw.drawStyledCircle\(int, int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setLineStyle\(int\)](#)  
[Attribute.setBorderColor\(int\)](#)  
[Attribute.setBorderShadowColor\(int\)](#)  
[Attribute.setLineThickness\(int\)](#)  
[Attribute.setLineCapsStyle\(int\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingGradient\(int, int, int, int\)](#)  
[Attribute.setFillingTile\(int\)](#)  
[Attribute.setAlpha\(int\)](#)  
[Draw.eraseDisplayArea\(int, int\)](#)

---

## Package **ilcd**

### Class **Draw**

#### **Public Method drawStyledCircle**

```
static void drawStyledCircle(int mode, int radius)
```



**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
mode	Bit 1, 3 ... 6	flags for the circle properties
radius	1 ... max[width, height]	radius of the circle

Draws a circle using the currently selected line thickness with the radius *radius* and the current cursor position as center. The parameter *mode* allows to set several properties for the circle:

mode
CIRCLE_DROP_SHADOW
CIRCLE_FILLED_WITH_BACKGROUND_COLOR
CIRCLE_USE_FOREGROUND_COLOR_FOR_FILLING (bit 3 must be set too)
CIRCLE_FILLED_WITH_CURRENT_FILL_PATTERN (bits 3 and 4 ignored if set)
CIRCLE_DRAW_WITHOUT_FRAME
CIRCLE_ALPHA_BLENDING_FOR_SHADOW_ONLY

**Note**

- Drawing the circle outline is done with the current border color (see [Attribute.setBorderColor\(int\)](#)). The shadow is drawn with the current border shadow color (see [Attribute.setBorderShadowColor\(int\)](#)). Outline and shadow drawing is done with alpha blending if set before (see [Attribute.setAlpha\(int\)](#)).
- By default, alpha blending is done for every activated feature (outline, filling, shadow). When *mode* CIRCLE\_ALPHA\_BLENDING\_FOR\_SHADOW\_ONLY is set, the shadow will still be drawn with alpha blending, but outline and filling will have full opacity (refer to [Attribute.setAlpha\(int\)](#)).
- The line style attribute is ignored for circle drawing (see [Attribute.setLineStyle\(int\)](#)).
- If the drop shadow bit is set, but the shadow offset (see [Attribute.setShadowOffset\(int, int\)](#)) leads to a negative center of the shadow circle, no shadow will be drawn.
- When filling with a pattern (CIRCLE\_FILLED\_WITH\_CURRENT\_FILL\_PATTERN), the fill pattern has to be set in advance via [Attribute.setFillingColor\(int\)](#), [Attribute.setFillingGradient\(int, int, int, int\)](#) or [Attribute.setFillingTile\(int\)](#). When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).

**Example**

```
Draw.drawStyledCircle(Draw.CIRCLE_DRAW_WITHOUT_OUTLINE |
                    Draw.CIRCLE_DROP_SHADOW,
                    125);
```

Draws a filled circle without an outline with a radius of 125 pixels and a drop shadow.

**See also:**

[Draw.drawLine\(int, int\)](#)  
[Draw.drawCircle\(int\)](#)  
[Draw.drawRectangle\(int, int, int\)](#)  
[Draw.drawEllipse\(int, int, int\)](#)  
[Attribute.setBorderColor\(int\)](#)  
[Attribute.setBorderShadowColor\(int\)](#)  
[Attribute.setLineThickness\(int\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingGradient\(int, int, int, int\)](#)  
[Attribute.setFillingTile\(int\)](#)  
[Attribute.setAlpha\(int\)](#)

---

**Package [ilcd](#)****Class [Draw](#)****Public Method [eraseDisplay](#)**

```
static void eraseDisplay()
```

**Throws:**

- [ILCDEException](#)

**Description:**

Clears the entire active viewport to the current background color. Afterwards the cursor position is set to 0/0.

**Note**

- If the selected viewport is 0 (the entire screen), all animations on this draw screen are stopped, even if they are drawn in another viewport.
- If inverse mode is active, the viewport is cleared to the foreground color instead.

**Example**

```
Draw.eraseDisplay();
```

Clears the display to the background color.

**See also:**

[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setBackgroundColor\(int\)](#)  
[Draw.eraseDisplayArea\(int, int\)](#)

---

**Package [ilcd](#)**  
**Class [Draw](#)****Public Method [eraseDisplayArea](#)**

```
static void eraseDisplayArea(int width, int height)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
width	0 ... display width (0 = full width)	horizontal value for the area to be erased
height	0 ... display height (0 = full height)	vertical value for the area to be erased

Clears an area of the screen specified by the parameters *width* and *height* beginning from the current cursor position.

**Note**

- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without notifying an error.
- Depending on the current setting of the inverse mode, the area is either cleared to the background or to the foreground color.

**Example**

```
Draw.eraseDisplayArea(400, 480);
```

Clears the left half of a display with 800x480 pixels, provided the cursor position was 0/0.

**See also:**

[Draw.eraseDisplay\(\)](#)  
[Control.setCursorPosition\(\)](#)  
[Control.setCursorPosition\(int, int\)](#)  
[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setBackgroundColor\(int\)](#)

**Package [ilcd](#)**  
**Class [Draw](#)****Public Method [fillDisplay](#)**

```
static void fillDisplay()
```

**Throws:**

- [ILCDEException](#)

**Description:**

Fills the entire active viewport with the active fill pattern (see [Attribute.setFillingColor\(int\)](#), [Attribute.setFillingGradient\(int, int, int, int\)](#) or [Attribute.setFillingTile\(int\)](#)). Afterwards the cursor position is set to 0/0.

**Note**

- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), filling is done with the according opacity.
- When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).

**Example**

```
Draw.fillDisplay();
```

Fills the display with the active fill pattern.

**See also:**

[Draw.fillDisplayArea\(int, int\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingGradient\(int, int, int, int\)](#)  
[Attribute.setFillingTile\(int\)](#)  
[Attribute.setAlpha\(int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)

**Package [ilcd](#)****Class [Draw](#)****Public Method fillDisplayArea**

```
static void fillDisplayArea(int width, int height)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
width	0 ... display width (0 = full width)	horizontal value for the area to be filled
height	0 ... display height (0 = full height)	vertical value for the area to be filled

Fills an area of the screen with the active fill pattern (see [Attribute.setFillingColor\(int\)](#), [Attribute.setFillingGradient\(int, int, int, int\)](#) or [Attribute.setFillingTile\(int\)](#)), beginning from the current cursor position.

### Note

- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without sending an error.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), filling is done with the according opacity.
- When filling with a tile graphic, any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).

### Example

```
Draw.fillDisplayArea(100, 50);
```

Fills and area of 100x50 pixels starting from the cursor position.

### See also:

[Draw.fillDisplay\(\)](#)  
[Attribute.setFillingColor\(int\)](#)  
[Attribute.setFillingGradient\(int, int, int, int\)](#)  
[Attribute.setFillingTile\(int\)](#)  
[Attribute.setAlpha\(int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Control.getCursorPosition\(\)](#)  
[Control.setCursorPosition\(int, int\)](#)

---

## Package [ilcd](#) Class [Draw](#)

### **Public Method** `invertDisplay`

```
static void invertDisplay()
```

### Throws:

- [ILCDException](#)

### Description:

Inverts all pixels on the currently active screen.

### Note

- Using this method does not turn the inverse mode on or off.

## Example

```
Draw.invertDisplay();
```

Black pixels are now white, red pixels are cyan, etc.

### See also:

[Attribute.setInverseMode\(boolean\)](#)

---

## Package [ilcd](#)

## Class [Draw](#)

### Public Method [invertDisplayArea](#)

```
static void invertDisplayArea(int width, int height)
```

### Throws:

- [ILCDException](#)

### Description:

Parameter	Range	Description
<i>width</i>	0 ... display width (0 = full width)	horizontal value for the area to be inverted
<i>height</i>	0 ... display height (0 = full height)	vertical value for the area to be inverted

Inverts an area of the currently active screen specified by the parameters *width* and *height* beginning from the current cursor position.

### Note

- If the values for *width* and/or *height* exceed the right or bottom margin, the controller corrects the values accordingly without throwing an exception.
- Using this command does not turn the inverse mode on or off.

## Example

```
Draw.invertDisplayArea(100, 240);
```

Inverts all colors in an area of 100x240 pixels.

### See also:

[Draw.invertDisplay\(\)](#)

[Control.setCursorPosition\(int, int\)](#)

[Attribute.setInverseMode\(boolean\)](#)

## Package [ilcd](#)

### Class [Draw](#)

#### **Public Method read1D2DRunLengthEncodedScanLine**

```
static void read1D2DRunLengthEncodedScanLine (int prevLineOffset, int
noOfPixels, byte[] data)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
prevLineOffset	-8 ... 8	y-offset of scanline referred to (0 = none = 1D encoded)
noOfPixels	1 ... display width	number of pixel to be scanned
data	0x00 ... 0xFF	array storing run length encoded data

Reads the color information of *noOfPixels* pixels in a horizontal line from left to right, starting from the current cursor position and writes the run-length encoding data to the *data* array.

#### Note

- When *prevLineOffset* is 0 or *prevLineOffset* + the current row is negative, the data will be 1D encoded.
- When *prevLineOffset* is other than 0 and *prevLineOffset* + the current row is 0 or higher, 2D encoding with reference to the relatively specified row is carried out.
- Color values in the reported data are made up as [16-Bit Color Values](#).
- For detailed information about the encoding algorithm and its usage, please refer to the [2D Run-Length Encoding Application Note.pdf](#).

There is also a local *2D Run-Length Encoding Application Note* version available which is named as *2D RLE Application Note.pdf*, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

Returns	Range	Description
data[0] ... data[1]	2 ... 2 * display width + display width / 32 + 2	length of encoded data in bytes
data[2] ...	0x00 ... 0xFF	run-length encoded data

#### Example

```
byte[] data = new byte[2 + 2 * 1024 + 1024 / 32 + 2];
Draw.read1D2DRunLengthEncodedScanLine(0, 50, data);
```

In this example a buffer for a 1024 pixel wide display has been allocated. Then a line of 50 red pixels has been scanned.

**See also:**

[Draw.write1D2DRunLengthEncodedScanLine\(int, int, int, byte\[\]\)](#)  
[Draw.readScanLine\(int\)](#)  
[Draw.writeScanLine\(int, short\[\]\)](#)  
[16-Bit Color Values](#)

**Package** [ilcd](#)

**Class** [Draw](#)

**Public Method** [readScanLine](#)

```
static void readScanLine(int noOfPixels, short[] data)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
noOfPixels	1 ... display width	number of pixel to be scanned

Reads the color information of *noOfPixels* pixels in a horizontal line from left to right, starting from the current cursor position.

**Note**

- Data for any reported pixel is made up as [16-Bit Color Value](#).

Returns	Range	Description
data[0] ...	0x0000 ... 0xFFFF	run-length encoded data

**Example**

```
short[] data = new short[2 * 1024];  
Draw.readScanLine(5, data);
```

In this example a line of 5 pixels has been scanned.

**See also:**

[Draw.writeScanLine\(int, short\[\]\)](#)  
[Draw.read1D2DRunLengthEncodedScanLine\(int, int\)](#)



[Draw.write1D2DRunLengthEncodedScanLine\(int, int, int, byte\[\]\)](#)  
[16-Bit Color Values](#)

---

Package [ilcd](#)  
Class [Draw](#)

**Public Method [scrollDown](#)**

```
static void scrollDown(int scrollY)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>scrollY</code>	0 ... display height - 1	vertical distance in pixels to be scrolled

Scrolls the currently active viewport *scrollY* pixels down.

**Note**

- The new rows at the top are drawn in the current background color.

**Example**

```
Draw.scrollDown(50);
```

Scrolls the viewport content 50 pixels down.

**See also:**

[Draw.scrollUp\(int\)](#)  
[Draw.scrollLeft\(int\)](#)  
[Draw.scrollRight\(int\)](#)  
[Memory.scrollUpScreen\(int, int\)](#)  
[Memory.scrollDownScreen\(int, int\)](#)  
[Memory.scrollLeftScreen\(int, int\)](#)  
[Memory.scrollRightScreen\(int, int\)](#)

---

Package [ilcd](#)  
Class [Draw](#)

**Public Method scrollLeft**

```
static void scrollLeft(int scrollX)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
scrollX	0 ... display width - 1	horizontal distance in pixels to be scrolled

Scrolls the currently active viewport *scrollX* pixels to the left.

**Note**

- The new columns at the right are drawn in the current background color.

**Example**

```
Draw.scrollLeft(25);
```

Scrolls the viewport content 25 pixels to the left.

**See also:**

[Draw.scrollRight\(int\)](#)  
[Draw.scrollUp\(int\)](#)  
[Draw.scrollDown\(int\)](#)  
[Memory.scrollUpScreen\(int, int\)](#)  
[Memory.scrollDownScreen\(int, int\)](#)  
[Memory.scrollLeftScreen\(int, int\)](#)  
[Memory.scrollRightScreen\(int, int\)](#)

---

Package [ilcd](#)  
Class [Draw](#)

**Public Method scrollRight**

```
static void scrollRight(int scrollX)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>scrollX</code>	0 ... display width - 1	horizontal distance in pixels to be scrolled

Scrolls the currently active viewport `scrollX` pixels to the right.

**Note**

- The new columns at the left are drawn in the current background color.

**Example**

```
Draw.scrollRight(250);
```

Scrolls the viewport content 250 pixels to the right.

**See also:**

[Draw.scrollLeft\(int\)](#)

[Draw.scrollUp\(int\)](#)

[Draw.scrollDown\(int\)](#)

[Memory.scrollUpScreen\(int, int\)](#)

[Memory.scrollDownScreen\(int, int\)](#)

[Memory.scrollLeftScreen\(int, int\)](#)

[Memory.scrollRightScreen\(int, int\)](#)

**Package [ilcd](#)****Class [Draw](#)****Public Method `scrollUp`**

```
static void scrollUp(int scrollY)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>scrollY</code>	0 ... display height - 1	vertical distance in pixels to be scrolled

Scrolls the currently active viewport `scrollY` pixels up.

**Note**

- The new rows at the bottom are drawn in the current background color.

**Example**

```
Draw.scrollUp(25);
```

Scrolls the viewport content 25 pixels up.

**See also:**

[Draw.scrollDown\(int\)](#)

[Draw.scrollLeft\(int\)](#)

[Draw.scrollRight\(int\)](#)

[Memory.scrollUpScreen\(int, int\)](#)

[Memory.scrollDownScreen\(int, int\)](#)

[Memory.scrollLeftScreen\(int, int\)](#)

[Memory.scrollRightScreen\(int, int\)](#)

**Package [ilcd](#)****Class [Draw](#)****Public Method [setClearPixel](#)**

```
static void setClearPixel(boolean onOff)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
onOff	false ... true	determines whether a pixel is set (true) or cleared (false)

Sets or clears the pixel at the current cursor position.

**Note**

- No advancing of the current cursor position will occur.
- Setting/clearing a pixel is done with the current foreground/background color. When inverse mode is on, setting/clearing a pixel will be done in the opposite way.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), the pixel is drawn with the according opacity.

**Example**

```
Draw.setClearPixel(true);
```

Sets a pixel at the current cursor position.

**See also:**

[Draw.setClearPixelAtXY\(int, int, boolean\)](#)  
[Attribute.setBackgroundColor\(int\)](#)  
[Attribute.setForegroundColor\(int\)](#)  
[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setAlpha\(int\)](#)

---

**Package** [ilcd](#)

**Class** [Draw](#)

**Public Method** [setClearPixelAtXY](#)

```
static void setClearPixelAtXY(int posX, int posY, boolean onOff)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
posX	0 ... display width - 1	horizontal position
posY	0 ... display height - 1	vertical position
onOff	false ... true	determines whether a pixel is set (true) or cleared (false)

Sets or clears a pixel at the specified position.

**Note**

- The cursor position will be updated after this method call.
- Setting/clearing a pixel is done with the current foreground/background color. When inverse mode is on, setting/clearing a pixel will be done in the opposite way.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), the pixel is drawn with the according opacity.

**Example**

```
Draw.setClearPixelAtXY(120, 55, true);
```

Sets or clears a pixel at the position x = 120 and y = 55 pixels.

**See also:**

[Attribute.setBackgroundColor\(int\)](#)  
[Attribute.setForegroundColor\(int\)](#)

[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setAlpha\(int\)](#)

Package [ilcd](#)  
 Class [Draw](#)

**Public Method [write1D2DRunLengthEncodedScanLine](#)**

```
static void write1D2DRunLengthEncodedScanLine(int prevLineOffset, int noOfPixels, int noOfRleBytes, byte[] data)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
prevLineOffset	-8 ... 8	y-offset of scanline referred to (0 = none = 1D encoded)
noOfPixels	1 ... display width	number of pixels to write
noOfRleBytes	2 ... display width + 2	length of encoded data in bytes
data	0x00 ... 0xFF	array storing run length encoded data

Writes a horizontal scan line consisting of *noOfPixels* pixels onto the screen. The *noOfRleBytes* long data is sent with run-length encoding.

**Note**

- The location where the pixels are written to (where the horizontal line begins) is determined by the current cursor position.
- The column address is incremented after every pixel. Even so, the scan line is drawn correctly until it hits the screen margin.
- When *prevLineOffset* is 0 or *prevLineOffset* + the current row is negative, only 1D encoded data will be accepted.
- When *prevLineOffset* is other than 0 and *prevLineOffset* + the current row is 0 or higher, the data can include sequences for 2D encoding with reference to the relatively specified row.
- Color values in the reported data are made up as [16-Bit Color Values](#).
- For detailed information about the encoding algorithm and its usage, please refer to the [2D Run-Length Encoding Application Note](#).

There is also a local *2D Run-Length Encoding Application Note* version available which is named as *2D RLE Application Note.pdf*, and can be found under the iLCD Manager XE installation path within the *Documentation* directory. Other useful pdf documents can be downloaded on the [www.demmel.com/en/service/downloads.htm](http://www.demmel.com/en/service/downloads.htm) website.

## Example

```
byte[] data = new byte[] { (byte) 0xB1 , 0x1F, 0x00 };
Draw.write1D2DRunLengthEncodedScanLine(0, 50, 3, data);
```

This example will write 50 blue pixels to the screen, starting from the current cursor position. The encoded data consists of 3 bytes: 0xB1 indicates a horizontal repetition (bit 7 set) of 50 pixels and 0x1F00 is the 16-bit color value for pure blue.

### See also:

[Draw.read1D2DRunLengthEncodedScanLine\(int, int\)](#)

[Draw.writeScanLine\(int, short\[\]\)](#)

[Draw.readScanLine\(int\)](#)

[16-Bit Color Values](#)

## Package [ilcd](#)

## Class [Draw](#)

### Public Method [writeScanLine](#)

```
static void writeScanLine(int noOfPixels, short[] pixelData)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
noOfPixels	1 ... display width	number of pixels to write
pixelData	0x0000 ... 0xFFFF	color value for pixels

Writes a horizontal scan line consisting of *noOfPixels* pixels onto the screen.

### Note

- Data for any sent pixel is made up as a [16-Bit Color Values](#).
- The location where the pixels are written to (where the horizontal line begins) is determined by the current cursor position.

## Example

```
short[] pixelData = new short[] { 0x00, (short) 0xFFFF,
                                   0xF8, (short) 0xFFFF,
                                   0xF8, (short) 0xFFFF,
                                   0x00 };
Draw.writeScanLine(7, pixelData);
```

This example will write 7 pixels to the screen, starting from the current cursor position. The first and the last pixel are black, pixels in between are white or red alternatingly.

**See also:**

[Draw.writeScanLineAndAdvance\(int, short\[\]\)](#)

[Draw.readScanLine\(int\)](#)

[Draw.write1D2DRunLengthEncodedScanLine\(int, int, int, byte\[\]\)](#)

[Draw.read1D2DRunLengthEncodedScanLine\(int, int\)](#)

[16-Bit Color Values](#)

**Package [ilcd](#)**

**Class [Draw](#)**

**Public Method [writeScanLineAndAdvance](#)**

```
static void writeScanLineAndAdvance(int noOfPixels, short[] pixelData)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
noOfPixels	1 ... display width	number of pixels to write
pixelData	0x0000 ... 0xFFFF	color value for pixels

Writes a horizontal scan line consisting of *noOfPixels* pixels onto the screen.

**Note**

- Data for any sent pixel is made up as a [16-Bit Color Value](#).
- The location where the pixels are written to (where the horizontal line begins) is determined by the current cursor position.
- The column address is incremented after every pixel.
- Only the y cursor position is incremented after the method call.

**Example**

```
short[] pixelData = new short[] { 0x00, (short) 0xFFFF,
                                   0xF8, (short) 0xFFFF,
                                   0xF8, (short) 0xFFFF,
                                   0x00 };
Draw.writeScanLineAndAdvance(7, pixelData);
```

This example will write 7 pixels to the screen, starting from the current cursor position. The first and the last pixel are black, pixels in between are white or red alternatingly.



**See also:**[Draw.writeScanLine\(int, short\[\]\)](#)[Draw.readScanLine\(int\)](#)[Draw.write1D2DRunLengthEncodedScanLine\(int, int, int, byte\[\]\)](#)[Draw.read1D2DRunLengthEncodedScanLine\(int, int\)](#)[16-Bit Color Value](#)**Package [ilcd](#)****Class [Draw](#)****Public Method [writeText](#)**

```
static void writeText(String textString)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>textString</code>	Unicode symbols encoded in UTF8	text to be output

Writes a *textString* to the current cursor position. Note that Strings are encoded with UTF8 internally.

**Note**

- All characters in the string have to be defined in the currently active Unicode font. Characters not defined are replaced with a space.
- While writing text, the cursor position is constantly updated to the end of the last character. Use "\r" to position the cursor at the beginning of the next line ("\n" will go to the next line keeping the current indent).
- If the cursor position exceeds the right or bottom margin during text output and the corresponding wrap mode is not set, the following characters are ignored (refer to [Control.setWrapMode\(boolean, boolean\)](#)). The cursor will then remain at the end of the last complete character.
- To get the horizontal and vertical space occupied by the outputted text use the [Control.getTextExtent\(String\)](#) method.
- The text can contain any ANSI sequence (see [ANSI Support](#)) to e.g. control the cursor position (on a character or graphic based position) when the ANSI mode is on.

**Example**

```
Draw.writeText("\u2610: Hello x!");
```

Writes specified text including Unicode symbols to the screen at the current cursor position and with the currently selected unicode font and attributes.

**See also:**

[Draw.writeASCIIText\(String\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Control.setCursorPosition\(int, int\)](#)  
[Attribute.setFont\(int\)](#)  
[Control.getTextExtent\(String\)](#)  
[ANSI Support](#)  
[Attribute](#)

---

**Package [ilcd](#)****Class [Draw](#)****Public Method [writeTextMessage](#)**

```
static void writeTextMessage(int messageIndex)
```

**Throws:**

- [ILCDEException](#)

```
static void writeTextMessage(String messageName)
```

**Throws:**

- [ILCDEException](#)

**Description:****by index:**

Parameter	Range	Description
messageIndex	0 ... max. text message index	index of the defined message

**by name:**

Parameter	Range	Description
messageName	ASCII chars (0x01 .. 0xFF)	name of the message

Writes a text string similar to the [Draw.writeText\(String\)](#) method, except that it uses messages previously defined in iLCD Manager XE.

**Note**

- Text messages can contain ANSI sequences as well.
- When addressing by index, the message offset (see [Extra.setMessageOffset\(int\)](#)) is taken into account.

- When addressing by name, the message prefix (refer to [Extra.setMessageNamePrefix\(String\)](#)) and suffix (refer to [Extra.setMessageNameSuffix\(String\)](#)) are taken into account.

### Example

```
Draw.writeTextMessage(30);
Draw.writeTextMessage("MESSAGE");
```

Writes the message with index 3 and message "MESSAGE" to the screen at the current cursor position with the currently selected font and attributes.

### See also:

[Draw.writeText\(String\)](#)  
[Control.setCursorPosition\(int, int\)](#)  
[Attribute.setFont\(int\)](#)  
[Control.getTextExtent\(String\)](#)  
[ANSI Support](#)  
[Attribute](#)

## Package **ilcd**

### Class **Draw**

#### Public Method **writeASCIIText**

```
static void writeASCIIText(String textString)
```

#### Throws:

- [ILCDException](#)

#### Description:

Parameter	Range	Description
textString	Unicode symbols encoded in UTF8	text to be output

Writes a string to the current cursor position. Note that each byte of the string will be interpreted as ASCII code. The following example demonstrates the difference between this method and [Draw.writeText\(String\)](#).

```
// UTF8 code for "V" is 0xE2 0x88 0x80
Draw.writeASCIIText("V"); // prints 3 characters according to the
// definition in the current font for 0xE2, 0x88 and 0x80.
Draw.writeText("V"); // prints "V" if defined in the
// current font or SPACE
```

## Note

- Characters not defined in the character table of a font are replaced with a space.
- If the column address exceeds the right or bottom margin during printing, and the corresponding wrap mode is not set, the following characters are ignored.
- Writing text will increase the row or column address and therefore change the cursor position.
- To get the horizontal and vertical space occupied by the outputted text use the [Control.getUnicodeTextExtent\(String\)](#) method.
- The text can contain any ANSI sequence (see [ANSI Support](#)) to e.g. control the cursor position (on a character or graphic based position) when the ANSI mode is on.

## Example

Non Unicode Font symbols with codes above 127 can be displayed using a char-array.

```
Attribute.setFont("CP1252");

char[] c = {0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80};
String str = new String(c);

// use ASCII Font with codepage CP1252
Draw.writeASCIIText(str);
```

This will write "€€€€€€€€€€" to the screen (€ = 0x80 in CP1252).

## See also:

[Draw.writeText\(String\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Control.setCursorPosition\(int, int\)](#)  
[Attribute.setFont\(int\)](#)  
[Control.getUnicodeTextExtent\(String\)](#)  
[ANSI Support](#)  
[Attribute](#)

## Package [ilcd](#)

### Class EEPROM

extends [Object](#)

The DPC3090 iLCD controller offers an on-chip EEPROM with a size of 4016 bytes for user data. Only the first 9 values are used by the iLCD controller itself, however these values can be freely written to. Any other location may be used for the controlling application to store and retrieve its private data which is kept, even after a power loss.

The 9 special values affect the default startup values for the LCD

Location	Contents	Range of Values	Reference
----------	----------	-----------------	-----------

0	LCD Contrast	0 ... 255	<a href="#">Control.setLCDContrast(int)</a>
1	LCD Backlight Intensity	0 ... 15	<a href="#">Control.setBacklightIntensity(int)</a>
2	LCD Backlight Mode	0 ... 2	<a href="#">Control.setBacklightMode(int)</a>
3	LCD Gamma Value	0 ... 255	<a href="#">Control.setLCDGammaValue(int)</a>

and the (optional) PCAP touch screen

Location	Contents	Range of Values	Reference
4	PCAP Number of Fingers	1 ... 5	<a href="#">Touch.setNumberOfTouchFingers(int)</a>
5	PCAP Threshold	0 ... 255	iLCD Manager XE
6	PCAP Gain	0 ... 255	iLCD Manager XE
7	PCAP Offset	0 ... 255	iLCD Manager XE
8	PCAP Checksum	0 ... 255	iLCD Manager XE

Although any value can be written to any location, the iLCD controller will automatically restrict values to the corresponding range when retrieving data from the LCD (first 4) values of the EEPROM.

It is recommended to modify the PCAP values in the "Properties" window found on iLCD Manager XE's "Device" page, the checksum is then automatically calculated and written. If the checksum is not valid, factory default values are loaded at startup and written to EEPROM. To set the factory defaults manually, refer to [EEPROM.setPCAPConfigurationToFactoryDefault\(\)](#). It is also possible to set the PCAP values with the [EEPROM.write\(int, int\)](#) method. Therefore the PCAP Checksum is calculated with:  $\text{Checksum} = (\text{byte}) (\text{NumberOfFingers} + \text{Threshold} + \text{Gain} + \text{Offset})$ . All values have to be unsigned byte.

The 9 values mentioned above are restored from the EEPROM and the corresponding internal methods (e.g. for setting the backlight intensity) are carried out at startup (also when a [General.rebootPanelController\(\)](#) is issued) and when a [General.resetAll\(\)](#) or a [General.resetAllAndShowStartupGraphic\(\)](#) method is called.

Please note that writing to the 9 special EEPROM locations via the [EEPROM.write\(int, int\)](#) method does not call the associated method. The new settings will only become active upon startup. When you want to set something like the backlight intensity, use the appropriate method instead of – or in addition to – the [EEPROM.write\(int, int\)](#) method.

#### Note

- It's not possible to instantiate this class. The methods in this class are static.

#### Public Methods

- [erase](#)
- [getSize](#)
- [read](#)
- [setPCAPConfigurationToFactoryDefault](#)
- [write](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

## Package [ilcd](#)

### Class [EEPROM](#)

#### Public Method `erase`

```
static void erase()
```

#### Throws:

- [ILCDEException](#)

#### Description:

Erases the EEPROM.

#### Note

- When the EEPROM is erased (all values are written to 0xFF) the 4 special locations (see [EEPROM](#)) are loaded from the Flash data after the EEPROM has been formatted.
- The appropriate values in the Flash memory are set via iLCD Manager XE.

#### See also:

[EEPROM](#)

[EEPROM.write\(int, int\)](#)

[EEPROM.getSize\(\)](#)

---

## Package [ilcd](#)

### Class [EEPROM](#)

#### Public Method `getSize`

```
static int getSize()
```

#### Description:

Returns	Range	Description
---------	-------	-------------

<code>eepromSize</code>	EEPROM size	size of the EEPROM emulation
-------------------------	-------------	------------------------------

Retrieves the size of the EEPROM in bytes.

### Example

```
int eepromSize = EEPROM.getSize();
```

### See also:

[EEPROM](#)  
[EEPROM.write\(int, int\)](#)  
[EEPROM.read\(int\)](#)

Package [ilcd](#)

## Class [EEPROM](#)

### Public Method [read](#)

```
static int read(int address)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
<code>address</code>	0 ... EEPROM size - 1	address of the byte to read

Reads the contents of the EEPROM at the location specified by *address*.

Returns	Range	Description
<code>value</code>	0 ... 255	contents of the EEPROM

### Note

- Any location not previously written to will return 0xFF.

### Example

```
EEPROM.read() == 0xFF;
```

If this expression evaluates to true, the EEPROM is written to its default value 0xFF and is therefore empty.

**See also:**

[EEPROM](#)  
[EEPROM.write\(int, int\)](#)  
[EEPROM.getSize\(\)](#)

---

**Package** [ilcd](#)**Class** [EEPROM](#)**Public Method** [setPCAPConfigurationToFactoryDefault](#)

```
static void setPCAPConfigurationToFactoryDefault()
```

**Throws:**

- [ILCDEException](#)

**Description:**

Loads the factory default values for PCAP touch screens and writes them to the according EEPROM loactions.

**Note**

- After this method is sent, the default values will be loaded at startup until modified (refer to [EEPROM](#)).

**See also:**

[EEPROM](#)

---

**Package** [ilcd](#)**Class** [EEPROM](#)**Public Method** [write](#)

```
static void write(int address, int data)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
-----------	-------	-------------



address	0 ... EEPROM size - 1	address of the byte to write to
data	0 ... 255	data to write

Writes data to the EEPROM at the location specified by address.

#### Note

- The EEPROM section has a suggested life time of more than 1,000,000 write cycles.

#### Example

```
EEPROM.write(23, 5);
```

Writes the value of 5 (*data*) into the EEPROM at location 23 (*address*).

#### See also:

[EEPROM](#)  
[EEPROM.read\(int\)](#)  
[EEPROM.getSize\(\)](#)

---

## Package [ilcd](#)

### Class [Extra](#)

extends [Object](#)

The class *Extra* provides the functionality to change the prefix, suffix, or offset or a name, graphic, or message.

#### Note

- It's not possible to instantiate this class. The methods in this class are static.

#### Public Methods

- [setFontNamePrefix](#)
- [setFontNameSuffix](#)
- [setFontOffset](#)
- [setGraphicNamePrefix](#)
- [setGraphicNameSuffix](#)
- [setGraphicOffset](#)
- [setMessageNamePrefix](#)
- [setMessageNameSuffix](#)
- [setMessageOffset](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)

- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [ilcd](#)  
Class [Extra](#)

### Public Method [setFontNamePrefix](#)

```
static void setFontNamePrefix(String prefix)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>prefix</code>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent font name (see [Attribute.setFont\(int\)](#)).

#### Note

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the font names for the English language are prefixed with "en\_" and for the German language with "de\_", the method

```
Attribute.setFont("font");
```

would select the font "en\_font" or "de\_font", dependent on the prior defined font prefix.

- When addressing fonts by index, an offset can be used to realize language switching. (see [Extra.setFontOffset\(int\)](#)).
- Remove any previously set Font Name Prefix by entering an empty string as *prefix*:

```
Extra.setFontNamePrefix("");
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

## Example

```
Extra.setFontNamePrefix("en_");
```

Set the font name prefix to "en\_" and will subsequently only select fonts that start with these characters.

### See also:

[Touch.setTouchFieldTextMessage\(int\)](#)  
[Attribute.setFont\(int\)](#)  
[Control.getTextMessageExtent\(int\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Extra.setFontOffset\(int\)](#)  
[Extra.setFontNameSuffix\(String\)](#)  
[Extra.setMessageNamePrefix\(String\)](#)  
[Extra.setGraphicNamePrefix\(String\)](#)

## Package **ilcd**

## Class **Extra**

### Public Method **setFontNameSuffix**

```
static void setFontNameSuffix(String suffix)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
suffix	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent font name (see [Attribute.setFont\(int\)](#)).

#### Note

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the font names for the English language are suffixed with "\_en" and for the German language with "\_de", the method

```
Attribute.setFont("font");
```

would select the font "font\_en" or "font\_de", dependent on the prior defined font prefix.

- When addressing fonts by index, an offset can be used to realize language switching. (see [Extra.setFontOffset\(int\)](#)).

- Remove any previously set Font Name Suffix by entering an empty string as *suffix*:

```
Extra.setFontNameSuffix("");
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

### Example

```
Extra.setFontNameSuffix("_en");
```

Set the font name suffix to "\_en" and will subsequently only set fonts that end with these characters.

### See also:

[Touch.setTouchFieldTextMessage\(int\)](#)

[Attribute.setFont\(int\)](#)

[Control.getTextMessageExtent\(int\)](#)

[Draw.writeTextMessage\(int\)](#)

[Extra.setFontOffset\(int\)](#)

[Extra.setFontNamePrefix\(String\)](#)

[Extra.setMessageNameSuffix\(String\)](#)

[Extra.setGraphicNameSuffix\(String\)](#)

## Package [ilcd](#)

### Class [Extra](#)

#### Public Method [setFontOffset](#)

```
static void setFontOffset(int offset)
```

#### Description:

Parameter	Range	Description
<code>offset</code>	0 ... max. font index - 1	offset of the font index

Sets an offset for the font index (see [Attribute.setFont\(int\)](#)).

#### Note

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the fonts for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Font Offset to 10 will automatically select the German fonts instead of the corresponding English ones.

- When addressing fonts by name, a prefix or suffix can be used to realize language switching. (see [Extra.setFontNamePrefix\(String\)](#) or [Extra.setFontNameSuffix\(String\)](#)).
- The default value for *offset* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

### Example

```
Extra.setFontOffset(3);
Attribute.setFont(1);
Draw.writeText("Hello World!");
```

Sets the font offset to 3. The subsequently called method to select the font with index 1 will now set the font with index 4 instead.

### See also:

[Attribute.setFont\(int\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Touch.setTouchFieldTextMessage\(int\)](#)  
[General.resetAll\(\)](#)  
[Extra.setMessageOffset\(int\)](#)  
[Extra.setGraphicOffset\(int\)](#)

## Package [ilcd](#)

### Class [Extra](#)

#### Public Method [setGraphicNamePrefix](#)

```
static void setGraphicNamePrefix(String prefix)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
prefix	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent graphic name (see [Graphic.displayLocalGraphic\(int\)](#)).

## Note

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the graphic names for the English language are prefixed with "en\_" and for the German language with "de\_", the method

```
Graphic.displayLocalGraphic("graphic");
```

would display the graphic "en\_graphic" or "de\_graphic", dependent on the prior defined graphic prefix.

- When addressing graphics by index, an offset can be used to realize language switching (see [Extra.setGraphicOffset\(int\)](#)).
- Remove any previously set Graphic Name Prefix by entering an empty string as *prefix*:

```
Extra.setGraphicNamePrefix("");
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without loosing e.g. the selected language.

## Example

```
Extra.setGraphicNamePrefix("en_");
```

Set the graphic name prefix to "en\_" and will subsequently only set graphics that start with these characters.

### See also:

[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)  
[Extra.setMessageOffset\(int\)](#)  
[Extra.setGraphicNameSuffix\(String\)](#)  
[Extra.setMessageNamePrefix\(String\)](#)  
[Extra.setFontNamePrefix\(String\)](#)

---

## Package [ilcd](#)

## Class [Extra](#)

### Public Method setGraphicNameSuffix

```
static void setGraphicNameSuffix(String suffix)
```

### Throws:

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
suffix	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent graphic name (see [Graphic.displayLocalGraphic\(int\)](#)).

**Note**

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the graphic names for the English language are suffixed with "\_en" and for the German language with "\_de", the method

```
Graphic.displayLocalGraphic("graphic");
```

would display the graphic "graphic\_en" or "graphic\_de", dependent on the prior defined graphic prefix.

- When addressing graphics by index, an offset can be used to realize language switching. (see [Extra.setGraphicOffset\(int\)](#)).
- Remove any previously set Graphic Name Suffix by entering an empty string as *suffix*:

```
Extra.setGraphicNameSuffix("");
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

**Example**

```
Extra.setGraphicNameSuffix("_en");
```

Set the graphic name suffix to "\_en" and will subsequently only use graphics that end with these characters.

**See also:**

[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)  
[Extra.setGraphicOffset\(int\)](#)  
[Extra.setGraphicNamePrefix\(String\)](#)  
[Extra.setMessageNameSuffix\(String\)](#)  
[Extra.setFontNameSuffix\(String\)](#)

**Package [ilcd](#)**  
**Class [Extra](#)**

**Public Method [setGraphicOffset](#)**

```
static void setGraphicOffset(int offset)
```

**Description:**

Parameter	Range	Description
<code>offset</code>	0 ... max. graphic index - 1	offset of the graphic index

Sets an offset for the graphic index (see [Graphic.displayLocalGraphic\(int\)](#)).

**Note**

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the graphics for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Graphic Offset to 10 will automatically draw the German graphics instead of the corresponding English ones.
- When addressing graphics by name, a prefix or suffix can be used to realize language switching. (see [Extra.setGraphicNamePrefix\(String\)](#) or [Extra.setGraphicNameSuffix\(String\)](#)).
- The default value for `offset` is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

**Example**

```
Extra.setGraphicOffset(10);  
Graphic.displayLocalGraphic(1);
```

Sets the graphic offset to 10. The subsequently called method to draw the graphic with index 1 will now use the graphic with index 11 instead.

**See also:**

[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.eraseAnimationFrameArea\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)  
[Control.setTextGraphicOrientation\(int\)](#)  
[General.resetAll\(\)](#)  
[Extra.setMessageOffset\(int\)](#)  
[Extra.setFontOffset\(int\)](#)



Package [ilcd](#)  
Class [Extra](#)

### Public Method `setMessageNamePrefix`

```
static void setMessageNamePrefix(String prefix)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>prefix</code>	up to 4 ASCII chars (0x01 .. 0xFF)	string to be prefix

Sets a prefix of case-sensitive string characters, which will be put in front of any subsequently sent message name (see [Draw.writeTextMessage\(int\)](#)).

#### Note

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the message names for the English language are prefixed with "en\_" and for the German language with "de\_", the method

```
Draw.writeTextMessage("message");
```

would display the message "en\_message" or "de\_message", dependent on the prior defined message prefix.

- When addressing text messages by index, an offset can be used to realize language switching. (see [Extra.setMessageOffset\(int\)](#)).
- Remove any previously set Message Name Prefix by entering an empty string as *prefix*:

```
Extra.setMessageNamePrefix("");
```

- The default value for *prefix* is an empty string. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

#### Example

```
Extra.setMessageNamePrefix("en_");
```

Set the message name prefix to "en\_" and will subsequently only call a message that start with these characters.

**See also:**

[Touch.drawTouchFieldTextMessage\(int\)](#)  
[Touch.setTouchFieldTextMessage\(int\)](#)  
[Control.getTextMessageExtent\(int\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Extra.setMessageOffset\(int\)](#)  
[Extra.setMessageNameSuffix\(String\)](#)  
[Extra.setGraphicNamePrefix\(String\)](#)  
[Extra.setFontNamePrefix\(String\)](#)

**Package [ilcd](#)****Class [Extra](#)****Public Method [setMessageNameSuffix](#)**

```
static void setMessageNameSuffix(String suffix)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
suffix	up to 4 ASCII chars (0x01 .. 0xFF)	string to be suffix

Sets a suffix of case-sensitive string characters, which will be put after any subsequently sent message name (see [Draw.writeTextMessage\(int\)](#)).

**Note**

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the message names for the English language are suffixed with "\_en" and for the German language with "\_de", the method

```
Draw.writeTextMessage("message");
```

would display the message "message\_en" or "message\_de", dependent on the prior defined message suffix.

- When addressing text messages by index, an offset can be used to realize language switching. (see [Extra.setMessageOffset\(int\)](#)).
- Remove any previously set Message Name Suffix by entering an empty string as *suffix*:

```
Extra.setMessageNameSuffix("");
```

- The default value for *suffix* is an empty string. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the

"Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

### Example

```
Extra.setMessageNameSuffix("_en");
```

Set the message name suffix to "\_en" and will subsequently only call a message that end with these characters.

### See also:

[Touch.drawTouchFieldTextMessage\(int\)](#)  
[Touch.setTouchFieldTextMessage\(int\)](#)  
[Control.getTextMessageExtent\(int\)](#)  
[Draw.writeTextMessage\(int\)](#)  
[Extra.setMessageOffset\(int\)](#)  
[Extra.setMessageNamePrefix\(String\)](#)  
[Extra.setGraphicNameSuffix\(String\)](#)  
[Extra.setFontNameSuffix\(String\)](#)

[Extra.setMessageNamePrefix\(String\)](#)  
[Extra.setMessageOffset\(int\)](#)  
[Draw.writeTextMessage\(int\)](#)

## Package [ilcd](#) Class [Extra](#)

### Public Method [setMessageOffset](#)

```
static void setMessageOffset(int offset)
```

#### Description:

Parameter	Range	Description
offset	0 ... max. text message index - 1	offset of the message index

Sets a message index offset (see [Draw.writeTextMessage\(int\)](#)).

#### Note

- This functionality can be used to support multiple languages without conditional methods or macros. For example, if the messages for the English language are indexed 0 to 9 and for the German language 10 to 19, setting the Message Offset to 10 will automatically display the German messages instead of the corresponding English ones.
- When addressing text messages by name, a prefix or suffix can be used to realize language switching. (see [Extra.setMessageNamePrefix\(String\)](#) or [Extra.setMessageNameSuffix\(String\)](#)).

- The default value for *offset* is 0. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#) if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear". Selecting "Keep" allows for using the reset methods without losing e.g. the selected language.

### Example

```
Extra.setMessageOffset(5);  
Draw.writeTextMessage(10);
```

Sets the message offset to 5. The subsequently called method to write the message with index 10 will now put out the message with index 15 instead.

### See also:

[Draw.writeTextMessage\(int\)](#)  
[Touch.drawTouchFieldTextMessage\(int\)](#)  
[Touch.setTouchFieldTextMessage\(int\)](#)  
[Extra.setGraphicOffset\(int\)](#)  
[Extra.setFontOffset\(int\)](#)

---

## Package [ilcd](#)

### Class General

extends [Object](#)

To get information about the hardware, software, network status, and the last error code, the methods of the *General* class are used. Reboot, and reset methods are also available in this class.

### Note

- It's not possible to instantiate this class. The methods in this class are static.

### Public Methods

- [getDeviceInfo](#)
- [getFirmwareInfo](#)
- [getFirmwareVersion](#)
- [getHardwareRevision](#)
- [getIdentificationInfo](#)
- [getILCDControllerName](#)
- [getInputBufferSize](#)
- [getLastErrorCode](#)
- [getNetworkStatus](#)
- [getProjectInfo](#)
- [getSerialNumber](#)
- [getTouchScreenType](#)
- [isReportingTouchCoordinatesEnabled](#)
- [isSimulator](#)
- [noOperation](#)

- [rebootPanelController](#)
- [resetAll](#)
- [resetAllAndShowStartupGraphic](#)
- [writeApplicationDataToFlash](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final byte TOUCH\_SCREEN\_CAPACITIVE\_MOUNTED
- static final byte TOUCH\_SCREEN\_NOT\_MOUNTED
- static final byte TOUCH\_SCREEN\_RESISTIVE\_MOUNTED

## Package [ilcd](#)

### Class [General](#)

#### Public Method [getDeviceInfo](#)

```
static DeviceInfo getDeviceInfo()
```

#### Description:

Returns	Description
static <a href="#">DeviceInfo</a>	device information

The `getDeviceInfo()` method is used to obtain the information of the used device. The `DeviceInfo` return type delivers different information of the used device.

#### Example

```
DeviceInfo deviceInfo = General.getDeviceInfo();
Draw.writeText(deviceInfo.getFirmwareInfo());
```

#### See also:

[DeviceInfo](#)

**Package [ilcd](#)****Class [General](#)****Public Method [getFirmwareInfo](#)**

```
static String getFirmwareInfo()
```

**Description:**

Returns	Description
firmwareInfo	firmware information

Returns a string with the actual information about the firmware.

Return example: *iLCD Firmware V5.00 (c) by demmel products*

**Example**

```
String firmwareInfo = General.getFirmwareInfo();
```

**See also:**

[General.getFirmwareVersion\(\)](#)

[General.getIdentificationInfo\(\)](#)

[General.getILCDControllerName\(\)](#)

**Package [ilcd](#)****Class [General](#)****Public Method [getFirmwareVersion](#)**

```
static String getFirmwareVersion()
```

**Description:**

Returns	Description
firmwareVersion	firmware version

Returns the firmware version as major and minor version separated by a dot.

Return example: *5.00*

**Example**

```
String firmwareVersion = General.getFirmwareVersion();
```

**See also:**

[General.getFirmwareInfo\(\)](#)  
[General.getIdentificationInfo\(\)](#)  
[General.getILCDControllerName\(\)](#)

---

**Package [ilcd](#)****Class [General](#)****Public Method [getHardwareRevision](#)**

```
static String getHardwareRevision()
```

**Description:**

Returns	Description
hardwareRevision	hardware revision

Returns a text string containing the hardware revision of the iLCD panel..  
Return example: 3.0

**Example**

```
String hardwareRevision = General.getHardwareRevision();
```

**See also:**

[General.getFirmwareInfo\(\)](#)

---

**Package [ilcd](#)****Class [General](#)****Public Method [getIdentificationInfo](#)**

```
static String getIdentificationInfo()
```

**Description:**

Returns	Description
identificationInfo	identification information

The `getIdentificationInfo()` method returns a string with the actual identification information.

Example string which is returned by this method: *iLCD Firmware*

### Example

```
Draw.writeText (General.getIdentificationInfo ());
```

### See also:

[General.getFirmwareInfo\(\)](#)

[General.getFirmwareVersion\(\)](#)

[General.getILCDControllerName\(\)](#)

---

Package [ilcd](#)

Class [General](#)

### Public Method `getILCDControllerName`

```
static String getILCDControllerName ()
```

### Description:

Returns	Description
iLCDControllerName	controller name

The `getILCDControllerName()` method returns the name of the iLCD controller as a string.

Example string which is returned by this method: *DPC3090*

### Example

```
Draw.writeText (General.getILCDControllerName ());
```

### See also:

[General.getFirmwareInfo\(\)](#)

[General.getFirmwareVersion\(\)](#)

[General.getIdentificationInfo\(\)](#)



**Package [ilcd](#)****Class [General](#)****Public Method getInputBufferSize**

```
static int getInputBufferSize()
```

**Description:**

Return s	Description
size	input buffer size in bytes

Returns the size of the serial port's input buffer.  
Return example: 4

**Example**

```
int size = General.getInputBufferSize();
```

**See also:**

[Controlling the iLCD via Serial Port](#)

---

**Package [ilcd](#)****Class [General](#)****Public Method getLastErrorCode**

```
static int getLastErrorCode()
```

**Description:**

Return s	Range	Description
error	0x00 ... 0x4BA	error code

Returns the error code of the last executed method.  
Return example: 0

**Note**

- When a method is successfully executed, the error code is set to 0x0.

## Example

```
int error = General.getLastErrorCode();
```

Package [ilcd](#)

Class [General](#)

### Public Method [getNetworkStatus](#)

```
static NetworkStatus getNetworkStatus()
```

#### Description:

Returns	Description
static <a href="#">NetworkStatus</a>	status of the interface

The `getNetworkStatus()` method is used get information about the TCP/IP connection. See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE. A [NetworkStatus](#) object is returned containing structured information.

## Example

```
NetworkStatus netStat = General.getNetworkStatus();  
short[] accessIp = netStat.getAccessIp();  
short[] currentIp = netStat.getCurrentIp();  
short[] deviceIp = netStat.getDeviceIp();
```

In the third line a buffer for the current IP address is declared, allocated and set. Alternatively a value can be obtained with a single call:

```
short[] accessIp = General.getNetworkStatus().getAccessIp();
```

#### See also:

[NetworkStatus](#)

Package [ilcd](#)

Class [General](#)

### Public Method [getProjectInfo](#)

```
static ProjectInfo getProjectInfo()
```

**Description:**

Returns	Description
static <a href="#">ProjectInfo</a>	information about the project stored in flash memory

The `getProjectInfo()` method is used to obtain information of the current project. The `ProjectInfo` return type delivers project specific informations.

**Note**

- The project modification date and time are updated when a modified project is either saved, exported or written to flash. When an unchanged project is written to flash, the date and time of the last save is used.

**Example**

```
ProjectInfo proInfo = General.getProjectInfo();
String proDesc = proInfo.getProjectDescription();
("05.17.16".equals(proInfo.getProjectModificationDate()))
```

If the last expression evaluates to true, the project was written to flash on May 17th, 2016. a value can be obtained with a single call:

```
("10:23:19".equals(General.getProjectInfo().getProjectModificationTime()
))
```

**See also:**

[ProjectInfo](#)

**Package [ilcd](#)****Class [General](#)****Public Method [getSerialNumber](#)**

```
static String getSerialNumber()
```

**Description:**

Returns	Description
serialNumber	serial number

The `getSerialNumber()` method returns a text string containing the unique serial number of the iLCD module.

Example string which is returned by this method: `ID-CILCD-0131185698-DDD7`

**Example**

```

if ("DH-XADC-00131154001-0566".equals(General.getSerialNumber()))
{
    Draw.writeText("This serial number is correct!");
}
else
{
    Draw.writeText("This serial nuber is wrong!");
    Draw.writeText("The correct serial number of this "
        + "device is: " + General.getSerialNumber());
}

```

If the second expression evaluates to true, the serial number of the iLCD module is: DH-XADC-00131154001-0566.

Package [ilcd](#)

Class [General](#)

**Public Method getTouchScreenType**

```
static int getTouchScreenType()
```

**Description:**

Return s	Description
type	type of touch screen

Returns the type of the currently mounted touch screen. Possible touch screens are listed in the table *type* below.

type
TOUCH_SCREEN_NOT_MOUNTED
TOUCH_SCREEN_RESISTIVE_MOUNTED
TOUCH_SCREEN_CAPACITIVE_MOUNTE D

**Example**

```

(General.General.getTouchScreenType() ==
General.TOUCH_SCREEN_CAPACITIVE_MOUNTED)

```

If this expression evaluates to true, the used iLCD has a capacitive touch screen mounted.

**See also:**

[Touch](#)

---

Package [ilcd](#)

Class [General](#)

### **Public Method isReportingTouchCoordinatesEnabled**

```
static boolean isReportingTouchCoordinatesEnabled()
```

**Description:**

Returns	Range	Description
enabled	false ... true	reporting is on (true) or off (false)

Returns true if reporting of touch coordinates is enabled.

**Example**

```
General.isReportingTouchCoordinatesEnabled()
```

When touch reports are sent, the touch coordinates will be appended to the report.  
Not supported by: DPC3020, DPC2060, DPC10xx

**See also:**

[Touch](#)

---

Package [ilcd](#)

Class [General](#)

### **Public Method isSimulator**

```
static boolean isSimulator()
```

**Description:**

Returns	Range	Description
enabled	false ... true	simulator is on(ture) or off(false)

Specifies whether the iLCD Manager XE is in the simulator mode (true) or not (false).

### Example

```
String simulatorMode = General.isSimulator();
```

---

Package [ilcd](#)

Class [General](#)

### Public Method noOperation

```
static void noOperation()
```

#### Description:

Sending this sequence does not cause any action, but terminates any incomplete method.

#### Note

- This method may be used to check the presence of an LCD panel on the selected serial port.
- 

Package [ilcd](#)

Class [General](#)

### Public Method rebootPanelController

```
static void rebootPanelController()
```

#### Description:

The panel controller is rebooted which gives the same effect as a hard reset.

#### Note

- The baud rate for serial ports will be reset to the default value set on the "Settings" page of iLCD Manager XE.

#### See also:

[Power.hardShutdownLongPowerOff\(\)](#)

---

## Package [ilcd](#)

### Class [General](#)

#### **Public Method `resetAll`**

```
static void resetAll()
```

#### **Description:**

Clears entire screen area and sets default values for all attributes:

- Cursor is set to 0, 0
- The screen orientation is set to the default value set by iLCD Manager XE.
- All touch fields are removed, report touch field coordinates and movements is disabled
- Text alignment is reset
- Bold, inverse and underline attributes are set to off
- Color values are set to default
- Font is set to startup font
- Underline position is set to 0
- All animations are stopped
- Animation engine is started
- LCD contrast is set to the corresponding EEPROM value
- LCD Gamma value is set to the corresponding EEPROM value
- Backlight intensity is set to the corresponding EEPROM value
- The backlight state is set to the corresponding EEPROM value
- Watchdog is disabled
- Power/Watchdog related pins are released
- Shutdown is cancelled
- ANSI mode is set to default
- Horizontal and vertical wrapping mode is set to default
- Auto-Linefeed is set to default
- TAB spacing is set to default
- Backlight blink frequency is set to default
- Keyboard is enabled/disabled according to default value
- Keyboard reporting is turned on/off according to default value
- Relay outputs are set to default
- Output blink frequencies are set to default
- All outputs are reset to their startup states
- All open files on the MicroSD card are closed
- The main screen is selected as draw and view screen
- All viewports are removed
- All scale factors are set to 1
- All line attributes are set to default
- The Macro Timer is set 0

Only if the "Extras on Reset" option on the "Settings" page of iLCD Manager XE is set to "Clear":

- All offsets are set to 0
- All prefixes and suffixes are set to empty strings

#### **See also:**

[General.resetAllAndShowStartupGraphic\(\)](#)

Package [ilcd](#)  
 Class [General](#)

**Public Method [resetAllAndShowStartupGraphic](#)**

```
static void resetAllAndShowStartupGraphic ()
```

**Description:**

This method calls the [General.resetAll\(\)](#) method and then displays the startup graphic on the position defined via iLCD Manager XE.

**See also:**

[General.resetAll\(\)](#)

---

Package [ilcd](#)  
 Class [General](#)

**Public Method [writeApplicationDataToFlash](#)**

```
static void writeApplicationDataToFlash(int flags, String filename)
```

**Throws:**

- [IOException](#)
- [ILCDEException](#)

**Description:**

Parameter	Range	Description
flags	Bit 0, 1, 7	flags for the flash update
filename	DOS filename (8.3 format)	name and path of the *.rid or *.lcdp-rflash file

Copies a raw flash file (\*.rid or \*.lcdp-flash) from the MicroSD card to the internal flash memory of the iLCD controller.

Flags can have the options:

Bit	Description
0	no reboot
1	no verification
7	verbose mode



## Notes

- Note that this flag has no effect in Java. Verbose mode enables the iLCD controller to send progress information. It issues a "." character every 100kByte written and a "v" character when verifying is started.
- When setting bit 0 (no reboot) in *flags*, the new flash data is not accessible until the iLCD controller is rebooted. Until then, only a basic flash setup is available.
- To create a raw flash file, use the "Export" feature on the "File" page of iLCD Manager XE.

## See also:

[Write Application Data to Flash](#)

---

## Package [ilcd](#)

### Class **Graphic**

extends [Object](#)

#### General Information About Animated Graphics

All graphics - including the animated ones - are loaded into the Flash memory of the iLCD or stored on a MicroSD card via iLCD Manager XE. Although there can be an unlimited number of animated graphics, a maximum of 8 animated graphics can be animated at the same time on a LCD screen. While animated graphics are shown on the screen, any other method like drawing text or graphics to any cursor position can be carried out, as the iLCD's animation engine runs in the background of the iLCD controller's firmware. Scrolling the screen while animations are active can be done, but will cause unwanted effects, as the position of animated graphics are not changed via the scroll screen methods.

When working with animated graphics, the first thing to do is to load an animated graphic to one of the 8 animation controls (referred to as *anim\_loc* in the following methods). All other animation related methods refer to the index of the animation control regardless of which animated graphic is loaded into it.

Please note that the single frames of an animated graphic may contain areas of the image, which are smaller in width and/or height than the complete image. This is exactly how animated GIFs work (iLCD Manager XE imports animated GIFs and takes care about the frames' size and position). So setting a certain frame number via the [Graphic.stopAnimationAndSetFrameNumber\(int, int\)](#) method may cause surprising effects when one does not keep in mind that only the (smaller) frame contents of the selected frame will be drawn, and not the whole image which is normally created by the sequence of consecutive frames.

#### Note

- It's not possible to instantiate this class. The methods in this class are static.

#### Public Methods

- [displayGraphicArea](#)
- [displayLocalGraphic](#)
- [eraseAnimationFrameArea](#)
- [eraseAnimationImageArea](#)
- [loadAnimatedGraphics](#)

- [moveAnimationToFrame](#)
- [removeAnimationBackground](#)
- [resumeAnimationEngine](#)
- [setAnimationBackgroundColor](#)
- [setAnimationBackgroundFrame](#)
- [setAnimationBackgroundGraphic](#)
- [setAnimationBackgroundScreen](#)
- [setAnimationCoordinatesToCursorPosition](#)
- [setAnimationCoordinatesToXY](#)
- [setAnimationRepetitions](#)
- [startOrRestartAnimation](#)
- [stopAnimationAndSetFrameNumber](#)
- [stopBreakAnimation](#)
- [suspendAnimationEngine](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- static final byte MOVE\_ANIM\_AUTO
- static final byte MOVE\_ANIM\_BACKWARD
- static final byte MOVE\_ANIM\_FORWARD

---

#### Package [ilcd](#)

### Class [Graphic](#)

#### Public Method displayGraphicArea

```
static void displayGraphicArea(int startX, int startY, int width, int height, int graphicIndex)
```

#### Throws:

- [ILCDException](#)

```
static void displayGraphicArea(int startX, int startY, int width, int height, String graphicNameOrFilename)
```

#### Throws:

- [ILCDException](#)

**Description:****by index:**

Parameter	Range	Description
startX	0 ... graphic width - 1	horizontal offset in graphic
startY	0 ... graphic height - 1	vertical offset in graphic
width	1 ... graphic width	width of the area to display
height	1 ... graphic height	height of the area to display
graphicIndex	0 ... max. graphic index	index of the graphic

**by name or filename:**

Parameter	Range	Description
startX	0 ... graphic width - 1	horizontal offset in graphic
startY	0 ... graphic height - 1	vertical offset in graphic
width	1 ... graphic width	width of the area to display
height	1 ... graphic height	height of the area to display
graphicNameOrFilename	ASCII chars (0x01 .. 0xFF) or DOS filename (8.3 format)	name of the graphic or name and path of the graphics file

Draws a part of a graphic according to the graphic's *graphicIndex* or *graphicNameOrFilename* at the current cursor position.

**Note**

- Coordinates scaling is not taken into account for these values, as they refer to the graphic and not to the screen (refer to [Control.setColumnCoordinatesScaling\(int, int\)](#) and [Control.setRowCoordinatesScaling\(int, int\)](#)).
- Graphic alignment is taken into account (refer to [Control.setGraphicAlignment\(int, int, int\)](#)). If the alignment area is smaller than the specified graphic area, it will be cropped accordingly.
- When addressing by index, the graphic offset (see [Extra.setGraphicOffset\(int\)](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Extra.setGraphicNamePrefix\(String\)](#)) and suffix (refer to [Extra.setGraphicNameSuffix\(String\)](#)) are taken into account.
- The cursor position is not changed.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), filling is done with the according opacity.
- Any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).
- The *graphicIndex* range from 0xFFE0 to 0xFFFF is reserved for internal use!
- When inverse mode is on, the graphic is shown in inverse mode.
- Graphics with transparencies are always drawn transparent, regardless of the transparent mode (see [Attribute.setTransparentModeEnabled\(boolean\)](#)).
- Monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.
- While graphics with a color depth of 8 bit occupy less space in the memory, 16 bit graphics are drawn faster as there is no need for converting calculations.

- To convert an image to a Raw iLCD Graphics Image, use the "Save As" button on iLCD Manager XE's "Graphics" page and select the \*.rii filetype.
- Graphics stored manually on the MicroSD card can be loaded by their filenames, but not by their graphic names.

### Example

```
Graphic.displayGraphicArea(100,200,10,20,5);
Graphic.displayGraphicArea(100,200,10,20,"GRAPHIC");
Graphic.displayGraphicArea(100,200,10,20,"DIR/FILE.RII");
```

Displays an area (100x200 pixel from pixel at position 10/20) of graphic with index 5 and graphic "GRAPHIC" from the on-board flash as well as graphic "FILE.RII" from the SD card's "DIR" folder at the current cursor position.

### See also:

[Graphic.displayLocalGraphic\(int\)](#)  
[Control.setGraphicAlignment\(int, int, int\)](#)  
[Extra.setGraphicOffset\(int\)](#)  
[Extra.setGraphicNamePrefix\(String\)](#)  
[Extra.setGraphicNameSuffix\(String\)](#)  
[Attribute.setAlpha\(int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)

## Package [ilcd](#)

## Class [Graphic](#)

### Public Method displayLocalGraphic

```
static void displayLocalGraphic(int graphicIndex)
```

#### Throws:

- [ILCDEException](#)

```
static void displayLocalGraphic(String graphicNameOrFilename)
```

#### Throws:

- [ILCDEException](#)

#### Description:

#### by index:

Parameter	Range	Description
graphicIndex	0 ... max. graphic index	index of the graphic

**by name or filename:**

Parameter	Range	Description
<code>graphicNameOrFilename</code>	ASCII chars (0x01 .. 0xFF) or DOS filename (8.3 format)	name of the graphic or name and path of the graphics file

Draws a graphic according to the graphic's `graphicIndex` or `graphicNameOrFilename` at the current cursor position.

**Note**

- Graphic alignment is taken into account (refer to [Control.setGraphicAlignment\(int, int, int\)](#)). If the alignment area is smaller than the specified graphic, it will be cropped accordingly.
- When addressing by index, the graphic offset (see [Extra.setGraphicOffset\(int\)](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Extra.setGraphicNamePrefix\(String\)](#)) and suffix (refer to [Extra.setGraphicNameSuffix\(String\)](#)) are taken into account.
- The cursor position is not changed.
- If an alpha value other than 255 was set before (see [Attribute.setAlpha\(int\)](#)), filling is done with the according opacity.
- Any previously set adjustments for graphics are taken into account (see [Attribute.setAdjustmentForGraphics\(int\)](#)).
- The `graphicIndex` range from 0xFFE0 to 0xFFFF is reserved for internal use!
- When inverse mode is on, the graphic is shown in inverse mode.
- Graphics with transparencies are always drawn transparent, regardless of the transparent mode (see [Attribute.setTransparentModeEnabled\(boolean\)](#)).
- Monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.
- While graphics with a color depth of 8 bit occupy less space in the memory, 16 bit graphics are drawn faster as there is no need for converting calculations.
- To show a graphic from the MicroSD card, the file has to be converted into in the RII (Raw iLCD Graphics Image) format. To do that, use the "Save As" button on iLCD Manager XE's "Graphics" page and select the \*.rii filetype.
- When a graphic is stored on the MicroSD card automatically (by activating the option "Store graphics to SD card" in the "Project Settings" section of iLCD Manager XE's "Settings" page), it can still be addressed by its index or name as if they would be stored in flash memory. Graphics stored manually on the MicroSD card can only be loaded by their filenames.

**Example**

```
Graphic.displayLocalGraphic(5);
Graphic.displayLocalGraphic("GRAPHIC");
Graphic.displayLocalGraphic("DIR/FILE.RII");
```

Displays the graphic with index 1 and graphic "GRAPHIC" from the on-board flash as well as graphic "FILE.RII" from the SD card's "DIR" folder at the current cursor position.

**See also:**

[Get Graphic Info](#)  
[Control.setGraphicAlignment\(int, int, int\)](#)  
[Graphic.displayGraphicArea\(int, int, int, int, int\)](#)  
[Extra.setGraphicOffset\(int\)](#)  
[Extra.setGraphicNamePrefix\(String\)](#)

[Extra.setGraphicNameSuffix\(String\)](#)  
[Attribute.setAlpha\(int\)](#)  
[Attribute.setAdjustmentForGraphics\(int\)](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)

---

Package [ilcd](#)

## Class [Graphic](#)

### **Public Method eraseAnimationFrameArea**

```
static void eraseAnimationFrameArea(int animLoc)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control

Erases the area covered by the current frame of the animated image.

#### Note

- By default, the frame area is filled with the current background color. If any background was assigned to the animation control however, the frame area is restored to the assigned background (refer to [Graphic.setAnimationBackgroundColor\(int, int\)](#), [Graphic.setAnimationBackgroundFrame\(int, int\)](#), [Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#) or [Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)).
- The area may be smaller than the area of the complete image and the upper left corner of the area to be erased is given by the x/y offset of the frame.

#### Example

```
Graphic.eraseAnimationFrameArea(2);
```

Erases the area on the screen covered by the current frame of animation 2 referenced by the animation control.

#### See also:

[Graphic.eraseAnimationImageArea\(int\)](#)

---

**Package [ilcd](#)****Class [Graphic](#)****Public Method eraseAnimationImageArea**

```
static void eraseAnimationImageArea(int animLoc)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control

Erases the area covered by the complete animated image.

**Note**

- By default, the animation area is filled with the current background color. If any background was assigned to the animation control however, the animation area is restored to the assigned background (refer to [Graphic.setAnimationBackgroundColor\(int, int\)](#), [Graphic.setAnimationBackgroundFrame\(int, int\)](#), [Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#) or [Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)).

**Example**

```
Graphic.eraseAnimationImageArea(2);
```

Erases the area on the screen covered by the complete animation 2 referenced by the animation control.

**See also:**

[Graphic.eraseAnimationFrameArea\(int\)](#)

---

**Package [ilcd](#)****Class [Graphic](#)****Public Method loadAnimatedGraphics**

```
static void loadAnimatedGraphics(int animLoc, int animGraphicIndex)
```

**Throws:**

- [ILCDEException](#)

```
static void loadAnimatedGraphics(int animLoc, String  
animGraphicNameOrFilename)
```

**Throws:**

- [ILCDEException](#)

**Description:****by index:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
animGraphicIndex	1 ... max. graphic index	index of the animated graphic

**by name or filename:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
animGraphicNameOrFilename	ASCII chars (0x01 .. 0xFF) or DOS filename (8.3 format)	name of the graphic or name and path of the graphics file

Loads an animated graphic to the animation control.

**Note**

- This command does not draw the animation, it just loads it to the animation control.
- The animation is placed at the current cursor position, graphic alignment is taken into account (refer to [Control.setGraphicAlignment\(int, int, int\)](#)).
- Positioning, alignment and cropping is done with respect to the currently selected viewport (refer to [Control.selectViewport\(int\)](#)). All re-positioning (refer to [Graphic.setAnimationCoordinatesToCursorPosition\(int\)](#) and [Graphic.setAnimationCoordinatesToXY\(int, int, int\)](#)) will be interpreted in this viewport too.
- The current background color (refer to [Attribute.setBackgroundColor\(int\)](#)) is stored in the animation control and subsequently used for frame removal if the disposal method [Graphic.removeAnimationBackground\(int\)](#) is used.
- When addressing by index, the graphic offset (see [Extra.setGraphicOffset\(int\)](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Extra.setGraphicNamePrefix\(String\)](#)) and suffix (refer to [Extra.setGraphicNameSuffix\(String\)](#)) are taken into account.
- The *animGraphicIndex* range from 0xFFE0 to 0xFFFF is reserved for internal use!
- If the animation is intended to be run in both directions (e.g. for a gauge), a background has to be assigned to the animation engine (refer to [Graphic.setAnimationBackgroundColor\(int, int\)](#), [Graphic.setAnimationBackgroundFrame\(int, int\)](#), [Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#) or [Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)).
- To convert an animated image to a Raw iLCD Graphics Image, use the "Save As" button on iLCD Manager XE's "Graphics" page and select the \*.rii filetype.
- Animated graphics stored manually on the MicroSD card can be loaded by their filenames, but not by their graphic names.
- Find general information about animated graphics in class [Graphic](#).

**Example**

```
Graphic.loadAnimatedGraphics(1, 4);
```



```
Graphic.loadAnimatedGraphics(2, "ANIMATION");
Graphic.loadAnimatedGraphics(3, "ANIM.RII");
```

Loads the animated graphic with index 4 into animation control #1, graphic "ANIMATION" into control #2 and graphic "ANIM.RII" from the SD card's root folder into control #3.

#### See also:

[Control.getGraphicInfo\(int\)](#)  
[Control.setGraphicAlignment\(int, int, int\)](#)  
[Extra.setGraphicOffset\(int\)](#)  
[Extra.setGraphicNameSuffix\(String\)](#)  
[Extra.setGraphicNamePrefix\(String\)](#)  
[Graphic.displayLocalGraphic\(int\)](#)  
[Graphic.startOrRestartAnimation\(int\)](#)  
[Graphic.Move Animation To Frame\(int, int, boolean\)](#)

## Package [ilcd](#)

### Class [Graphic](#)

#### Public Method [moveAnimationToFrame](#)

```
static void moveAnimationToFrame(int animLoc, int frame, int direction)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>animLoc</code>	0 ... 7	index of the animation control
<code>frame</code>	0 ... number of frames - 1	frame of the animation
<code>direction</code>	0 ... 1	determines whether the animation runs forwards (0) or backwards (1)

Start the animated graphic loaded to the animation control referred by `animLoc` and stop again when the specified `frame` is reached. This can be done in both directions, dependet on the value of `direction`.

#### Note

- Generally, animated gifs are not able to run in both directions. To run an animation backwards, a background must be assigned to the animation engine (refer to [Graphic.setAnimationBackgroundColor\(int, int\)](#), [Graphic.setAnimationBackgroundFrame\(int, int\)](#), [Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#) or [Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)).

- Specifying a frame of 0xFFFF will let the animation run indefinitely (same as [Graphic.startOrRestartAnimation\(int\)](#)) in forward or backward direction.
- If the frame *frame* is currently showing, the animation stays as it is.
- If the animation engine has never been drawn before it will start from the first (running forward) or last frame (running backwards).
- Find general information about animated graphics in chapter [Graphic](#).

### Example

```
Graphic.moveAnimationToFrame(3, 5, 1);
```

Starts the animation loaded into animation control #3 in backwards direction and stops at frame #5.

### See also:

[Graphic](#)  
[Graphic.loadAnimatedGraphics\(int, int\)](#)  
[Graphic.startOrRestartAnimation\(int\)](#)  
[Control.getGraphicInfo\(int\)](#)

---

## Package [ilcd](#)

## Class [Graphic](#)

### Public Method [removeAnimationBackground](#)

```
static void removeAnimationBackground(int animLoc)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control

Removes any previously assigned background from the animation engine *animLoc*.

### Note

- Find general information about animated graphics in class [Graphic](#).

### Example

```
Graphic.removeAnimationBackground(4);
```

This will remove any previously set background from the animation control #4.

**See also:**

[Graphic.setAnimationBackgroundColor\(int, int\)](#)  
[Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#)  
[Graphic.setAnimationBackgroundFrame\(int, int\)](#)  
[Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)

---

**Package [ilcd](#)****Class [Graphic](#)****Public Method [resumeAnimationEngine](#)**

```
static void resumeAnimationEngine ()
```

**Throws:**

- [ILCDEException](#)

**Description:**

Resumes the animation engine when previously stopped via [Graphic.suspendAnimationEngine\(\)](#).

**Note**

- This method is especially useful when used in conjunction with the [Graphic.suspendAnimationEngine\(\)](#) method to conveniently stop and resume all animations on the screen.
- By default, the animation engine is running. Hence it will be automatically started on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- Find general information about animated graphics in class [Graphic](#).

**Example**

```
Graphic.resumeAnimationEngine ();
```

Resumes the animation engine and all animations at the frame they have stopped.

**See also:**

[Graphic.suspendAnimationEngine\(\)](#)

---

**Package [ilcd](#)****Class [Graphic](#)****Public Method [setAnimationBackgroundColor](#)**

```
static void setAnimationBackgroundColor (int animLoc, int colorValue)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>animLoc</code>	0 ... 7	index of the animation control
<code>colorValue</code>	0x000000 ... 0xFFFFFFFF	color value for the animation background

Assigns the background color `colorValue` to the animation engine `animLoc`.

**Note**

- When a background color is assigned to an animation control, every frame is filled with the specified background color before the next one is drawn.
- If the animation was never drawn before, the whole animation area is filled with the background color before the first frame is drawn.
- Restoring frames to background allows an animation to run backwards (refer to [Graphic.moveAnimationToFrame\(int, int, int\)](#)), but slows it down a bit.
- The background is filled with the assigned background color when using the methods [Graphic.eraseAnimationImageArea\(int\)](#) or [Graphic.eraseAnimationFrameArea\(int\)](#) as well.
- Find general information about animated graphics in class [Graphic](#).

**Example**

```
Graphic.setAnimationBackgroundColor(5, 0xFFFF00);
```

This will assign a yellow color to the animation control #5.

**See also:**

[Graphic.removeAnimationBackground\(int\)](#)  
[Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#)  
[Graphic.setAnimationBackgroundFrame\(int, int\)](#)  
[Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)  
[Graphic.eraseAnimationImageArea\(int\)](#)  
[Graphic.eraseAnimationFrameArea\(int\)](#)

**Package [ilcd](#)****Class [Graphic](#)****Public Method [setAnimationBackgroundFrame](#)**

```
static void setAnimationBackgroundFrame(int animLoc, int frame)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
frame	0 ... number of frames - 1	background frame of the animation

Specifies the frame *frame* as background for the animation engine *animLoc*.

**Note**

- When a background frame is assigned to an animation control, every frame is restored to the underlying part of the background before the next one is drawn.
- As soon as a background frame is assigned, this frame will be skipped in running animations. It can still be set manually though (refer to [Graphic.stopAnimationAndSetFrameNumber\(int, int\)](#)).
- If the background frame doesn't cover the entire animation area, only the part where the areas of the background and the previous frame overlap will be restored.
- If the animation was never drawn before, the whole animation area is filled with the background frame before the first frame is drawn.
- Restoring frames to background allows an animation to run backwards (refer to [Graphic.moveAnimationToFrame\(int, int, int\)](#)), but slows it down a bit.
- The background is restored to the assigned background frame when using the methods [Graphic.eraseAnimationImageArea\(int\)](#) or [Graphic.eraseAnimationFrameArea\(int\)](#) as well.
- Find general information about animated graphics in class [Graphic](#).

**Example**

```
Graphic.setAnimationBackgroundFrame(7, 12);
```

This will specify frame #12 as the background frame for animation control #5.

**See also:**

[Graphic.removeAnimationBackground\(int\)](#)  
[Graphic.setAnimationBackgroundColor\(int, int\)](#)  
[Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#)  
[Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)  
[Graphic.eraseAnimationImageArea\(int\)](#)  
[Graphic.eraseAnimationFrameArea\(int\)](#)

**Package [ilcd](#)****Class [Graphic](#)****Public Method [setAnimationBackgroundGraphic](#)**

```
static void setAnimationBackgroundGraphic(int animLoc, int startX, int startY, int graphicIndex)
```

**Throws:**

- [ILCDEException](#)

```
static void setAnimationBackgroundGraphic(int animLoc, int startX, int startY, String graphicNameOrFilename)
```

**Throws:**

- [ILCDEException](#)

**Description:****by index:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
startX	0 ... graphic width - 1	horizontal offset in background graphic
startY	0 ... graphic height - 1	vertical offset in background graphic
graphicIndex	0 ... max. graphic index	index of the background graphic

**by name or filename:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
startX	0 ... graphic width - 1	horizontal offset in background graphic
startY	0 ... graphic height - 1	vertical offset in background graphic
graphicNameOrFilename	ASCII chars (0x01 .. 0xFF) or DOS filename (8.3 format)	name of the graphic or name and path of the graphics file

Assigns the background graphic specified by *graphicIndex* or *graphicNameOrFilename* to the animation engine *animLoc*.

**Note**

- When a background graphic is assigned to an animation control, every frame is restored to the underlying part of the background before the next one is drawn.
- If the animation was never drawn before, the whole animation area is filled with the according part of background graphic before the first frame is drawn.
- When addressing by index, the graphic offset (see [Extra.setGraphicOffset\(int\)](#)) is taken into account.
- When addressing by name, the graphic prefix (refer to [Extra.setGraphicNamePrefix\(String\)](#)) and suffix (refer to [Extra.setGraphicNameSuffix\(String\)](#)) are taken into account.
- Background graphics can be larger than the animation area. In this case, *offsetX* and *offsetY* have to be set to the offset between the origin of the background graphic and the origin of the animation area.
- Restoring frames to background allows an animation to run backwards (refer to [Graphic.moveAnimationToFrame\(int, int, int\)](#)), but slows it down a bit. Even more so if animated or background graphic have to be loaded from SD card.

- The background is restored to the assigned background graphic when using the methods [Graphic.eraseAnimationImageArea\(int\)](#) or [Graphic.eraseAnimationFrameArea\(int\)](#) as well.
- Find general information about animated graphics in class [Graphic](#).

### Example

```
Graphic.setAnimationBackgroundGraphic(1, 0, 0, 7);
Graphic.setAnimationBackgroundGraphic(2, 10, 20, "GRAPHIC");
Graphic.setAnimationBackgroundGraphic(3, 0, 0, "DIR/FILE.RII");
```

These commands will assign:

- graphic with index #7 without offset to the animation control #1
- part of the graphic with name "GRAPHIC" starting from position 10/20 to the animation control #2
- graphic with filename "FILE.RII" from the SD card's "DIR" folder without offset to the animation control #3

### See also:

[Graphic.removeAnimationBackground\(int\)](#)  
[Graphic.setAnimationBackgroundColor\(int, int\)](#)  
[Graphic.setAnimationBackgroundFrame\(int, int\)](#)  
[Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)  
[Graphic.eraseAnimationImageArea\(int\)](#)  
[Graphic.eraseAnimationFrameArea\(int\)](#)

## Package [ilcd](#)

### Class [Graphic](#)

#### Public Method [setAnimationBackgroundScreen](#)

```
static void setAnimationBackgroundScreen(int animLoc, int posX, int posY, int screen)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
posX	0 ... display width - 1	horizontal position on background screen
posY	0 ... display height - 1	vertical position on background screen
screen	M, 0 ... number of screens - 1	index of the background screen (M = main screen)

Assigns the background screen *screen* to the animation engine *animLoc*.

### Note

- When a background screen is assigned to an animation control, every frame is restored to the underlying part of the background before the next one is drawn.
- If the animation was never drawn before, the whole animation area is filled with the content of the background screen before the first frame is drawn.
- If the animation was never drawn before, the whole animation area is filled with the content of the background screen before the first frame is drawn.
- The background can be located anywhere on the background screen. *posX* and *posY* can be used to specify the position of the background on the screen.
- Restoring frames to background allows an animation to run backwards (refer to [Graphic.moveAnimationToFrame\(int, int, int\)](#)), but slows it down a bit.
- The background is restored to the assigned screen content when using the methods [Graphic.eraseAnimationImageArea\(int\)](#) or [Graphic.eraseAnimationFrameArea\(int\)](#) as well.
- Find general information about animated graphics in class [Graphic](#).

### Example

```
Graphic.setAnimationBackgroundScreen(1, 20, 30, 2);
```

This will specify the content of screen #2 starting from cursor position 20/30 as background for the animation control #1.

Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

### See also:

[Graphic.removeAnimationBackground\(int\)](#)  
[Graphic.setAnimationBackgroundColor\(int, int\)](#)  
[Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#)  
[Graphic.setAnimationBackgroundFrame\(int, int\)](#)  
[Graphic.eraseAnimationImageArea\(int\)](#)  
[Graphic.eraseAnimationFrameArea\(int\)](#)

## Package [ilcd](#)

## Class [Graphic](#)

### Public Method [setAnimationCoordinatesToCursorPosition](#)

```
static void setAnimationCoordinatesToCursorPosition(int animLoc)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
-----------	-------	-------------



animLoc	0 ... 7	index of the animation control
---------	---------	--------------------------------

Sets the coordinates for an animated graphic stored in the animation control to the current cursor position.

#### Note

- Graphic alignment is taken into account (refer to [Control.setGraphicAlignment\(int, int, int\)](#)).
- The coordinates are assigned to the animation control independent from the currently selected viewport. They will be interpreted with respect to the origin of the viewport in which the animation was loaded (refer to [Graphic.loadAnimatedGraphics\(int, int\)](#)).
- Setting the coordinate without stopping an already running animation may cause unwanted effects.

#### Example

```
Graphic.setAnimationCoordinatesToCursorPosition(2);
```

This example sets the coordinates for the graphic referenced with index 2 by the animation control to the current cursor position.

#### See also:

[Graphic.setAnimationCoordinatesToXY\(int, int, int\)](#)

[Graphic.loadAnimatedGraphics\(int, int\)](#)

[Graphic.displayLocalGraphic\(int\)](#)

### Package [ilcd](#)

### Class [Graphic](#)

#### **Public Method [setAnimationCoordinatesToXY](#)**

```
static void setAnimationCoordinatesToXY(int animLoc, int posX, int posY)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
posX	0 ... display width - 1	horizontal position of the graphic
posY	0 ... display height - 1	vertical position of the graphic

Allows to specify coordinates for a graphic loaded into the animation control.

**Note**

- Graphic alignment is taken into account (refer to [Control.setGraphicAlignment\(int, int, int\)](#)).
- The coordinates are assigned to the animation control independent from the currently selected viewport. They will be interpreted with respect to the origin of the viewport in which the animation was loaded (refer to [Graphic.loadAnimatedGraphics\(int, int\)](#)).
- Setting the coordinate without stopping an already running animation may cause unwanted effects.

**Example**

```
Graphic.setAnimationCoordinatesToXY(0, 220, 150);
```

This example will access the animated graphic referenced in the animation control with index 0 and set its drawing coordinates to x = 220 pixels and y = 150 pixels.

**See also:**

[Graphic.loadAnimatedGraphics\(int, int\)](#)

[Graphic.displayLocalGraphic\(int\)](#)

**Package [ilcd](#)****Class [Graphic](#)****Public Method [setAnimationRepetitions](#)**

```
static void setAnimationRepetitions(int animLoc, int repeat)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
repeat	0 ... 65535	number for animation repetitions (0 = endless)

Sets the repetitions for a specific animation.

**Note**

- Normally the repetitions of an animated graphic are set via iLCD Manager XE, but the number of repetitions until the animation stops automatically can be overwritten with this method.
- A repeat value of 0 sets unlimited repetitions.

**Example**

```
Graphic.setAnimationRepetitions(0, 100);
```

This will cause the animation 0 to be repeated 100 times.

**See also:**

[Graphic.startOrRestartAnimation\(int\)](#)

[Graphic.stopAnimationAndSetFrameNumber\(int, int\)](#)

Package [ilcd](#)

Class [Graphic](#)

**Public Method startOrRestartAnimation**

```
static void startOrRestartAnimation(int animLoc)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control

Starts or restarts (if previously stopped) the animated graphic loaded to the animation control referred by *animLoc*.

**Note**

- If the animated graphic does not run endlessly (that means the number of repetitions are greater than 0), the internal repetition counter is reset before the animation is (re)started.
- If the animation control *animLoc* is already running, this command has no effect.
- By default, all animations are not running. Hence all animations are stopped on startup and by the commands [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#). The loaded animation controls (refer to [Graphic.loadAnimatedGraphics\(int, int\)](#)) as well as set backgrounds (refer to [Graphic.setAnimationBackgroundColor\(int, int\)](#), [Graphic.setAnimationBackgroundFrame\(int, int\)](#), [Graphic.setAnimationBackgroundGraphic\(int, int, int, int\)](#) or [Graphic.setAnimationBackgroundScreen\(int, int, int, int\)](#)) are not deleted by this command.
- Find general information about animated graphics in class [Graphic](#).

**Example**

```
Graphic.startOrRestartAnimation(4);
```

Starts the animation loaded into animation control 4.

**See also:**

[Graphic.setAnimationRepetitions\(int, int\)](#)

[Graphic.stopAnimationAndSetFrameNumber\(int, int\)](#)

[Graphic.stopBreakAnimation\(int\)](#)  
[Graphic.suspendAnimationEngine\(\)](#)

---

Package [ilcd](#)

## Class [Graphic](#)

### **Public Method stopAnimationAndSetFrameNumber**

```
static void stopAnimationAndSetFrameNumber(int animLoc, int frame)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
animLoc	0 ... 7	index of the animation control
frame	0 ... number of frames - 1	frame of the animation

Stops the animation referenced via *animLoc* and shows the specified *frame*.

#### Note

- If *frame* exceeds the number of existing frames or *animLoc* has never been loaded with an animated graphic before.
- The animation doesn't have to run to show a specific frame.
- The method [Graphic.setAnimationCoordinatesToXY\(int, int, int\)](#) can be used to modify the position where the frame is drawn.
- Find general information about animated graphics in class [Graphic](#).

#### Example

```
Graphic.stopAnimationAndSetFrameNumber(2, 4);
```

Stops the animation control with index 2 and displays frame 4 of the animation.

#### See also:

[Graphic.setAnimationRepetitions\(int, int\)](#)  
[Graphic.setAnimationCoordinatesToXY\(int, int, int\)](#)  
[Graphic.stopBreakAnimation\(int\)](#)  
[Graphic.suspendAnimationEngine\(\)](#)

---

Package [ilcd](#)  
Class [Graphic](#)

### Public Method `stopBreakAnimation`

```
static void stopBreakAnimation(int animLoc)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>animLoc</code>	0 ... 7, 65	index of the animation control (65 = all)

Stops the animation specified via `animLoc`.

#### Note

- When the value `65` is passed as `animLoc` parameter, than all running animations will stop.
- Animations stopped by this method can be restarted via the [Graphic.startOrRestartAnimation\(int\)](#) method exactly where they have been stopped.
- Find general information about animated graphics in class [Graphic](#).

#### Example

```
Graphic.stopBreakAnimation(2);
```

Stops the animation stored under position 2 at its current frame.

#### See also:

[Graphic.suspendAnimationEngine\(\)](#)  
[Graphic.startOrRestartAnimation\(int\)](#)  
[Graphic.resumeAnimationEngine\(\)](#)

---

Package [ilcd](#)  
Class [Graphic](#)

### Public Method `suspendAnimationEngine`

```
static void suspendAnimationEngine()
```

#### Throws:

- [ILCDEException](#)

**Description:**

Allows to stop all animated graphics at the current frame by stopping the animation engine.

**Note**

- Compared to the [Graphic.stopBreakAnimation\(int\)](#) method with parameter *65*, this method allows for a more convenient way of resuming all the animations by simple using the [Graphic.resumeAnimationEngine\(\)](#) method from the point where they have stopped.
- By default, the animation engine is running. Hence it will be automatically started on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- Using this method is highly recommended before reading the LCD's screen content via the [Draw.readScanLine\(int\)](#) method, as scrambled screen contents may be read when animations are active!
- Find general information about animated graphics in class [Graphic](#).

**See also:**

[Graphic.stopBreakAnimation\(int\)](#)  
[Graphic.resumeAnimationEngine\(\)](#)

---

**Package [ilcd](#)****Class [GraphicInfo](#)**

extends [Object](#)

The *GraphicInfo* class provide different methods to obtain specific information about a graphic, and is used together with the [Control.getGraphicInfo\(\)](#) method as the return type.

**Public Methods**

- [getColorDepth](#)
- [getGraphicIndex](#)
- [getGraphicName](#)
- [getNumberOfFrames](#)
- [getSize](#)
- [isAnimated](#)
- [isDisabled](#)
- [isTransparent](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Public Fields**

- static final byte GRAPHIC\_MAX\_NAME\_LENGTH

- static final byte OFFS\_TO\_GRAPHIC\_NAME
- 

Package [ilcd](#)

Class [GraphicInfo](#)

### Public Method [getColorDepth](#)

```
int getColorDepth()
```

#### Description:

Returns	Description
colorDepth	the depth of the graphic's color

The `getColorDepth()` method is used to obtain the color depth of a graphic.

#### Example

```
int graphicDepth = Control.getGraphicInfo(0).getColorDepth();  
Draw.writeText(" graphic depth: " + graphicDepth);
```

#### See also:

[Control.getGraphicInfo\(int\)](#)

---

Package [ilcd](#)

Class [GraphicInfo](#)

### Public Method [getGraphicIndex](#)

```
int getGraphicIndex()
```

#### Description:

Returns	Description
graphicIndex	the index of a graphic

The `getGraphicIndex()` method is used to obtain the index of a graphic.

#### Example

```
int graphicIndex = Control.getGraphicInfo(0).getGraphicIndex();  
Draw.writeText(" graphic index: " + graphicIndex);
```

**See also:**[Control.getGraphicInfo\(int\)](#)

---

Package [ilcd](#)Class [GraphicInfo](#)**Public Method `getGraphicName`**

```
String getGraphicName()
```

**Description:**

Returns	Description
graphicName	the name of a graphic

The `getGraphicName()` method is used to obtain the name of a graphic.

**Example**

```
String graphicName = Control.getGraphicInfo(0).getGraphicName();  
Draw.writeText(" graphic index: " + graphicName);
```

**See also:**[Control.getGraphicInfo\(int\)](#)

---

Package [ilcd](#)Class [GraphicInfo](#)**Public Method `getNumberOfFrames`**

```
int getNumberOfFrames()
```

**Description:**

Returns	Description
numberOfFrames	the numbers of frames

The `getNumberOfFrames()` method is used to obtain the number of frames which is provided by an animated graphic.



## Example

```
int numberOfFrames = Control.getGraphicInfo(1).getNumberOfFrames();
Draw.writeText(" number of frames: "+ numberOfFrames);
```

## See also:

[Control.getGraphicInfo\(int\)](#)

---

## Package [ilcd](#)

## Class [GraphicInfo](#)

### Public Method [getSize](#)

```
Size getSize()
```

#### Description:

Return s	Description
<a href="#">Size</a>	the size of a graphic

The `getSize()` method is used to obtain the size of a graphic. The `Size` return type delivers the height and width values of a graphic.

## Example

```
Size graphicSize = Control.getGraphicInfo(1).getSize();
Draw.writeText(" height of the graphic: " + graphicSize.getHeight() + "\r"
               + " width of the graphic:  " + graphicSize.getWidth());
```

## See also:

[Control.getGraphicInfo\(int\)](#)

---

## Package [ilcd](#)

## Class [GraphicInfo](#)

### Public Method [isAnimated](#)

```
boolean isAnimated()
```

**Description:**

Returns	Description
isAnimated	whether the graphic is animated or not

This method returns true if the graphic is animated.

**Example**

```
if (Control.getGraphicInfo(1).isAnimated())
{
    Draw.writeText(" This graphic supports animation! ");
}
else
{
    Draw.writeText(" This graphic does not support animation! ");
}
```

**See also:**

[Control.getGraphicInfo\(int\)](#)

---

**Package [ilcd](#)****Class [GraphicInfo](#)****Public Method isDisabled**

```
boolean isDisabled()
```

**Description:**

Returns	Description
isDisabled	whether the graphic is disabled or not

This method returns true if the graphic is disabled.

**Example**

```
if (Control.getGraphicInfo(1).isDisabled())
{
    Draw.writeText(" This graphic is disabled! ");
}
else
{
    Draw.writeText(" This graphic is not disabled! ");
}
```

**See also:**[Control.getGraphicInfo\(int\)](#)

---

**Package [ilcd](#)****Class [GraphicInfo](#)****Public Method [isTransparent](#)**

```
boolean isTransparent()
```

**Description:**

Returns	Description
isTransparent	whether the graphic is transparent or not

This method returns true if the graphic is transparent.

**Example**

```
if (Control.getGraphicInfo(1).istransparent())
{
    Draw.writeText(" This graphic is transparent! ");
}
else
{
    Draw.writeText(" This graphic is not transparent! ");
}
```

**See also:**[Control.getGraphicInfo\(int\)](#)

---

**Package [ilcd](#)****Class [ILCDEEPROMException](#)**

extends [ILCDEEPROMException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDEEPROMException* class is a subclass from the [ILCDEEPROMException](#) class. An exception can be thrown during an EEPROM access operation.

**Public Constructors**

- [ILCDEEPROMException](#)

**Methods inherited from [ilcd.ILCDEException](#)**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

**Package [ilcd](#)****Class [ILCDEEPROMException](#)****Public Constructor ILCDEEPROMException**

```
ILCDEEPROMException(int code)
```

**Description:**

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

**See also:**

[Common](#)

---

**Package [ilcd](#)****Class [ILCDEException](#)**

extends [Exception](#) → [Throwable](#) → [Object](#)

An *ILCDEException* is thrown when an iLCD specific error occurs. For this reason, the iLCD provides several iLCD error codes which are specified in the *Common* class. The [getErrorCode\(\)](#) method is used to process an error code in an application.

The *ILCDEException* class is a subclass of the [java.lang.Exception](#) class. Other iLCD specific exception classes are subclasses of the *ILCDEException* class. The [ILCDEExceptionFactory](#) determines which specific exception might be thrown when an exception occurs.

An iLCD specific exception can be caught with a *try-catch* statement.

### Public Constructors

- [ILCDEException](#)

### Public Methods

- [getErrorCode](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

## Package [ilcd](#)

## Class [ILCDEException](#)

### Public Constructor [ILCDEException](#)

```
ILCDEException(int code)
```

```
ILCDEException(String description)
```

### Description:

#### constructor [ILCDEException](#)(int code):

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

**constructor `ILCDException(String description)`:**

Parameter	Description
<code>description</code>	detailed message

The *description* parameter is stored for later use.

**See also:**

[Common](#)

---

**Package [ilcd](#)**

**Class [ILCDException](#)**

**Public Method `getErrorCode`**

```
int getErrorCode()
```

**Description:**

Returns	Description
<code>errorCode</code>	specific error code of the occurred exception

This method is used to retrieve the error code of the occurred exception.

**Example**

```
try
{
    ...
}
catch(ILCDException exc)
{
    errorCode = exc.getErrorCode();
}
```

An exception is caught in the example above and the error code is returned by the *getErrorCode()* method.

**See also:**

[Common](#)

## Package [ilcd](#)

### Class **ILCDExceptionFactory**

extends [Object](#)

The *ILCDException* class is a subclass of the [java.lang.Exception](#) class. Other iLCD specific exception classes are subclasses of the *ILCDException* class. The [ILCDExceptionFactory](#) determines which specific exception might be thrown when an exception occurs.

#### Public Constructors

- [ILCDExceptionFactory](#)

#### Public Methods

- [createExceptionFromCode](#)
- [createGeneralException](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

#### Description:

This method constructs a new *ILCDExceptionFactory* object.

#### Package [ilcd](#)

### Class **[ILCDExceptionFactory](#)**

#### **Public Constructor ILCDExceptionFactory**

```
ILCDExceptionFactory()
```

#### Description:

Creates a new *ILCDExceptionFactory* object.

#### See also:

[ILCDException](#)

---

Package [ilcd](#)

## Class [ILCDExceptionFactory](#)

### Public Method `createExceptionFromCode`

```
static ILCDException createExceptionFromCode(int code)
```

#### Description:

Parameter	Description
code	detailed code

The `createExceptionFromCode(int)` method is used to determine a specific iLCD exception based on the received `code`.

Returns	Description
ILCDException	ILCDException object

This method returns an `ILCDException` specific object.

#### See also:

[Common](#)

---

Package [ilcd](#)

## Class [ILCDExceptionFactory](#)

### Public Method `createGeneralException`

```
static ILCDException createGeneralException(String description)
```

#### Description:

Parameter	Description
description	detailed message

The `createGeneralException(String)` method is used to specify a detailed message.

Returns	Description
ILCDException	ILCDException object



This method returns an *ILCDException* specific object.

**See also:**

[Common](#)

---

**Package [ilcd](#)**

**Class [ILCDFSFileSystemException](#)**

extends [ILCDException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDFSFileSystemException* class is a subclass from the [ILCDException](#) class. An exception can be thrown during an file system access operation.

**Public Constructors**

- [ILCDFSFileSystemException](#)

**Methods inherited from [ilcd.ILCDException](#)**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

**Package [ilcd](#)**

**Class [ILCDFSFileSystemException](#)**

**Public Constructor ILCDFSFileSystemException**

```
ILCDFSFileSystemException(int code)
```

**Description:**

Parameter	Description
code	detailed code

The `code` parameter is stored for later use and can be retrieved with the `getErrorCode()` method.

**See also:**

[Common](#)

---

**Package [ilcd](#)****Class [ILCDFlashException](#)**

extends [ILCDEException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The `ILCDFlashException` class is a subclass from the [ILCDEException](#) class. An exception can be thrown during a flash access operation.

**Public Constructors**

- [ILCDFlashException](#)

**Methods inherited from [ilcd.ILCDEException](#)**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
-

**Package [ilcd](#)****Class [ILCDFSFlashException](#)****Public Constructor [ILCDFSFlashException](#)**

```
ILCDFSFlashException(int code)
```

**Description:**

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

**See also:**

[Common](#)

---

**Package [ilcd](#)****Class [ILCDGeneralException](#)**

extends [ILCDException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDGeneralException* class is a subclass from the [ILCDException](#) class. This class is a general class of exceptions and is used to specify a user defined error message.

**Public Constructors**

- [ILCDGeneralException](#)

**Methods inherited from [ilcd.ILCDException](#)**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)

- [notifyAll](#)
- 

Package [ilcd](#)

## Class [ILCDGeneralException](#)

### Public Constructor [ILCDGeneralException](#)

```
ILCDGeneralException(String description)
```

#### Description:

Parameter	Description
description	detailed message

The *ILCDGeneralException*(String) method is used to specify a detailed message.

#### See also:

[Common](#)

---

Package [ilcd](#)

## Class [ILCDGraphException](#)

extends [ILCDException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDGraphException* class is a subclass from the [ILCDException](#) class. An exception can be thrown during the processing of a graphical operation.

### Public Constructors

- [ILCDGraphException](#)

### Methods inherited from [ilcd.ILCDException](#)

- [getErrorCode](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

## Package [ilcd](#)

### Class [ILCDGraphException](#)

#### Public Constructor [ILCDGraphException](#)

```
ILCDGraphException(int code)
```

#### Description:

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

#### See also:

[Common](#)

---

## Package [ilcd](#)

### Class [ILCDMacroException](#)

extends [ILCDException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDMacroException* class is a subclass from the [ILCDException](#) class. An exception can be thrown during an invalid iLCD specific macro operation.

#### Public Constructors

- [ILCDMacroException](#)

#### Methods inherited from [ilcd.ILCDException](#)

- [getErrorCode](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

### Package [ilcd](#)

### Class [ILCDMacroException](#)

#### Public Constructor [ILCDMacroException](#)

```
ILCDMacroException(int code)
```

#### Description:

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

#### See also:

[Common](#)

---

### Package [ilcd](#)

### Class [ILCDMemoryException](#)

extends [ILCDException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDMemoryException* class is a subclass from the [ILCDException](#) class. An exception can be thrown during an LCD memory operation.

## Public Constructors

- [ILCDMemoryException](#)

## Methods inherited from [ilcd.ILCDException](#)

- [getErrorCode](#)

## Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

## Package [ilcd](#)

## Class [ILCDMemoryException](#)

### Public Constructor ILCDMemoryException

```
ILCDMemoryException(int code)
```

#### Description:

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

#### See also:

[Common](#)

---

**Package [ilcd](#)****Class [ILCDSDCException](#)**

extends [ILCDEException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDSDCException* class is a subclass from the [ILCDEException](#) class. An exception can be thrown during an SD card operation.

**Public Constructors**

- [ILCDSDCException](#)

**Methods inherited from [ilcd.ILCDEException](#)**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [ilcd](#)****Class [ILCDSDCException](#)****Public Constructor ILCDSDCException**

```
ILCDSDCException(int code)
```

**Description:**

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.



**See also:**[Common](#)

---

**Package [ilcd](#)****Class [ILCDTouchEvent](#)**extends [ILCDException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The *ILCDTOUCHEXception* class is a subclass from the [ILCDException](#) class. An exception can be thrown during an touch operation.

**Public Constructors**

- [ILCDTouchEvent](#)

**Methods inherited from [ilcd.ILCDException](#)**

- [getErrorCode](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

**Package [ilcd](#)****Class [ILCDTouchEvent](#)****Public Constructor [ILCDTouchEvent](#)**

```
ILCDTouchEvent(int code)
```

**Description:**

Parameter	Description
-----------	-------------

code	detailed code
------	---------------

The `code` parameter is stored for later use and can be retrieved with the `getErrorCode()` method.

**See also:**

[Common](#)

---

## Package [ilcd](#)

### Class `ILCDUnknownRuntimeException`

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

The `ILCDUnknownRuntimeException` class is a subclass from the [java.lang.RuntimeException](#) class and is used in several iLCD classes.

#### Public Constructors

- [ILCDUnknownRuntimeException](#)

#### Public Methods

- [getErrorCode](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
-

Package [ilcd](#)

## Class [ILCDUnknownRuntimeException](#)

### Public Constructor [ILCDUnknownRuntimeException](#)

```
ILCDUnknownRuntimeException(int code)
```

#### Description:

Parameter	Description
code	detailed code

The *code* parameter is stored for later use and can be retrieved with the *getErrorCode()* method.

#### See also:

[Common](#)

---

Package [ilcd](#)

## Class [ILCDUnknownRuntimeException](#)

### Public Method [getErrorCode](#)

```
int getErrorCode()
```

#### Description:

Returns	Description
errorCode	specific error code of the occurred exception

This method is used to retrieve the error code of the occurred unknown runtime exception.

#### Example

```
try
{
    ...
}
catch(ILCDUnknownenRuntimeException exc)
{
    errorCode = exc.getErrorCode();
}
```

A unknown runtime exception is caught in the example above and the error code is returned by the *getErrorCode()* method.

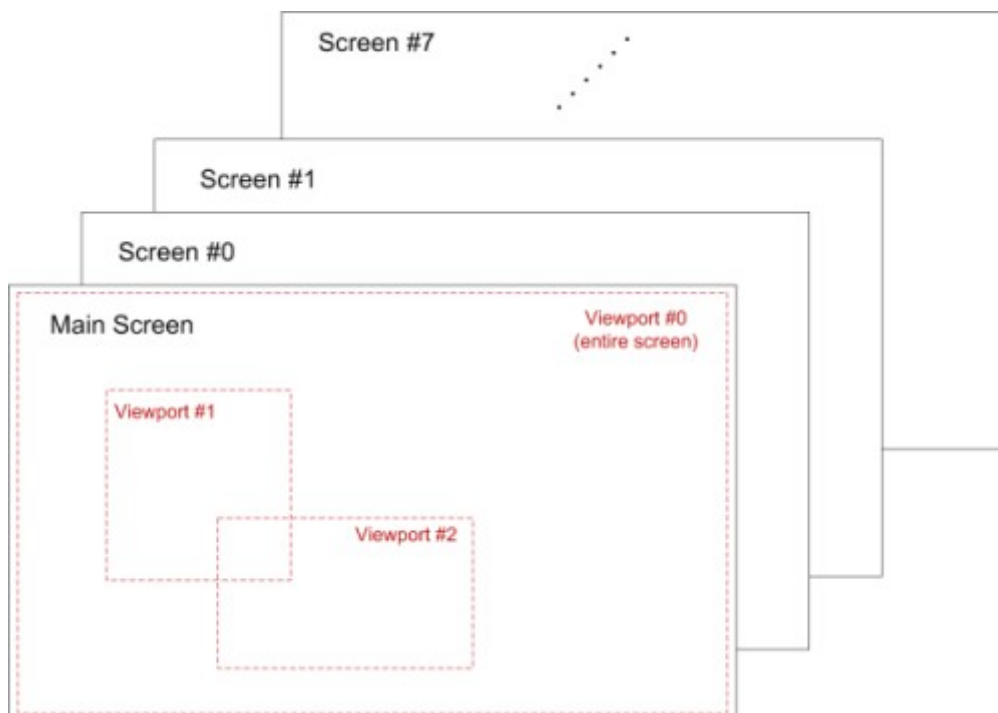
**See also:**[Common](#)**Package [ilcd](#)****Class Memory**extends [Object](#)

Color iLCD controllers DPC3080 and higher contain some RAM for storing complete screen contents. This allows the user to select a screen memory area to draw to (draw screen) and a different one that is displayed (view screen). When changing the view screen, the previously drawn content is displayed instantly without any visible buildup. This can be used in turn to create multiple screens or pages with minimum switching delays.

Color iLCDs need a rather high amount of memory to store a screen (width \* height \* 2 bytes), so these models use an external RAM of typical 8 MByte to store screens into. When using a 1024 x 600 pixel LCD, four user accessible screens are available. For all other models with iLCD controller DPC3080 and higher, the number of screens is limited to a maximum of eight. To retrieve the number of available screens, the [Memory.getNumberOfScreens\(\)](#) method can be used.

iLCDs with the DPC3050 controller are not equipped with any external RAM, so the method [Memory.getNumberOfScreens\(\)](#) will return 0 for these models. Therefore, the methods in this chapter except [Memory.copyScreenArea\(\)](#) will always return `0x15` which means the command is not acknowledge.

Every iLCD additionally provides a main screen, which can be addressed by the screen index 'M' (ASCII code 0x4D). The main viewport - which represents the entire screen area - of each screen is defined as 0.



Not supported by: DPC3050, DPC3020, DPC2060, DPC10xx

## Note

- It's not possible to instantiate this class. The methods in this class are static.

## Public Methods

- [copyScreenArea](#)
- [copyScreenFrom](#)
- [copyScreenTo](#)
- [getDrawScreenParameters](#)
- [getNumberOfScreens](#)
- [getViewScreenParameters](#)
- [invertScreen](#)
- [paintScreenFrom](#)
- [restoreCursorAttributesFromMemory](#)
- [saveCursorAttributesToMemory](#)
- [scrollDownScreen](#)
- [scrollLeftScreen](#)
- [scrollRightScreen](#)
- [scrollUpScreen](#)
- [setDrawScreen](#)
- [setHeightOfScreen](#)
- [setViewScreen](#)
- [setWidthOfScreen](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

## Package [ilcd](#)

## Class [Memory](#)

### Public Method [copyScreenArea](#)

```
static void copyScreenArea (int width, int height, int screen, int posX,  
int posY)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Paramete	Range	Description
----------	-------	-------------

r		
width	1 ... display width	width of the area to copy
height	1 ... display height	height of the area to copy
screen	M, 0 ... number of screens - 1	index of the screen to copy to (M = main screen)
posX	0 ... display width - 1	horizontal position to copy the content to
posY	0 ... display height - 1	vertical position to copy the content to

Copies the content of an area of the currently active draw screen starting from the current cursor position.

### Note

- When the selected area exceeds the screen, it is truncated accordingly.
- The *screen* (index) can also be set to the currently active draw screen, allowing copying an area to a different position on the same screen.
- Even if the target screen was never selected as draw screen before, it is possible to select it as view screen after copying an area to it.

### Example

```
Memory.copyScreenArea(25, 45, 'M', 500, 250);
```

This example copies an area of 25 pixels width and 45 pixels height (starting from the current cursor position) from the currently active draw screen to main screen at coordinates x = 500 and y = 250 pixels.

### See also:

[Memory](#)  
[Memory.copyScreenTo\(int\)](#)  
[Memory.copyScreenFrom\(int\)](#)  
[Memory.paintScreenFrom\(int\)](#)  
[Memory.setViewScreen\(int\)](#)

## Package **ilcd**

### Class **Memory**

#### Public Method **copyScreenFrom**

```
static void copyScreenFrom(int screen)
```

#### Throws:

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>screen</code>	M, 0 ... number of screens - 1	index of the screen to copy from (M = main screen)

Restores the content of a screen specified by the index `screen` to the current draw screen.

**Note**

- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is ignored (different to [Memory.paintScreenFrom\(int\)](#)).
- This method can be used to restore a screen previously copied with [Memory.copyScreenTo\(int\)](#).

**Example**

```
Memory.copyScreenFrom(2);
```

Provided the screen with index 2 was activated, its contents is copied to the active draw screen.

**See also:**

[Memory](#)  
[Memory.copyScreenTo\(int\)](#)  
[Memory.paintScreenFrom\(int\)](#)  
[Memory.setDrawScreen\(int\)](#)  
[Memory.setViewScreen\(int\)](#)

**Package [ilcd](#)****Class [Memory](#)****Public Method [copyScreenTo](#)**

```
static void copyScreenTo(int screen)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>screen</code>	M, 0 ... number of screens - 1	index of the screen to copy content to (M = main screen)

Captures the current draw screen's content and copies it to the screen referenced by the index `screen`.

**Note**

- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is ignored.
- When capturing to a screen that has been cropped with [Memory.setHeightOfScreen\(int, int\)](#) or [Memory.setWidthOfScreen\(int, int\)](#), the width and height of this screen are restored to the full display size and it can be selected as view or draw screen again.
- [Memory.copyScreenFrom\(int\)](#) can be used to restore a screen copied with this command.

**Example**

```
Memory.copyScreenTo(2);
```

This example copies the content of the currently active draw screen to screen 2.

**See also:**

[Memory](#)

[Memory.copyScreenFrom\(int\)](#)

Package [ilcd](#)

Class [Memory](#)

**Public Method [getDrawScreenParameters](#)**

```
static ScreenParameters getDrawScreenParameters()
```

**Description:**

Returns	Description
static <a href="#">ScreenParameters</a>	contains parameters of the currently active draw screen

The [getDrawScreenParameters\(\)](#) method returns the parameters of the currently active draw screen.

**Example**

```
ScreenParameters scrPara = Memory.getDrawScreenParameters();
int index                = scrPara.getScreenIndex();
int viewportOrientation  = scrPara.getViewportOrientation();
int x                    = scrPara.getX();
```

Alternatively a value can be obtained with a single call:

```
int x = Memory.getDrawScreenParameters().getX();
```

**See also:**

[Memory](#)

[Memory.setDrawScreen\(int\)](#)



[Memory.setViewScreen\(int\)](#)  
[Memory.getViewScreenParameters\(\)](#)  
[Attribute](#)

---

Package [ilcd](#)  
Class [Memory](#)

**Public Method [getNumberOfScreens](#)**

```
static int getNumberOfScreens ()
```

**Throws:**

- [ILCDEException](#)

**Description:**

Returns	Description
number	number of screens

Returns the number of available screens excluding the main screen.

**Note**

- Before calling any of the screen memory related methods, the number of available screens should be retrieved.
- The number returned may be 0 when using large LCDs and, if so, the screen memory related methods cannot be used then.
- The number of maximum available screens is limited to 8 regardless of the controller RAM available.

**Example**

```
int number = Memory.getNumberOfScreens ();
```

**See also:**

[Memory](#)  
[Memory.setViewScreen\(int\)](#)  
[Memory.setDrawScreen\(int\)](#)

---

Package [ilcd](#)  
Class [Memory](#)

**Public Method [getViewScreenParameters](#)**

```
static ScreenParameters getViewScreenParameters ()
```

**Description:**

Returns	Description
static <a href="#">ScreenParameters</a>	index of the currently active view screen

The `getViewScreenParameters()` method returns the parameters of the currently active view screen.

**Note**

- The active viewport index is only relevant when the according screen is activated as draw screen. In this case, the reported viewport will be active immediately after switching draw screens.

**Example**

```
ScreenParameters scrPara = Memory.getViewScreenParameters();
int index             = scrPara.getScreenIndex();
int viewportOrientation = scrPara.getViewportOrientation();
int x                 = scrPara.getX();
```

Alternatively a value can be obtained with a single call:

```
int x = Memory.getViewScreenParameters().getX();
```

**See also:**

[Memory](#)  
[Memory.setViewScreen\(int\)](#)  
[Memory.setDrawScreen\(int\)](#)  
[Memory.getDrawScreenParameters\(\)](#)

**Package [ilcd](#)****Class [Memory](#)****Public Method [invertScreen](#)**

```
static void invertScreen(int screen)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to invert (M = main screen)

Inverts the contents of the screen indexed by *screen*.

### Example

```
Memory.invertScreen(2);
```

Provided the screen with index 2 was activated, its contents will be inverted.

### See also:

[Memory](#)  
[Draw.invertDisplay\(\)](#)

Package [ilcd](#)

## Class [Memory](#)

### Public Method [paintScreenFrom](#)

```
static void paintScreenFrom(int screen)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to paint from (M = main screen)

Paints the content of a screen specified by the index *screen* to the current draw screen at the current cursor position.

### Note

- This method is only available when the main viewport (index 0) of the active draw screen is activated.
- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is taken into account (different to [Memory.copyScreenFrom\(int\)](#)).

### Example

```
Memory.paintScreenFrom(2);
```

Provided the screen with index 2 was activated, its contents is copied to the current cursor position on the active draw screen.

**See also:**[Memory](#)[Memory.copyScreenTo\(int\)](#)[Attribute.setInverseMode\(boolean\)](#)[Memory.setDrawScreen\(int\)](#)[Memory.setViewScreen\(int\)](#)Package [ilcd](#)Class [Memory](#)**Public Method restoreCursorAttributesFromMemory**

```
static void restoreCursorAttributesFromMemory(int index)
```

**Throws:**

- [ILCDException](#)

**Description:**

Parameter	Range	Description
index	0 ... 7	index of the storage location

Restores cursor position, viewport and attributes (see class [Attribute](#)) from memory position *index*.

**Example**

```
Memory.restoreCursorAttributesFromMemory(3);
```

Restores the cursor and attributes from the memory at the *index* 3.

**See also:**[Memory](#)[saveCursorAttributesToMemory\(int\)](#)Package [ilcd](#)Class [Memory](#)**Public Method saveCursorAttributesToMemory**

```
static void saveCursorAttributesToMemory(int index)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
index	0 ... 7	index of the storage location

Saves current cursor position, viewport and attributes (see class [Attribute](#)) to memory position *index*.

**Example**

```
Memory.saveCursorAttributesToMemory(2);
```

Saves current cursor and attributes to the memory at the *index* 2.

**See also:**

[Memory](#)  
[Memory.restoreCursorAttributesFromMemory\(\)](#)

**Package [ilcd](#)****Class [Memory](#)****Public Method [scrollDownScreen](#)**

```
static void scrollDownScreen(int screen, int scrolly)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
scrolly	0 ... display height - 1	vertical distance for the screen to scroll down

Scrolls the screen with index *screen* *scrolly* pixels down.

**Note**

- The new rows at the top are drawn in the current background color.

- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is taken into account accordingly.

### Example

```
Memory.scrollDownScreen(2, 150);
```

Provided the screen with index 2 was activated, its contents will be scrolled 150 pixels down.

### See also:

[Memory](#)

[Attribute.setInverseMode\(boolean\)](#)

[Attribute.setBackgroundColor\(int\)](#)

[Memory.scrollUpScreen\(int, int\)](#)

[Memory.scrollLeftScreen\(int, int\)](#)

[Memory.scrollRightScreen\(int, int\)](#)

## Package [ilcd](#)

### Class [Memory](#)

#### Public Method [scrollLeftScreen](#)

```
static void scrollLeftScreen(int screen, int scrollX)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>screen</code>	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
<code>scrollX</code>	0 ... display width - 1	horizontal distance for the screen to scroll left

Scrolls the screen with index `screen` `scrollX` pixels to the left.

#### Note

- The new rows at the right are drawn in the current background color.
- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is taken into account accordingly.

### Example

```
Memory.scrollLeftScreen(5, 450);
```

Provided the screen with index 5 was activated, its contents will be scrolled 450 pixels to the left.

**See also:**[Memory](#)[Attribute.setInverseMode\(boolean\)](#)[Attribute.setBackgroundColor\(int\)](#)[Memory.scrollUpScreen\(int, int\)](#)[Memory.scrollDownScreen\(int, int\)](#)[Memory.scrollRightScreen\(int, int\)](#)**Package** [ilcd](#)**Class** [Memory](#)**Public Method** [scrollRightScreen](#)

```
static void scrollRightScreen(int screen, int scrollX)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
scrollX	0 ... display width - 1	horizontal distance for the screen to scroll right

Scrolls the screen with index *screen* *scrollX* pixels to the right.

**Note**

- The new rows at the left are drawn in the current background color.
- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is taken into account accordingly.

**Example**

```
Memory.scrollRightScreen(1, 88);
```

Provided the screen with index 1 was activated, its contents will be scrolled 88 pixels to the right.

**See also:**[Memory](#)[Attribute.setInverseMode\(boolean\)](#)[Attribute.setBackgroundColor\(int\)](#)[Memory.scrollUpScreen\(int, int\)](#)[Memory.scrollDownScreen\(int, int\)](#)[Memory.scrollLeftScreen\(int, int\)](#)

---

Package [ilcd](#)

## Class [Memory](#)

### Public Method `scrollUpScreen`

```
static void scrollUpScreen(int screen, int scrolly)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>screen</code>	M, 0 ... number of screens - 1	index of the screen to be scrolled (M = main screen)
<code>scrolly</code>	0 ... display height - 1	vertical distance for the screen to scroll up

Scrolls the screen with index `screen` `scrolly` pixels up.

#### Note

- The new rows at the bottom are drawn in the current background color.
- The currently set inverse mode (see [Attribute.setInverseMode\(boolean\)](#)) is taken into account accordingly.

#### Example

```
Memory.scrollUpScreen(3, 50);
```

Provided the screen with index 3 was activated, its contents will be scrolled 50 pixels up.

#### See also:

[Memory](#)  
[Attribute.setInverseMode\(boolean\)](#)  
[Attribute.setBackgroundColor\(int\)](#)  
[Memory.scrollDownScreen\(int, int\)](#)  
[Memory.scrollLeftScreen\(int, int\)](#)  
[Memory.scrollRightScreen\(int, int\)](#)

---



**Package [ilcd](#)****Class [Memory](#)****Public Method [setDrawScreen](#)**

```
static void setDrawScreen(int screen)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the draw screen (M = main screen)

Selects the screen with index *screen* as the current draw screen.

**Note**

- Cursor position, active viewport and all attributes (see class [Attribute](#)) are restored to the state when this draw screen was lastly active.
- All subsequently issued drawing commands are sent to this screen until another one is activated. To show the results of these methods on the display, the according screen has to be activated as view screen.
- The default value for *screen* is 'M' (main screen). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Memory.setDrawScreen(0);
```

Sets screen 0 as the current draw screen.

**See also:**

[Memory](#)

[Memory.getDrawScreenParameters\(Memory.ScreenParameters\)](#)

[Memory.setViewScreen\(int\)](#)

**Package [ilcd](#)****Class [Memory](#)****Public Method [setHeightOfScreen](#)**

```
static void setHeightOfScreen(int screen, int height)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to set the height (M = main screen)
height	0 ... display height	value for the height to be set

Sets the height of the screen indexed by *screen*.

**Note**

- The height of the main screen (index M) is not modifiable.
- Width (refer to [Memory.setWidthOfScreen\(int, int\)](#)) and height of a screen are taken into account when being copied (refer to [Memory.copyScreenFrom\(int\)](#)) or painted (refer to [Memory.paintScreenFrom\(int\)](#)).
- If the height of an image is decreased the bottom part of the image will be cut.
- When cropping a screen, this screen can not be selected as draw or view screen any longer. To make a screen selectable again, the width and height have to be restored to the full display size.

**Example**

```
Memory.setHeightOfScreen(1, 52);
```

Provided the screen with index 1 was activated, its height will be set to 52 pixels.

**See also:**

[Memory](#)  
[Memory.setWidthOfScreen\(int, int\)](#)  
[Memory.copyScreenFrom\(int\)](#)  
[Memory.paintScreenFrom\(int\)](#)

**Package [ilcd](#)****Class [Memory](#)****Public Method [setViewScreen](#)**

```
static void setViewScreen(int screen)
```

**Throws:**

- [ILCDException](#)

**Description:**

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the view screen (M = main screen)

Selects the screen with index *screen* as the current view screen.

#### Note

- When changing the view screen, its content is displayed immediately.
- In case of a touchscreen event, only touch fields defined on the currently active view screen are evaluated. Where touch fields are defined is dependent on the active draw screen when the method is carried out.
- If screen *index* was never activated as a draw screen before or *index* exceeds the available screens, a runtime exception will occur.
- The default value for *screen* is 'M' (main screen). It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

#### Example

```
Memory.setViewScreen(3);
```

Sets screen 3 as view screen and shows its content on the display.

#### See also:

[Memory](#)  
[Memory.getViewScreenParameters\(\)](#)  
[Memory.setDrawScreen\(int\)](#)

Package [ilcd](#)

### Class [Memory](#)

#### Public Method [setWidthOfScreen](#)

```
static void setWidthOfScreen(int screen, int width)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
screen	M, 0 ... number of screens - 1	index of the screen to set the height (M = main screen)
width	0 ... display width	value for the width to be set

Sets the width of the screen with index *screen*.

#### Note

- The width of the main screen (index 'M') is not modifiable.

- Width and height (refer to [Memory.setHeightOfScreen\(int, int\)](#)) of a screen are taken into account when being copied (refer to [Memory.copyScreenFrom\(int\)](#)) or painted (refer to [Memory.paintScreenFrom\(int\)](#)).
- If the width of a screen is decreased, the right hand part of the screen will be cut.
- When cropping a screen, this screen can not be selected as draw or view screen any longer. To make a screen selectable again, the width and height have to be restored to the full display size.

### Example

```
Memory.setWidthOfScreen(4, 123);
```

Provided the screen with index 4 was activated, its width will be set to 123 pixels.

### See also:

[Memory](#)

[Memory.setHeightOfScreen\(int, int\)](#)

[Memory.copyScreenFrom\(int\)](#)

[Memory.paintScreenFrom\(int\)](#)

---

## Package [ilcd](#)

### Class [NetworkStatus](#)

extends [Object](#)

Instances of the class *NetworkStatus* are used as return values for the method [General.getNetworkStatus](#) in order to obtain information about the TCP/IP connection. Also see the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

### Public Methods

- [getAccessIp](#)
- [getCurrentIp](#)
- [getDeviceIp](#)
- [getDhcpIp](#)
- [getLastHttpIp](#)
- [getPasswdMode](#)
- [getStandardGateway](#)
- [getSubnetMask](#)
- [isTcpIpEnabled](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [ilcd](#)

## Class [NetworkStatus](#)

### **Public Method `getAccessIp`**

```
short[] getAccessIp()
```

#### **Description:**

Returns the *AccessIP* address that is currently set in the iLCD project settings or 0.0.0.0 if the access is unrestricted (every IP can access the device).

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

## Class [NetworkStatus](#)

### **Public Method `getCurrentIp`**

```
short[] getCurrentIp()
```

#### **Description:**

Returns the IP address with which the devices is currently connected to the network.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

## Class [NetworkStatus](#)

### **Public Method `getDeviceIp`**

```
short[] getDeviceIp()
```

#### **Description:**

Returns the static IP address that is currently defined in the iLCD project settings (if DHCP is not enabled).

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

Class [NetworkStatus](#)

**Public Method [getDhcpIp](#)**

```
short[] getDhcpIp()
```

**Description:**

Returns the IP address of the DHCP server if *DHCP enabled* is set in the iLCD project settings.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

Class [NetworkStatus](#)

**Public Method [getLastHttpIp](#)**

```
short[] getLastHttpIp()
```

**Description:**

Returns the IP address of the last HTTP access to the device.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

Class [NetworkStatus](#)

**Public Method [getPasswdMode](#)**

```
int getPasswdMode()
```

**Description:**

Returns the current password mode of the device.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

Possible values are:

Returns	Description
0	no password
1	plain text
2	CRC32

---

3	MD5
---	-----

---

Package [ilcd](#)

Class [NetworkStatus](#)

**Public Method [getStandardGateway](#)**

```
short[] getStandardGateway()
```

**Description:**

Returns the IP address of the standard gateway.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

Class [NetworkStatus](#)

**Public Method [getSubnetMask](#)**

```
short[] getSubnetMask()
```

**Description:**

Returns the subnet mask currently defined in the iLCD project settings.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

Package [ilcd](#)

Class [NetworkStatus](#)

**Public Method [isTcpIpEnabled](#)**

```
boolean isTcpIpEnabled()
```

**Description:**

Returns true if the TCP/IP settings are enabled in the iLCD project.

Also See the *TCP/IP Settings* category on the [Settings Tab](#) in the iLCD Manager XE.

---

**Package [ilcd](#)****Interface [OnTouchListener](#)**

The *OnTouchListener* interface is used for receiving touch events. Touch event listeners (classes implementing this interface) are required to implement the *onTouch()* method in order to react on touch events. The *onTouch()* method will be called automatically by the event management system after the listener object has been added to the [EventManager](#). Whenever a touch event occurs, the *onTouch()* method is invoked.

To receive touch events, touch field reporting has to be enabled. This is done using the method [Touch.setTouchFieldReportingEnabled\(boolean\)](#). Also a touch event listener object has to be added to the event system by calling [addListener\(OnTouchListenerObject\)](#). See [Touch](#) on how to create a touch field.

**Note**

- In order to use touch event handling the abstract class [Application](#) has to be extended.

**Public Methods**

- [onTouch](#)

**Package [ilcd](#)****Interface [OnTouchListener](#)****Public Method [onTouch](#)**

```
abstract void onTouch(TouchEvent event)
```

**Description:**

Parameter	Description
<a href="#">TouchEvent</a>	needs a TouchEvent object

The [onTouch\(\)](#) method is invoked when a touch event occurs.

**Example**

```
// within the constructor of a application class
// a new touch field is defined and created
...
Control.setCursorPosition(100, 50);
Touch.setTouchFieldReportingEnabled(true);

EventManager.getTouchEventDispatcher().addListener(OnTouchListener);
Touch.setTouchFieldWidth(100);
Touch.setTouchFieldHeight(50);
Touch.createDefineTouchField(1, 0);
```



```
...  
// new implementation of the onTouch() method  
// within the application class  
public void onTouch(TouchEvent event)  
{ ...
```

**See also:**

[Touch.setTouchFieldReportingEnabled\(boolean\)](#)  
[EventManager.getTouchEventDispatcher\(\).addListener\(\)](#)  
[Touch.setTouchFieldWidth\(int\)](#)  
[Touch.setTouchFieldHeight\(int\)](#)  
[Touch.createDefineTouchField\(int, int\)](#)

---

**Package [ilcd](#)****Class Position**

extends [Object](#)

The class *Position* is used in the iLCD *Control* class to obtain the current cursor position (refer to [Control.getCursorPosition\(\)](#)). There are two axes available on the screen. The *x* value describes the horizontal axis and the *y* value describes the vertical axis of the display.

**Public Constructors**

- [Position](#)

**Public Methods**

- [getX](#)
- [getY](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
  - [toString](#)
  - [wait](#)
  - [hashCode](#)
  - [notify](#)
  - [notifyAll](#)
- 

**Package [ilcd](#)****Class [Position](#)****Public Constructor Position**

```
Position ()
```

```
Position(int x, int y)
```

**Description:****constructor Position():**

The *Position()* constructor is used for the predefined *Control.setCursorPosition()* method. Please consider, there are no constructor or setter methods available to set up a position. To setup a position for the display cursor the *Control.setCursorPosition(int, int)* method is used.

**See also:**

[Control.setCursorPosition\(\)](#)  
[Control.setCursorPosition\(int, int\)](#)

**Package [ilcd](#)****Class [Position](#)****Public Method [getX](#)**

```
int getX()
```

**Description:**

Return s	Description
X	the stored x value within a new created <i>Position</i> object

A new created *Position* object can be used to save the current cursor position. A new cursor position is assigned to a *Position* object when the *Control.setCursorPosition()* method is called. Once a *Position* object is created, the x and y values at the objects creation time will be stored into the *Position* object for further use. The *Control.setCursorPosition()* method returns a *Position* object which contains the x, and y coordinate of the display.

**Example**

```
Position pos = Control.setCursorPosition();  
Draw.writeText("stored x position: " + pos.getX());
```

**See also:**

[Control.setCursorPosition\(\)](#)

## Package [ilcd](#)

### Class [Position](#)

#### Public Method [getY](#)

```
int getY()
```

#### Description:

Return s	Description
Y	the stored y value within a new created <i>Position</i> object

A new created *Position* object can be used to save the current cursor position. A new cursor position is assigned to a *Position* object when the *Control.setCursorPosition()* method is called. Once a *Position* object is created, the x and y values at the objects creation time will be stored into the *Position* object for further use. The *Control.setCursorPosition()* method returns a *Position* object which contains the x, and y coordinate of the display.

#### Example

```
Position pos = Control.setCursorPosition();
Draw.writeText("stored y position: " + pos.getY());
```

#### See also:

[Control.setCursorPosition\(\)](#)

## Package [ilcd](#)

### Class Power

extends [Object](#)

#### General Information About Power/Watchdog Related Inputs/Outputs

The iLCD controllers allow most port pins to be assigned as digital or analog inputs, outputs (pull down or push/pull) or keyboard columns via iLCD Manager XE. All methods referring to port pins below refer to the logical port name, not the physical port pin name.

The logical port names dealing with Power/Watchdog related Inputs/Outputs are as follows:

Port Name	Inputs/Outputs
ARES	watchdog reset output
APWR	PC power off output
ASPWR	disconnect PC's power switch output
APSWI	input from the PC's power switch

If a port function is not previously defined via iLCD Manager XE, the corresponding method is inactive. So, for example, when running the watchdog the ARES function must be assigned to a physical port pin via iLCD Manager XE to enable the pin to go high when the watchdog triggers.

Find following methods in this chapter as well as in the corresponding category when using the parameter completion feature of iLCD Manager XE:

#### Note

- It's not possible to instantiate this class. The methods in this class are static.

#### Public Methods

- [cancelShutdown](#)
- [feedWatchdog](#)
- [getPowerState](#)
- [hardShutdownLongPowerOff](#)
- [resetMotherboard](#)
- [setWatchdogInterval](#)
- [shutdownPowerOff](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- static final byte POWER\_DOWN\_SEQUENCE\_IS\_RUNNING
- static final byte POWER\_KEY\_STATE\_PRESSED

---

#### Package [ilcd](#)

#### Class [Power](#)

#### Public Method [cancelShutdown](#)

```
static void cancelShutdown ()
```

#### Description:

Cancels shutdown sequence.

#### Note

- This method deactivates the port pin with function APWR assigned to and does not wait for the PC's shutdown command anymore (when the power key was pressed before).
- The state of the LCD display and outputs remain unchanged.

- Pressing the power down key after this method has been received triggers the power down sequence again.
- If no pin is assigned to the APWR port function, no hardware activity will be carried out.

**See also:**[Power](#)[Power.shutdownPowerOff\(\)](#)

---

**Package [ilcd](#)****Class [Power](#)****Public Method [feedWatchdog](#)**

```
static void feedWatchdog ()
```

**Description:**

Resets the watchdog timer.

**Note**

- If the watchdog is enabled, feeding allows the application to further process another watchdog interval without pulling the reset line.

[Power](#)[Power.setWatchdogInterval\(int\)](#)

---

**Package [ilcd](#)****Class [Power](#)****Public Method [getPowerState](#)**

```
static int getPowerState ()
```

**Description:**

Returns the power state of the application.

**Note**

- Bit 0 of *state* indicates the power key.

Returns	Range	Description
state	0 ... 1	power state of the application

## Example

```
Power.getPowerState() == 0
```

If this expression evaluates to true, the power key is not pressed.

### See also:

[Power](#)

[Power.setWatchdogInterval\(int\)](#)

---

## Package [ilcd](#)

## Class [Power](#)

### Public Method [hardShutdownLongPowerOff](#)

```
static void hardShutdownLongPowerOff()
```

#### Description:

Activates the port pin with function APWR assigned to for a long period. The "Long power down" impulse is 5000ms by default, but can be modified on the "Settings" page of iLCD Manager XE.

#### Note

- If the hard-shutdown message is empty, no shutdown message box is displayed.
- If no pin is assigned to the APWR port function, no hardware activity will be carried out on shutdown.
- The display shows the hard-shutdown message box ("Hard Shutdown" by default, but can be changed on the "Settings" page of iLCD Manager XE).

## Example

```
Power.hardShutdownLongPowerOff();
```

Triggers a hard shutdown of the controlling application.

### See also:

[Power](#)

[Power.shutdownPowerOff\(\)](#)

---

Package [ilcd](#)  
Class [Power](#)

**Public Method [resetMotherboard](#)**

```
static void resetMotherboard()
```

**Description:**

Resets the motherboard of the controlling application.

**Note**

- This method disables the watchdog and activates the pin assigned to the ARES function for a certain defined via iLCD Manager XE. Outputs and LCD display remain in the previous state, which means that if the customer should see a message and/or the outputs should have a certain state while rebooting the PC, the corresponding sequences have to be sent by the PC before triggering the reset sequence.

**Example**

```
Power.resetMotherboard();
```

Resets the motherboard.

**See also:**

[Power](#)

Package [ilcd](#)  
Class [Power](#)

**Public Method [setWatchdogInterval](#)**

```
static void setWatchdogInterval(int interval)
```

**Description:**

Parameter	Range	Description
interval	0 ... 65535	interval in units of 10ms

Sets the watchdog interval according to *interval*.

**Note**

- The maximum interval is 655.35 seconds (10.92 minutes).
- Setting the interval causes the current watchdog interval to be retriggered.
- An *interval* of 0 disables the watchdog.

- By default, the watchdog is disabled. Hence it will be automatically disabled on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).
- If the PC fails to trigger the watchdog before its time is expired, the panel controller activates the port pin assigned to the ARES function for a certain time (the "Reset impulse" is 200ms by default, but can be modified on the "Settings" page of iLCD Manager XE), disables the watchdog and shows the watchdog reset message ("Watchdog Reset" by default, but can be modified on the "Settings" page of iLCD Manager XE) on the LCD display.
- If the watchdog-reset message is empty, no watchdog reset message is displayed. If no pin is assigned to the ARES port function, no hardware activity will be carried out on watchdog reset.

### Example

```
Power.setWatchdogInterval(1000);
```

Sets the watchdog interval to 10 seconds.

### See also:

[Power](#)

[Power.feedWatchdog\(\)](#)

[Power.shutdownPowerOff\(\)](#)

---

## Package [ilcd](#)

## Class [Power](#)

### Public Method [shutdownPowerOff](#)

```
static void shutdownPowerOff ()
```

### Description:

Activates the port pin with function APWR assigned to for a short time. The "Power down impulse" is 200ms by default, but can be modified on the "Settings" page of iLCD Manager XE.

### Note

- If the shutdown message is empty, no shutdown message box is displayed.
- If no pin is assigned to the APWR port function, no hardware activity will be carried out on shutdown.
- The display shows the shutdown message box ("Shutting Down.." by default, but can be changed on the "Settings" page of iLCD Manager XE).
- The shutdown sequence is aborted by the method [General.resetAll\(\)](#).

### Example

```
Power.shutdownPowerOff ();
```

Starts the shutdown sequence of the controlling application.



**See also:**[Power](#)[Power.hardShutdownLongPowerOff\(\)](#)**Package [ilcd](#)****Class [ProjectInfo](#)**extends [Object](#)

The class *ProjectInfo* is used in the iLCD [General](#) class to obtain information about the current project (refer to [General.getProjectInfo\(\)](#)).

**Public Methods**

- [getProjectDescription](#)
- [getProjectFileName](#)
- [getProjectModificationDate](#)
- [getProjectModificationTime](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [ilcd](#)****Class [ProjectInfo](#)****Public Method [getProjectDescription](#)**

```
String getProjectDescription()
```

**Description:**

Returns	Description
projectDescription	a string which contains the description of the project

**Example**

```
Draw.drawText(" get project description: "
+ General.getProjectInfo().getProjectDescription());
```

**See also:**[General.getProjectInfo\(\)](#)

---

Package [ilcd](#)Class [ProjectInfo](#)**Public Method `getProjectFileName`**

```
String getProjectFileName ()
```

**Description:**

Returns	Description
<code>projectFileName</code>	a string which contains the file name of the project

Returns the project filename (without extension).

**Example**

```
Draw.writeText(" get project file name: "  
              + General.getProjectInfo().getProjectFileName());
```

**See also:**[General.getProjectInfo\(\)](#)

---

Package [ilcd](#)Class [ProjectInfo](#)**Public Method `getProjectModificationDate`**

```
String getProjectModificationDate ()
```

**Description:**

Returns	Description
<code>projectModificationDate</code>	a string which contains the modification date of the project

Returns the project modification date ("MM.DD.YY").

**Example**

```
Draw.writeText(" get project modification date: "
```

```
+ General.getProjectInfo().getProjectModificationDate();
```

**See also:**

[General.getProjectInfo\(\)](#)

---

**Package [ilcd](#)****Class [ProjectInfo](#)****Public Method [getProjectModificationTime](#)**

```
String getProjectModificationTime()
```

**Description:**

Returns the project modification time ("HH:MM:SS").

**Description:**

Returns	Description
projectModificationTime	a string which contains the modification date of the project

Returns the project modification time ("HH:MM:SS").

**Example**

```
Draw.drawText(" get project modification time: "
+ General.getProjectInfo().getProjectModificationTime());
```

**See also:**

[General.getProjectInfo\(\)](#)

---

**Package [ilcd](#)****Class [ScreenParameters](#)**

extends [Object](#)

The class *ScreenParameters* is used in the iLCD [Memory](#) class to obtain the screen information. To obtain specific screen parameter please use the [Memory.getDrawScreenParameters\(\)](#) or [Memory.getViewScreenParameters\(\)](#) methods.

**Public Methods**

- [getScreenIndex](#)

- [getTextOrientation](#)
- [getViewportIndex](#)
- [getViewportOrientation](#)
- [getX](#)
- [getY](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

#### Package [ilcd](#)

### Class [ScreenParameters](#)

#### Public Method [getScreenIndex](#)

```
int getScreenIndex()
```

#### Description:

Returns	Description
screenIndex	a integer which contains the index of the screen

Returns the index of the currently active draw screen in the range of 'M', 0 ... number of screens - 1 (M = main screen).

#### Example

```
ScreenParameters scrPar = Memory.getViewScreenParameters();
if (scrPar.getScreenIndex() == 'M')
{
    Draw.writeText(" This is the main screen! ");
}
```

#### See also:

[Memory.getDrawScreenParameters\(\)](#)

[Memory.getViewScreenParameters\(\)](#)

**Package [ilcd](#)****Class [ScreenParameters](#)****Public Method [getTextOrientation](#)**

```
int getTextOrientation()
```

**Description:**

text/ graphic viewport or Orientation	Description
0	no rotation
1	the text/graphic items or the viewport are rotated by 90°
2	the text/graphic items or the viewport are rotated by 180°
3	the text/graphic items or the viewport are rotated by 270°

Returns the orientation of the text/graphic items relative to the [Viewport Related Methods](#) orientation in the range of 0 ... 3.

**Example**

```
ScreenParameters scrPar = Memory.getViewScreenParameters();
if (scrPar.getTextOrientation() == 0)
{
    Draw.writeText(" No rotation! ");
}
```

**See also:**

[Memory.getDrawScreenParameters\(\)](#)

[Memory.getViewScreenParameters\(\)](#)

**Package [ilcd](#)****Class [ScreenParameters](#)****Public Method [getViewportIndex](#)**

```
int getViewportIndex()
```

**Description:**

Returns	Range	Description
viewportIndex	0 ... 8	a integer which contains the index of the viewport

**Description:**

Returns the index of the screen's current selected viewport in the range of 0 ... 8.

**Example**

```
// define and select a new viewport
Control.setCursorPosition(310,210);
Control.defineViewport(3, 2, 250, 200);
Control.selectViewport(3);

// get screen parameter
ScreenParameters scrPar = Memory.getViewScreenParameters();
if (scrPar.getViewportIndex() == 3) // verify if viewport index is equal
to 3
{
    Draw.writeText(" 2 is the selected viewport index! ");
}
```

**See also:**

[Memory.getDrawScreenParameters\(\)](#)

[Memory.getViewScreenParameters\(\)](#)

**Package [ilcd](#)****Class [ScreenParameters](#)****Public Method [getViewportOrientation](#)**

```
int getViewportOrientation()
```

**Description:**

Returns	Range	Description
viewportOrientation	0 ... 3	a integer which contains the orientation of the viewport

**Description:**

Returns the orientation of [Viewport Related Methods](#) relative to the screen orientation in the range of 0 ... 3.

**Example**

```
// define and select a new viewport
Control.setCursorPosition(310,210);
Control.defineViewport(3, 2, 250, 200);
Control.selectViewport(3);
```

```
// get screen parameter
ScreenParameters scrPar = Memory.getViewScreenParameters();
if (scrPar.getViewportOrientation() == 2) // verify if viewport
orientation is equal to 2
{
    Draw.writeText(" 2 is the selected viewport orientation! ");
}
```

**See also:**

[Memory.getDrawScreenParameters\(\)](#)

[Memory.getViewScreenParameters\(\)](#)

**Package [ilcd](#)****Class [ScreenParameters](#)****Public Method [getX](#)**

```
int getX()
```

**Description:**

Returns	Range	Description
x	0 ... display width -1	a integer which contains the width value

Returns the horizontal coordinate of the cursor position related to the viewport in the range of 0 ... display width - 1.

**Example**

```
// define and select a new viewport
Control.setCursorPosition(310,210);
Control.defineViewport(3, 2, 250, 200);
Control.selectViewport(3);

// set a new cursor position within the current selected viewport
Control.setCursorPosition(10,10);

// get screen parameter
ScreenParameters scrPar = Memory.getViewScreenParameters();
if (scrPar.getX() == 10) // verify if the cursor position is setup
correctly
{
    Draw.writeText(" The current x cursor position is 10 within the
viewport 3! ");
}
```

**See also:**

[Memory.getDrawScreenParameters\(\)](#)  
[Memory.getViewScreenParameters\(\)](#)

---

**Package** [ilcd](#)**Class** [ScreenParameters](#)**Public Method** [getY](#)

```
int getY()
```

**Description:**

Return s	Range	Description
y	0 ... display height - 1	a integer which contains the height value

Returns the vertical coordinate of the cursor position related to the viewport in the range of 0 ... display height - 1.

**Example**

```
// define and select a new viewport
Control.setCursorPosition(310,210);
Control.defineViewport(3, 2, 250, 200);
Control.selectViewport(3);

// set a new cursor position within the current selected viewport
Control.setCursorPosition(10,10);

// get screen parameter
ScreenParameters scrPar = Memory.getViewScreenParameters();
if (scrPar.getY() == 10) // verify if the cursor position is setup
correctly
{
    Draw.writeText(" The current y cursor position is 10 within the
viewport 3! ");
}
```

**See also:**

[Memory.getDrawScreenParameters\(\)](#)  
[Memory.getViewScreenParameters\(\)](#)

---



## Package [ilcd](#)

### Class **Size**

extends [Object](#)

The class *Size* is used in several iLCD classes to obtain the height and width of the display current in use (refer to [Control.getDisplaySize\(\)](#)), the extent of a specific text (refer to [Control.getTextExtent\(\)](#), [Control.getTextMessageExtent\(\)](#), and [Control.getUnicodeTextExtent\(\)](#)), or to get size information about a graphic (refer to [GraphicInfo.getSize\(\)](#)). The class *Size* might as well be used for custom purposes.

#### Public Constructors

- [Size](#)

#### Public Methods

- [getHeight](#)
- [getWidth](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [ilcd](#)

### Class **Size**

#### Public Constructor Size

```
Size()
```

```
Size(int width, int height)
```

#### Description:

##### constructor **Size()**:

The *Size()* constructor is used for predefined iLCD methods only. Don't use this constructor for general purpose. If this constructor is called to create a *Size* object for general purpose the stored values are undefined till the [setHeight\(int\)](#) and [setWidth\(int\)](#) methods are called. The *getHeight()* and *getWidth()* methods are used to obtain the values.

##### constructor **Size(int width, int height)**:

Parameter	Range	Description
-----------	-------	-------------

height	-2.147.483.648 ... 2.147.483.647	the height value of a size object
width	-2.147.483.648 ... 2.147.483.647	the width value of a size object

A new created *Size* object can be used to store a user defined size with a height and width. The [getHeight\(int\)](#) and [setWidth\(int\)](#) methods can be used to set new values during the lifetime of the *Size* object. The [getHeight\(\)](#) and [getWidth\(\)](#) methods are used to obtain the values which are assigned to the *Size* object.

### Example

```
Size size = new Size(100, 99);
Draw.writeText("size height: " + size.getHeight() + "\r"
              + "size width: " + size.getWidth());
```

### See also:

[Control.getDisplaySize\(\)](#)  
[Control.getTextExtent\(String\)](#)  
[Control.getTextMessageExtent\(String\)](#)  
[Control.getUnicodeTextExtent\(String\)](#)  
[GraphicInfo.getSize\(\)](#)

## Package [ilcd](#) Class [Size](#)

### Public Method [getHeight](#)

```
int getHeight\(\)
```

#### Description:

Return s	Range	Description
height	-2.147.483.648...2.147.483.647	the height value of a Size object

Returns the height value of a *Size* object.

### Example

```
Draw.writeText("" + Control.getDisplaySize().getHeight());
```

### See also:

[Control.getDisplaySize\(\)](#)  
[Control.getTextExtent\(String\)](#)  
[Control.getTextMessageExtent\(String\)](#)  
[Control.getUnicodeTextExtent\(String\)](#)  
[GraphicInfo.getSize\(\)](#)

---

**Package [ilcd](#)****Class [Size](#)****Public Method [getWidth](#)**

```
int getWidth()
```

**Description:**

Returns	Range	Description
width	-2.147.483.648...2.147.483.647	the width value of a Size object

Returns the width value of a *Size* object.

**Example**

```
Draw.writeText("" + Control.getDisplaySize().getWidth());
```

**See also:**

[Control.getDisplaySize\(\)](#)  
[Control.getTextExtent\(String\)](#)  
[Control.getTextMessageExtent\(String\)](#)  
[Control.getUnicodeTextExtent\(String\)](#)  
[GraphicInfo.getSize\(\)](#)

---

**Package [ilcd](#)****Class [Size](#)****Public Method [setHeight](#)**

```
void setHeight(int height)
```

**Description:**

Parameter	Range	Description
height	-2.147.483.648...2.147.483.647	the height value of a Size object

The [setHeight\(int\)](#) method can only be used if a *Size* object is created before.

**Example**

```
Size size = new Size(0, 0);  
size.setHeight(100);
```

**See also:**[Size](#)**Package [ilcd](#)**  
**Class [Size](#)****Public Method [setWidth](#)**

```
void setWidth(int width)
```

**Description:**

Parameter	Range	Description
width	-2.147.483.648...2.147.483.647	the width value of a <i>Size</i> object

The [setWidth\(int\)](#) method can only be used if a *Size* object is created before.

**Example**

```
Size size = new Size(0, 0);
size.setWidth(100);
```

**See also:**[Size](#)**Package [ilcd](#)****Class [Touch](#)**

extends [Object](#)

The class *Touch* provides methods for creating and handling touch fields. You may define up to 64 rectangular (possibly overlapping) touch fields. The maximum field size is the pixel screen size used by the current iLCD.

When using multiple screens, note that touch fields are created at the screen index of the currently active draw screen (see [setDrawScreen](#)). On the other hand, only touch fields on the activated view screen get evaluated and reported.

**PCAP Touch Screen**

Projected capacitive (PCAP) touch screens support up to 5 simultaneously evaluated touch points. The sensitivity is determined by values stored in the EEPROM (refer to Class [EEPROM](#)).

## Notes

- It's not possible to instantiate this class. The methods in this class are static.
- In order to use touch event handling the abstract class [Application](#) has to be extended and the [OnTouchListener](#) interface has to be implemented.

## Public Methods

- [calibrateTouchScreen](#)
- [createDefineTouchField](#)
- [drawTouchFieldTextMessage](#)
- [getCurrentTouchFieldIndex](#)
- [getNumberOfTouchFingers](#)
- [removeTouchField](#)
- [retrieveLastTouchScreenEvent](#)
- [setCurrentTouchFieldIndex](#)
- [setCursorToTouchField](#)
- [setNumberOfTouchFingers](#)
- [setThresholdForMovementReporting](#)
- [setTouchFieldHeight](#)
- [setTouchFieldReportingEnabled](#)
- [setTouchFieldTextMessage](#)
- [setTouchFieldWidth](#)
- [getPcapControllerInformation](#)
- [verifyTouchScreenCalibration](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

## Package [ilcd](#)

## Class [Touch](#)

### Public Method [calibrateTouchScreen](#)

```
static void calibrateTouchScreen()
```

#### Throws:

- [ILCDException](#)

#### Description:

Starts the procedure to calibrate the touch screen.

**Note**

- All iLCD panels with a touch panel are calibrated after production, but you may repeat the process anytime.
- When issuing this method, the iLCD panel shows instructions for calibration on the screen. After pressing two positions, the calibration is done and the calibration values are stored in the iLCD controller automatically.
- Sending [General.resetAll\(\)](#) stops the calibration process, the previous existing values are used then.

**Example**

```
Touch.calibrateTouchScreen();
```

Start the calibration procedure.

**See also:**

[Touch.verifyTouchScreenCalibration\(\)](#)

**Package [ilcd](#)****Class [Touch](#)****Public Method [createDefineTouchField](#)**

```
static void createDefineTouchField(int fieldIdx, int key)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
<code>fieldIdx</code>	0 ... 63 (-1 = next free touch field index)	index of the touch field
<code>key</code>	0 ... 255	value reported in case of touch event

Creates or redefines the touch field with index `fieldIdx` with the current values and properties previously set (e.g. [Touch.setTouchFieldWidth\(int\)](#), etc.). The value `key` can be set in the range 0 till 255.

**Note**

- The upper left position of the touch field is taken from the current cursor position.
- `fieldIdx` may have the hex value 0xFF (decimal byte -1), in this case the controller will choose the next free (unassigned) touch field index.
- Using this method also sets the 'current touch field index' (see [Touch.setCurrentTouchFieldIndex\(int\)](#)), which can be used by [Touch.drawTouchFieldTextMessage\(int\)](#).

- If a value greater than 255, or smaller than 0 is assigned to the *key* the behavior is undefined.

### Example

```
Touch.createDefineTouchField(-1, 0);
```

This will create a touch field with the next free index (*fieldIdx*). If the new created touch field is pressed the index (*fieldIdx*) of the touch field, the event type, the event coordinates, and the point ID (optional) will be set.

### See also:

[Touch](#)  
[TouchEvent](#)  
[Touch.removeTouchField\(int\)](#)  
[enableDisableTouchFieldReporting\(boolean\)](#)  
[Touch.setCurrentTouchFieldIndex\(int\)](#)  
[Touch.drawTouchFieldTextMessage\(int\)](#)

## Package **ilcd**

## Class **[Touch](#)**

### Public Method **drawTouchFieldTextMessage**

```
static void drawTouchFieldTextMessage(int fieldIdx)
```

#### Throws:

- [ILCDEException](#)

#### Description:

Parameter	Range	Description
<code>fieldIdx</code>	0 ... 63	index of the touch field (-1 = current index)

#### Note

- The message offset, prefix and suffix are taken into account (see [Extra.setMessageOffset\(int\)](#), [Extra.setMessageNamePrefix\(String\)](#) and [Extra.setMessageNameSuffix\(String\)](#)).
- If *fieldIdx* is set to 0xFF (decimal byte -1) the 'current touch field index' is used (refer to [Touch](#)).
- If no text message is assigned to the field and the key assigned to the field is other than 0x0, the key is printed instead. This behavior is useful for single character touch fields like "keyboard" fields, as no text message has to be assigned for every touch field.
- Note that when a make or break macro is executed, the cursor position is automatically set to the upper/left corner of the corresponding touch field. So when using this method inside a touch macro, text output can be placed or aligned (see [Control.setTextAlignment\(int, int, int\)](#)) relative to the touch field position.

## Example

```
Touch.drawTouchFieldTextMessage(5);
```

Draws the text message assigned to the touch field with index 5 in its top left corner, given the settings for the message offset aren't changed.

### See also:

[Touch](#)

[Touch.setCurrentTouchFieldIndex\(int\)](#)

[Touch.setTouchFieldTextMessage\(int\)](#)

[Extra.setMessageOffset\(int\)](#)

[Extra.setMessageNamePrefix\(String\)](#)

[Extra.setMessageNameSuffix\(String\)](#)

Package [ilcd](#)

Class [Touch](#)

### Public Method [getCurrentTouchFieldIndex](#)

```
static int getCurrentTouchFieldIndex()
```

#### Description:

Returns the *TouchFieldIndex* of the last touch field created by calling the method [createDefineTouchField](#).

Package [ilcd](#)

Class [Touch](#)

### Public Method [getNumberOfTouchFingers](#)

```
static int getNumberOfTouchFingers()
```

#### Throws:

- [ILCDEException](#)

#### Description:

Returns	Range	Description
ingers	1 ... 5	number of simulataneously evaluated touch points

Returns the number of evaluated touch points to *fingers*.



**Note**

- While *fingers* is higher than 1, an automatically assigned point ID is appended to all touch field reports.
- On iLCD panels with resistive touch screen, *fingers* is always 1.
- On iLCD panels with capacitive touch screen, the maximum value for *fingers* is 5.
- The default value for *fingers* is 1 in any case. It will be automatically set to default on startup.

**Example**

```
Draw.writeText("getNumberOfTouchFingers(): " +
Touch.getNumberOfTouchFingers() + " ");
```

Returns the number of touch points evaluated simultaneously.

**See also:**

[Touch](#)

[Touch.setNumberOfTouchFingers\(\)](#)

[Touch.createDefineTouchField\(int, int\)](#)

Package [ilcd](#)

**Class [Touch](#)****Public Method [removeTouchField](#)**

```
static void removeTouchField(int fieldIdx)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
fieldIdx	0 ... 63	index of the touch field (-1 = all, -2 = all in viewport)

Removes the touch field specified by *fieldIdx*.

**Note**

- If *fieldIdx* is set to 0xFF (decimal byte -1), all touch fields on all screens and viewports are removed.
- In firmware versions 4.17 and higher, a *fieldIdx* of 0xFE (decimal byte -2) removes all touch fields in the active viewport. When the active viewport is 0, all touch fields that are not defined in user-defined viewports are removed. This method can be used to remove all touch fields on the current draw screen (refer to class [Memory](#)).

- The methods [General.resetAll\(\)](#) and [General.rebootPanelController\(\)](#) remove all touch fields as well.

### Example

```
Touch.removeTouchField(12);
```

Removes the touch field with index 12.

### See also:

[Touch](#)

[Touch.createDefineTouchField\(int, int\)](#)

## Package [ilcd](#)

### Class [Touch](#)

#### **Public Method retrieveLastTouchEvent**

```
static TouchEvent retrieveLastTouchEvent ()
```

#### Throws:

- [ILCDEException](#)

#### Description:

Returns	Range	Description
event	K, k, M	event type (refer to table below)
pointId	0 ... 4	(optional) ID of the touch point (finger)
fieldIdx	0 ... 63	index of the touch field
evCoordX	0 ... display width	horizontal position of the event
evCoordY	0 ... display height	vertical position of the event

- Note that for historical reasons the *FieldKey* entry of the returned *TouchEvent* is always 0.

### Example

```
Touch.retrieveLastTouchEvent ();
```

The last touch event was a make event (touch field was pressed) reported from the touch field with index 1 at coordinates x = 125 pixels and y = 150 pixels. Here, the *point\_id* is missing since only one touch point is evaluated.

**See also:**[Touch](#)[Touch.createDefineTouchField\(int, int\)](#)[Touch.setNumberOfTouchFingers\(int\)](#)Package [ilcd](#)Class [Touch](#)**Public Method setCurrentTouchFieldIndex**

```
static void setCurrentTouchFieldIndex(int fieldIdx)
```

**Throws:**

- [ILCDException](#)

**Description:**

Parameter	Range	Description
fieldIdx	0 ... 63	current touch field index

Sets the iLCD controller's 'current touch field index' to *fieldIdx*.

**Note**

- The iLCD controller's 'current touch field index' is an internal variable which is automatically updated when a touch field is created, pressed or released.
- [Touch.drawTouchFieldTextMessage\(int\)](#).

**Example**

```
Touch.setCurrentTouchFieldIndex(23);
```

Sets the 'current touch field index' of the iLCD controller to 23.

**See also:**[Touch](#)[Touch.createDefineTouchField\(int, int\)](#)[Touch.enableDisableTouchFieldReporting\(boolean\)](#)[Touch.drawTouchFieldTextMessage\(int\)](#)

**Package [ilcd](#)**  
**Class [Touch](#)****Public Method [setCursorToTouchField](#)**

```
static void setCursorToTouchField(int fieldIdx)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
fieldIdx	0 ... 63	index of the touch field (-1 = current index)

Sets the cursor to the upper/left corner of the touch field specified by *fieldIdx*.

**Note**

- If *fieldIdx* is set to 0xFF (decimal byte -1) the 'current touch field index' is used (refer to class [Touch](#)).

**Example**

```
Touch.setCursorToTouchField(5);
```

Sets the cursor position to the upper/left corner of the touch field with index 5.

**See also:**

[Touch](#)  
[Touch.createDefineTouchField\(int, int\)](#)

---

**Package [ilcd](#)**  
**Class [Touch](#)****Public Method [setNumberOfTouchFingers](#)**

```
static void setNumberOfTouchFingers(int fingers)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
-----------	-------	-------------

<code>fingers</code>	<code>1 ... 5</code>	number of simulataneously evaluated touch points
----------------------	----------------------	--

Sets the number of evaluated touch points to *fingers*.

### Note

- While *fingers* is higher than 1, an automatically assigned point ID is appended to all touch field reports.
- On iLCD panels with resistive touch screen, *fingers* is always 1.
- On iLCD panels with capacitive touch screen, the maximum value for *fingers* is 5.
- The default value for *fingers* is 1 in any case. It will be automatically set to default on startup.

### Example

```
Touch.setNumberOfTouchFingers(2);
```

Sets the number of touch points evaluated simultaneously to 2 (works on iLCD panels with capacitive touch screen only).

### See also:

[Touch](#)  
[Touch.getNumberOfTouchFingers\(\)](#)  
[Touch.createDefineTouchField\(int, int\)](#)

## Package [ilcd](#)

## Class [Touch](#)

### Public Method [setThresholdForMovementReporting](#)

```
static void setThresholdForMovementReporting(int threshold)
```

### Throws:

- [ILCDEException](#)

### Description:

Parameter	Range	Description
<code>threshold</code>	<code>0 ... 255</code>	threshold for touch move events

When reporting of touch movements is enabled (refer to [Touch.setTouchFieldReportingEnabled\(boolean\)](#)), reports are sent every time a touch point moves by *threshold* pixels.

**Note**

- Calling this method does not change anything for touch fields already defined.
- The default value for *threshold* is 8. It will be automatically set to default on startup.

**Example**

```
Touch.setThresholdForMovementReporting(20);
```

Movement reports are sent every time the touch events moves by 20 pixels in any direction.

**See also:**

[Touch](#)

---

**Package [ilcd](#)****Class [Touch](#)****Public Method [setTouchFieldHeight](#)**

```
static void setTouchFieldHeight(int height)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
height	0 ... display height	height of the touch field (0 = full height)

Sets up the height of any subsequently defined touch field (see [Touch.createDefineTouchField\(int, int\)](#)) according to *height*.

**Note**

- Sending this method does not change anything for touch fields already defined.
- If *height* is 0, the full height of the current viewport is used.
- After startup and after calling the [General.resetAll\(\)](#) method *height* is set to 20.

**Example**

```
Touch.setTouchFieldHeight(25);
```

Sets the height for the next touch field to be created to 25 pixels.

**See also:**[Touch](#)[Touch.setTouchFieldWidth\(int\)](#)[Touch.createDefineTouchField\(int, int\)](#)Package [ilcd](#)Class [Touch](#)**Public Method setTouchFieldReportingEnabled**

```
static void setTouchFieldReportingEnabled(boolean enable)
```

**Throws:**

- [ILCDException](#)

**Description:**

Parameter	Range	Description
enable	false ... true	determines whether touch events are raised (true).

**Note**

- If set to false the events will still be raised.
- The default value for *enable* is false. It will be automatically set to default on startup and by the methods [General.resetAll\(\)](#) or [General.rebootPanelController\(\)](#).

**Example**

```
Touch.setTouchFieldReportingEnabled(true);
```

Enables touch field event handling by invoking the method [EventManager.start\(\)](#). If a touch field is pressed afterwards, the index of the touch field, event type, event coordinates and the (optional) ID of the touch point (finger) will be set in [TouchEvent](#).

**See also:**[Touch](#)[TouchEvent](#)[Touch.createDefineTouchField\(int, int\)](#)[Touch.setCurrentTouchFieldIndex\(int\)](#)

## Package [ilcd](#)

### Class [Touch](#)

#### Public Method `setTouchFieldTextMessage`

```
static void setTouchFieldTextMessage(int messageIndex)
```

##### Throws:

- [ILCDEException](#)

```
static void setTouchFieldTextMessage(String messageName)
```

##### Throws:

- [ILCDEException](#)

##### Description:

##### by index:

Parameter	Range	Description
<code>messageIndex</code>	0 ... max. text message index	index of the text message

##### by name:

Parameter	Range	Description
<code>messageName</code>	ASCII chars (0x01 .. 0xFF)	name of the message

Assigns the text message defined by `messageIndex` or `messageName` to any subsequently defined touch field (see [Touch.createDefineTouchField\(int, int\)](#)).

##### Note

- Calling this method does not change anything for touch fields already defined.
- Setting `messageIndex` to 0xFFFF causes no text to be assigned to the subsequently defined touch field.
- After startup and after calling the [General.resetAll\(\)](#) method `messageIndex` is set to -1.
- Text messages assigned to touch fields are used by the [Touch.drawTouchFieldTextMessage\(int\)](#) method.
- The text message can even contain any valid ANSI sequence so allowing various attributes/fonts to be changed easily.
- At this time, the attributes message offset, prefix and suffix are ignored and the existence of the message index or name is not verified (see [Extra.setMessageOffset\(int\)](#), [Extra.setMessageNamePrefix\(String\)](#) and [Extra.setMessageNameSuffix\(String\)](#)). Offset resp. pre-/suffix is taken into account when the message is actually drawn (see [Touch.drawTouchFieldTextMessage\(int\)](#)).

##### Example

```
Touch.setTouchFieldTextMessage(3);
```



Assigns the text message with index 3 to any subsequently defined touch field.

**See also:**

[Touch](#)

[Touch.drawTouchFieldTextMessage\(int\)](#)

[Touch.createDefineTouchField\(int, int\)](#)

[Touch.drawTouchFieldTextMessage\(int\)](#)

[Touch.setCurrentTouchFieldIndex\(int\)](#)

[Extra.setMessageOffset\(int\)](#)

[Extra.setMessageNamePrefix\(String\)](#)

[Extra.setMessageNameSuffix\(String\)](#)

---

**Package [ilcd](#)**

**Class [Touch](#)**

**Public Method [setTouchFieldWidth](#)**

```
static void setTouchFieldWidth(int width)
```

**Throws:**

- [ILCDEException](#)

**Description:**

Parameter	Range	Description
width	0 ... display width	width of the touch field (0 = full width)

Sets up the width of any subsequently defined touch field (see [Touch.createDefineTouchField\(int, int\)](#)) according to *width*.

**Note**

- Calling this method does not change anything for touch fields already defined.
- If *width* is 0, the full width of the current viewport is used.
- After startup and after calling the [General.resetAll\(\)](#) method *width* is set to 30.

**Example**

```
Touch.setTouchFieldWidth(120);
```

Sets the width for the next touch field to be created to 120 pixels.

**See also:**

[Touch](#)

[Touch.setTouchFieldHeight\(int\)](#)

[Touch.createDefineTouchField\(int, int\)](#)

---

Package [ilcd](#)

Class [Touch](#)

### **Public Method getPcapControllerInformation**

```
String getPcapControllerInformation()
```

**Description:**

Returns	Description
PcapControllerInformation	PCAP information

The `getPcapControllerInformation()` method returns information regarding the PCAP of the module as a string.

Example string which is returned by this method: *Manuf.: FocalTech, ChipID: 85, FW: 3, FW-lib.: 48.3*

**Example**

```
Draw.writeText(Touch.getPcapControllerInfo());
```

**See also:**

[General.getDeviceInfo\(\)](#)

---

Package [ilcd](#)

Class [Touch](#)

### **Public Method verifyTouchScreenCalibration**

```
static void verifyTouchScreenCalibration()
```

**Throws:**

- [ILCDEException](#)

**Description:**

The calibration of the touch screen can be tested after calling this method. The position of every touch event is marked with a small cross on the screen.

**Note**

- The validation can be stopped via the [General.resetAll\(\)](#) method.

## Example

```
Touch.verifyTouchScreenCalibration();
```

When a touch event occurs, the screen turns white and every touch event shows a small cross.

### See also:

[Touch.calibrateTouchScreen\(\)](#)  
[Touch.createDefineTouchField\(int, int\)](#)

---

## Package [ilcd](#)

### Class [TouchEvent](#)

extends [Object](#)

The class [TouchEvent](#) is used to receive informations about touch field events. A [TouchEvent](#) object is passed to the method [onTouch\(\)](#) of the [OnTouchListener](#) Interface. There are basically three types of touch events: pressed, released and moved. When an event occurs, the  $x$  and  $y$  coordinates of the event can be read from the iLCD display. It's possible to create 64 rectangular touch fields with [Touch.createDefineTouchField\(int, int\)](#). The first parameter defines the touch field index which is stored in the [TouchEvent](#) object.

### Note

- In order to use touch event handling the abstract class [Application](#) has to be extended and the [OnTouchListener](#) interface has to be implemented.

### Multi Touch

Up to 5 fingers can be distinguished when simultaneously touching the iLCD. A point ID is given to each finger.

### Public Methods

- [getEventType](#)
- [getFieldIdx](#)
- [getFieldKey](#)
- [getPointID](#)
- [getX](#)
- [getY](#)
- [isTouchMoved](#)
- [isTouchPressed](#)
- [isTouchReleased](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)

- [notify](#)
- [notifyAll](#)

---

Package [ilcd](#)

## Class [TouchEvent](#)

### Public Method `getEventType`

```
char getEventType()
```

#### Description:

Returns	Range	Description
<code>eventType</code>	'k', 'K', 'M'	current state of a created and defined touch field

The [getEventType\(\)](#) method returns the current state of a created and defined touch field. A pressed, released and moved state can be distinguished.

#### Example

```
public void onTouch(TouchEvent event)
{
    Draw.writeText(" getEventType: " + event.getEventType());
    ...
}
```

#### See also:

[Touch.isTouchPressed\(\)](#)  
[Touch.isTouchReleased\(\)](#)

---

Package [ilcd](#)

## Class [TouchEvent](#)

### Public Method `getFieldIdx`

```
int getFieldIdx()
```

#### Description:

Returns	Range	Description
<code>fieldIdx</code>	0 ... 63	value which is set when a touch field is created and defined

The *fieldIdx* variable is used to determine which created and defined touch field is pressed, released, or moved. The value is set during the [Touch.createDefineTouchField\(int, int\)](#) method is called which create and define a new touch field within the screen.

### Example

```
public void onTouch(TouchEvent event)
{ ...
  if(event.isTouchReleased())
  {
    Draw.writeText(" getFieldIdx: " + event.getFieldIdx());
    ...
  }
}
```

### See also:

[Touch.isTouchPressed\(\)](#)  
[Touch.isTouchReleased\(\)](#)

## Package [ilcd](#)

## Class [TouchEvent](#)

### Public Method [getFieldKey](#)

```
int getFieldKey()
```

#### Description:

Returns	Range	Description
fieldKey	0 ... 255	value which is set when a touch field is created and defined

Up to 256 different field key values can be assigned during the touch field creation (see [Touch.createDefineTouchField\(int, int\)](#)). The field key values are user defined values which can be received by the [getFieldKey\(\)](#) method once a touch field is created.

#### Note

- If a value greater than 255, or smaller than 0 is assigned to the field key during creation (see [Touch.createDefineTouchField\(int, int\)](#)) the behavior is undefined.

### Example

```
public void onTouch(TouchEvent event)
{
  if(event.isTouchReleased())
  {
    Draw.writeText(" getFieldKey: " + event.getFieldKey());
  }
}
```

---

...

**See also:**[Touch.createDefineTouchField\(int, int\)](#)[Touch.isTouchPressed\(\)](#)[Touch.isTouchReleased\(\)](#)

---

**Package [ilcd](#)****Class [TouchEvent](#)****Public Method [getPointID](#)**

```
int getPointID()
```

**Description:**

Returns	Range	Description
pointID	0 ... 4	value of the current point

Up to 5 fingers can be distinguished when simultaneously touching the iLCD. A point ID is given to each finger. The value of the point id depends on the order of the touch events (fingers touching the screen). The value 0 will be assigned to the first finger on the touch screen. If the first finger doesn't release the touch screen and a second finger will touch the screen the value 1 will be assigned to the second touch event.

**Example**

```
public void onTouch(TouchEvent event)
{ ...
    if(event.isTouchReleased())
    {
        Draw.writeText(" getPointID: " + event.getPointID());
    }
    ...
}
```

**See also:**[Touch.isTouchPressed\(\)](#)[Touch.isTouchReleased\(\)](#)

---

**Package [ilcd](#)****Class [TouchEvent](#)****Public Method [getX](#)**

```
int getX()
```

**Description:**

Returns	Range	Description
x	created and defined touch field width: - display width ... display width - 1	x value depends on the cursor position of the created and defined touch field

With the [getX\(\)](#) method the *x* position of the current event is returned. The value of the *x* coordinate depends not on the screen but on the defined and created touch field. So if a defined and created touch field starts with a current cursor position of 100/100 (*y/x* coordinate). The [getX\(\)](#) method will return 0 even if the absolute *x* coordinate is 100. The [getX\(\)](#) method returns the *x* coordinate of the current touch field, and not the absolute *x* coordinate of the entire screen.

**Example**

```
public void onTouch(TouchEvent event)
{ ...
  if(event.isTouchReleased())
  {
    Draw.writeText(" getX: " + event.getX());
    ...
  }
}
```

**See also:**

[TouchEvent.isTouchPressed\(\)](#)  
[TouchEvent.isTouchReleased\(\)](#)

**Package [ilcd](#)****Class [TouchEvent](#)****Public Method [getY](#)**

```
int getY()
```

**Description:**

Returns	Range	Description
y	created and defined touch field height: - display height ... display height - 1	y value depends on the cursor position of the created and defined touch field

	height - 1	
--	------------	--

With the [getY\(\)](#) method the  $y$  position of the current event is returned. The value of the  $y$  coordinate depends not on the screen but on the defined and created touch field. So if a defined and created touch field starts with a current cursor position of 100/100 ( $y/x$  coordinate). The [getY\(\)](#) method will return 0 even if the absolute  $y$  coordinate is 100. The [getY\(\)](#) method returns the  $y$  coordinate of the current touch field, and not the absolute  $y$  coordinate of the entire screen.

### Example

```
public void onTouch(TouchEvent event)
{ ...
  if(event.isTouchReleased())
  {
    Draw.writeText(" getY: " + event.getY());
    ...
  }
}
```

### See also:

[Touch.isTouchPressed\(\)](#)  
[Touch.isTouchReleased\(\)](#)

## Package [ilcd](#)

## Class [TouchEvent](#)

### Public Method isTouchMoved

```
boolean isTouchMoved()
```

### Description:

Returns	Range	Description
moved	true ... false	becomes true when the touch field coordinate is changed

The current state of the defined and created touch field is evaluated to true, if the pressed touch field coordinate is moved.

### Example

```
public void onTouch(TouchEvent event)
{ ...
  if(event.isTouchMoved())
  { ...
  }
}
```



**See also:**

[Touch.isTouchPressed\(\)](#)  
[Touch.isTouchReleased\(\)](#)

---

**Package [ilcd](#)****Class [TouchEvent](#)****Public Method [isTouchPressed](#)**

```
boolean isTouchPressed()
```

**Description:**

Returns	Range	Description
pressed	true ... false	becomes true when the touch field is pressed

The current state of the defined and created touch field is evaluated to true, if the pressed touch field is already pressed.

**Example**

```
public void onTouch(TouchEvent event)
{ ...
    if(event.isTouchPressed())
    { ...
```

**See also:**

[Touch.isTouchReleased\(\)](#)  
[Touch.isTouchMoved\(\)](#)

---

**Package [ilcd](#)****Class [TouchEvent](#)****Public Method [isTouchReleased](#)**

```
boolean isTouchReleased()
```

**Description:**

Returns	Range	Description
released	true ... false	becomes true when the touch field is released

The current state of the defined and created touch field is evaluated to true, if the pressed touch field is released.

### Example

```
public void onTouch(TouchEvent event)
{ ...
    if(event.isTouchReleased())
    { ...
```

### See also:

[Touch.isTouchPressed\(\)](#)

[Touch.isTouchMoved\(\)](#)

---

## Package [ilcd](#)

### Class **TouchEventDispatcher**

extends [AsyncEvent](#) → [Object](#)

The iLCD Java API implements the commonly used observer pattern. Listeners (observers) subscribe to certain events distributed by an dispatcher object that notifies the listeners when an event occurs. The [TouchEventDispatcher](#) class provides this kind of dispatcher functionality for distributing touch events. The actual TouchEventDispatcher object is part of the class [EventManagement](#).

### Notes

- In order to use touch event handling the abstract class [Application](#) has to be extended and the [OnTouchListener](#) interface has to be implemented.
- [EventManagement](#) is automatically started when invoking the method [setTouchFieldReportingEnabled](#).

### Public Constructors

- [TouchEventDispatcher](#)

### Public Methods

- [addListener](#)
- [getListenerCount](#)
- [processEvents](#)
- [removeAllListener](#)
- [removeListener](#)

### Methods inherited from [javax.events.AsyncEvent](#)

- [addHandler](#)
- [getHandler](#)
- [handledBy](#)
- [removeHandler](#)
- [setHandler](#)

- [fire](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

---

### Package [ilcd](#)

## Class [TouchEventDispatcher](#)

### Public Method addListener

```
void addListener(OnTouchListener listener)
```

#### Description:

Parameter	Description
<a href="#">OnTouchListener</a>	OnTouchListener object

The [OnTouchListener](#) interface must be implemented by classes which want to be notified on touch events. The [addListener\(\)](#) method adds an [OnTouchListener](#) object to the touch event dispatcher.

#### Example

```
public class App implements OnTouchListener
...
    // within the class constructor
    EventManagement.getTouchEventDispatcher().addListener(this);
...
```

#### See also:

[Touch](#)  
[TouchEvent](#)  
[OnTouchListener](#)

---

### Package [ilcd](#)

## Class [TouchEventDispatcher](#)

### Public Method getListenerCount

```
int getListenerCount()
```

**Description:**

Returns	Range	Description
int	0 ... 63	the count of the registered listeners

This method returns the total number touch event listeners.

**Example**

```
Draw.writeText(" registered listeners: "
              +
              EventManagement.getTouchEventDispatcher().getListenerCount());
```

**See also:**

[Touch](#)  
[TouchEvent](#)  
[OnTouchListener](#)

---

**Package [ilcd](#)****Class [TouchEventDispatcher](#)****Public Method processEvents**

```
void processEvents()
```

**Description:**

If an touch event occurs the [processEvents\(\)](#) method will notify all touch event listeners by calling their implementation of the [onTouch\(\)](#) method.

**Note**

- This method will be called from the abstract class [Application](#) in it's [run\(\)](#) method. It shouldn't normally be called by the user except an Application class is not present.

**Example**

```
EventManagement.getTouchEventDispatcher().processEvent();
```

**See also:**

[Application](#)

---

Package [ilcd](#)

## Class [TouchEventDispatcher](#)

### Public Method removeAllListener

```
void removeAllListener()
```

#### Description:

This method removes all touch event listeners.

#### Example

```
EventManager.getTouchEventDispatcher().removeAllListener();
```

#### See also:

[Application](#)

---

Package [ilcd](#)

## Class [TouchEventDispatcher](#)

### Public Method removeListener

```
void removeListener(OnTouchListener listener)
```

#### Description:

Parameter	Description
<a href="#">OnTouchListener</a>	needs a OnTouchListener object

This method removes a specific touch event listener.

#### Example

```
public class Button implements OnTouchListener
{
    ...
    EventManagement.getTouchEventDispatcher().removeListener(this);
    ...
}
```

#### See also:

[Application](#)

---

## Package java.io

Provides classes and interfaces for system input and output through data streams, serialization and the file system.

Note that SD Card handling is done within the class [File](#).

## Interfaces

- [DataInput](#)
- [DataOutput](#)
- [Externalizable](#)
- [FileFilter](#)
- [FilenameFilter](#)
- [ObjectInput](#)
- [ObjectOutput](#)
- [Serializable](#)

## Classes

- [BufferedInputStream](#)
- [BufferedOutputStream](#)
- [BufferedReader](#)
- [BufferedWriter](#)
- [ByteArrayInputStream](#)
- [ByteArrayOutputStream](#)
- [CharArrayReader](#)
- [CharArrayWriter](#)
- [CharConversionException](#)
- [DataInputStream](#)
- [DataOutputStream](#)
- [EOFException](#)
- [File](#)
- [FileDescriptor](#)
- [FileInputStream](#)
- [FileNotFoundException](#)
- [FileOutputStream](#)
- [FileReader](#)
- [FileWriter](#)
- [FilterInputStream](#)
- [FilterOutputStream](#)
- [FilterReader](#)
- [FilterWriter](#)
- [InputStream](#)
- [InputStreamReader](#)
- [InterruptedIOException](#)
- [InvalidClassException](#)
- [InvalidObjectException](#)
- [IOException](#)
- [LineNumberInputStream](#)
- [LineNumberReader](#)
- [NotActiveException](#)
- [NotSerializableException](#)
- [ObjectStreamException](#)
- [OptionalDataException](#)
- [OutputStream](#)

- [OutputStreamWriter](#)
- [PipedInputStream](#)
- [PipedOutputStream](#)
- [PipedReader](#)
- [PipedWriter](#)
- [PrintStream](#)
- [PrintWriter](#)
- [PushbackInputStream](#)
- [PushbackReader](#)
- [RandomAccessFile](#)
- [Reader](#)
- [SequenceInputStream](#)
- [StreamCorruptedException](#)
- [StreamTokenizer](#)
- [StringBufferInputStream](#)
- [StringReader](#)
- [StringWriter](#)
- [SyncFailedException](#)
- [UnsupportedEncodingException](#)
- [UTFDataFormatException](#)
- [WriteAbortedException](#)
- [Writer](#)

## Package [java.io](#)

### Class **BufferedInputStream**

extends [FilterInputStream](#) → [InputStream](#) → [Object](#)

Adds the ability to buffer the input and the mark and reset methods to another input stream. When a `BufferedInputStream` is created, an internal buffer array is created. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time. The mark operation remembers a point in the input stream and the reset operation causes all the bytes read since the most recent mark operation to be reread before new bytes are taken from the contained input stream.

#### Public Constructors

- [BufferedInputStream](#)

#### Public Methods

- [available](#)
- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [reset](#)
- [skip](#)

#### Methods inherited from [java.io.FilterInputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)

- [read](#)
- [close](#)

#### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [BufferedInputStream](#)

#### Public Constructor BufferedInputStream

```
BufferedInputStream(InputStream in)
```

```
BufferedInputStream(InputStream in, int bufsize)
```

#### Description:

##### constructor **BufferedInputStream(InputStream in):**

Creates a BufferedInputStream and saves its argument, the input stream in, for later use.

##### constructor **BufferedInputStream(InputStream in, int bufsize):**

Creates a BufferedInputStream with the specified buffer size, and saves its argument, the input stream in, for later use.

#### Package [java.io](#)

### Class [BufferedInputStream](#)

#### Public Method available

```
int available()
```

#### Overrides:

- [available](#) in class [FilterInputStream](#)



**Throws:**

- [IOException](#)

**Description:**

Returns the number of bytes that can be read from this input stream without blocking.

Package [java.io](#)

**Class [BufferedInputStream](#)****Public Method close**

```
void close ()
```

**Overrides:**

- [close](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Closes this input stream and releases any system resources associated with the stream.

Package [java.io](#)

**Class [BufferedInputStream](#)****Public Method mark**

```
void mark (int readlimit)
```

**Overrides:**

- [mark](#) in class [FilterInputStream](#)

**Description:**

Set a mark at the current position in the input stream. Return to the marked position by calling the reset method.

Package [java.io](#)

**Class [BufferedInputStream](#)****Public Method markSupported**

```
boolean markSupported ()
```

**Overrides:**

- [markSupported](#) in class [FilterInputStream](#)

**Description:**

Checks if the input stream supports the mark and reset methods.

Package [java.io](#)

**Class [BufferedInputStream](#)****Public Method [read](#)**

```
int read()
```

**Overrides:**

- [read](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:****read() method:**

Reads the next byte from this input stream. The byte is returned as an int in the range of 0 to 255. If the end of the stream is reached -1 is returned. This method blocks until input data is available, the end of the stream is reached or an exception is thrown.

**read(byte[] buf, int offset, int len) method:**

Reads up to *len* bytes from this input stream into the array *buf* starting at *offset*. The method blocks until input is available.

Package [java.io](#)

**Class [BufferedInputStream](#)****Public Method [reset](#)**

```
void reset()
```

**Overrides:**

- [reset](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Repositions this stream to the position that was marked with the mark method.

Package [java.io](#)

**Class [BufferedInputStream](#)****Public Method [skip](#)**

```
int skip(int num_bytes)
```

**Overrides:**

- [skip](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Skips over and discards *num\_bytes* bytes of data from this input stream. A smaller number of bytes (down to 0) might be skipped. The actual number of skipped bytes is returned.

Package [java.io](#)

**Class [BufferedOutputStream](#)**

extends [FilterOutputStream](#) → [OutputStream](#) → [Object](#)

Writing to a buffered output stream doesn't necessarily cause a call to the underlying system for each byte. Instead the data is written into an internal buffer and then to the underlying stream when the buffer is full, the buffered output stream is closed or explicitly flushed.

**Public Constructors**

- [BufferedOutputStream](#)

**Public Methods**

- [flush](#)
- [write](#)

**Methods inherited from [java.io.FilterOutputStream](#)**

- [close](#)
- [flush](#)
- [write](#)

## Methods inherited from [java.io.OutputStream](#)

- [flush](#)
- [close](#)
- [write](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [BufferedOutputStream](#)

#### **Public Constructor [BufferedOutputStream](#)**

```
BufferedOutputStream(OutputStream out)
```

```
BufferedOutputStream(OutputStream out, int size)
```

#### **Description:**

##### **constructor [BufferedOutputStream\(OutputStream out\)](#):**

Creates a new buffered output stream for writing to the underlying output stream *out* with a default size of 512 bytes.

##### **constructor [BufferedOutputStream\(OutputStream out, int size\)](#):**

Creates a buffered output stream for writing to the underlying output stream *out* with a specified buffer size of *size*-bytes.

## Package [java.io](#)

### Class [BufferedOutputStream](#)

#### **Public Method [flush](#)**

```
void flush ()
```

#### **Overrides:**

- [flush](#) in class [FilterOutputStream](#)

#### **Throws:**

- [IOException](#)

#### **Description:**

Flushes this buffered output stream, forcing all buffered output bytes to be written to the underlying output stream.

**Package [java.io](#)****Class [BufferedOutputStream](#)****Public Method [write](#)**

```
void write(byte[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [FilterOutputStream](#)

**Throws:**

- [IOException](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [FilterOutputStream](#)

**Throws:**

- [IOException](#)

**Description:****`write(byte[] buf, int offset, int len)` method:**

Writes *len* bytes from the array *buf* starting at *offset* to this buffered output stream.

This method stores bytes from the input array to the internal buffer of the stream, flushing the buffer if necessary.

**`write(int b)` method:**

Writes the specified byte to this buffered output stream.

**Package [java.io](#)****Class [BufferedReader](#)**

extends [Reader](#) → [Object](#)

Class for reading text from a character input stream, buffering characters for improving efficiency.

In general it is advisable to wrap a [BufferedReader](#) around any [Reader](#) (i.e. [FileReader](#)) whose read operation might be costly, for example:

```
BufferedReader in = new BufferedReader(new FileReader("file.in"));
```

**Public Constructors**

- [BufferedReader](#)

## Public Methods

- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [readLine](#)
- [ready](#)
- [reset](#)
- [skip](#)

## Methods inherited from [java.io.Reader](#)

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [BufferedReader](#)

#### Public Constructor [BufferedReader](#)

```
BufferedReader(Reader in)
```

```
BufferedReader(Reader in, int size)
```

#### Description:

##### constructor [BufferedReader\(Reader in\)](#):

Create a buffering character-input stream that uses a default-sized input buffer.

##### constructor [BufferedReader\(Reader in, int size\)](#):

Create a buffering character-input stream that uses an input buffer of the specified size.

## Package [java.io](#)

### Class [BufferedReader](#)

#### Public Method [close](#)

```
void close()
```

**Overrides:**

- [close](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Close the stream.

Package [java.io](#)

**Class [BufferedReader](#)****Public Method mark**

```
void mark(int readLimit)
```

**Overrides:**

- [mark](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Set a mark at the current position in the input stream. Return to the marked position by calling the reset method.

The parameter *readLimit* specifies the Limit on the number of characters that may be read while still preserving the mark. After reading this many characters, attempting to reset the stream may fail. A limit value larger than the size of the input buffer will cause a new buffer to be allocated whose size is no smaller than limit. Therefore large values should be used with care.

Package [java.io](#)

**Class [BufferedReader](#)****Public Method markSupported**

```
boolean markSupported()
```

**Overrides:**

- [markSupported](#) in class [Reader](#)

**Description:**

Checks if the input stream supports the mark and reset methods.

**Package [java.io](#)****Class [BufferedReader](#)****Public Method read**

```
int read()
```

**Overrides:**

- [read](#) in class [Reader](#)

**Throws:**

- [IOException](#)

```
int read(char[] buf, int offset, int count)
```

**Overrides:**

- [read](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:****read() method:**

Read a single character.

**read(char[] buf, int offset, int count) method:**

Read *count* characters into the array *buf*, starting at index *offset*. Returns the number of characters read or -1 if the end of the stream has been reached.

**Package [java.io](#)****Class [BufferedReader](#)****Public Method readLine**

```
String readLine()
```

**Throws:**

- [IOException](#)

**Description:**

Read a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a line feed. Returns a string containing the contents of the line without any line terminating characters or null if the end of the stream has been reached.



Package [java.io](#)  
Class [BufferedReader](#)

**Public Method [ready](#)**

```
boolean ready()
```

**Overrides:**

- [ready](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Checks if the stream is ready to be read. A buffered character stream is ready if the buffer is not empty or the underlying character stream is ready.

Package [java.io](#)  
Class [BufferedReader](#)

**Public Method [reset](#)**

```
void reset()
```

**Overrides:**

- [reset](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Reset the stream to the most recent mark.

Package [java.io](#)  
Class [BufferedReader](#)

**Public Method [skip](#)**

```
int skip(int count)
```

**Overrides:**

- [skip](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Skip *count* characters.

Package [java.io](#)

**Class [BufferedWriter](#)**

extends [Writer](#) → [Object](#)

Class for writing text to a character output stream, buffering characters for efficiency.

In general it is advisable to wrap a `BufferedWriter` around any `Writer` (i.e. `FileWriter`) whose write operation might be costly, for example:

```
PrintWriter in = new PrintWriter(new BufferedWriter(new
FileWriter("file.out")));
```

**Public Constructors**

- [BufferedWriter](#)

**Public Methods**

- [close](#)
- [flush](#)
- [newLine](#)
- [write](#)

**Methods inherited from [java.io.Writer](#)**

- [flush](#)
- [close](#)
- [write](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

**Class [BufferedWriter](#)****Public Constructor [BufferedWriter](#)**

```
BufferedWriter(Writer out)
```

```
BufferedWriter(Writer ox, int size)
```

**Description:****constructor `BufferedWriter(Writer out):`**

Create a buffered character output stream that uses a default sized buffer.

**constructor `BufferedWriter(Writer ox, int size):`**

Create a buffered character output stream with the specified buffer size.

Package [java.io](#)

**Class [BufferedWriter](#)****Public Method `close`**

```
void close ()
```

**Overrides:**

- [close](#) in class [Writer](#)

**Throws:**

- [IOException](#)

**Description:**

Close this stream.

Package [java.io](#)

**Class [BufferedWriter](#)****Public Method `flush`**

```
void flush ()
```

**Overrides:**

- [flush](#) in class [Writer](#)

**Throws:**

- [IOException](#)

**Description:**

Flush this stream.

Package [java.io](#)

**Class [BufferedWriter](#)****Public Method `newLine`**

```
void newLine ()
```

**Throws:**

- [IOException](#)

**Description:**

Write a line separator. The result of the call `System.getProperty("line.separator", "\n")` will be used.

Package [java.io](#)

**Class [BufferedWriter](#)****Public Method write**

```
void write(char[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

```
void write(int oneChar)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Throws:**

- [IOException](#)

```
void write(String str, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****`write(char[] buf, int offset, int len)` method:**

Write *len* characters of the character array *buf*, starting at *offset*.

**`write(int oneChar)` method:**

Write a single character.

**write(String str, int offset, int len) method:**

Write portion (*len* characters) of a string, starting at *offset*.

**Package [java.io](#)****Class [ByteArrayInputStream](#)**

extends [InputStream](#) → [Object](#)

A [ByteArrayInputStream](#) contains an internal buffer containing bytes that may be read from the stream. An internal counter keeps track of the next byte to be supplied by the read method.

**Public Constructors**

- [ByteArrayInputStream](#)

**Public Methods**

- [available](#)
- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [reset](#)
- [skip](#)

**Methods inherited from [java.io.InputStream](#)**

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [ByteArrayInputStream](#)****Public Constructor [ByteArrayInputStream](#)**

```
ByteArrayInputStream(byte[] buf)
```

```
ByteArrayInputStream(byte[] buf, int offset, int length)
```

**Description:****constructor `ByteArrayInputStream(byte[] buf)`:**

Creates a `ByteArrayInputStream` so that it uses *buf* as its buffer array. The buffer array is not copied. The initial position in the buffer is at index 0, the internal counter is the array length.

**constructor `ByteArrayInputStream(byte[] buf, int offset, int length)`:**

Creates a `ByteArrayInputStream` so that it uses *buf* as its buffer array. The buffer array is not copied. The initial position in the buffer is *offset* and the internal counter is set to the minimum of *offset+length* and *buf.length*.

Package [java.io](#)

**Class [ByteArrayInputStream](#)****Public Method available**

```
int available ()
```

**Overrides:**

- [available](#) in class [InputStream](#)

**Description:**

Returns the number of remaining bytes that can be read (or skipped over) from this input stream.

Package [java.io](#)

**Class [ByteArrayInputStream](#)****Public Method close**

```
void close ()
```

**Overrides:**

- [close](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Closing a `ByteArrayInputStream` has no effect. The methods in this class can be called after the stream has been closed without generating an `IOException`.

Package [java.io](#)

**Class [ByteArrayInputStream](#)****Public Method mark**

```
void mark (int readlimit)
```

**Overrides:**

- [mark](#) in class [InputStream](#)

**Description:**

The currently marked position in the stream. `ByteArrayInputStream` objects are marked at position zero by default when constructed. They may be marked at another position within the buffer by the `mark()` method. The current buffer position is set to this point by the `reset()` method.

Package [java.io](#)

**Class [ByteArrayInputStream](#)****Public Method `markSupported`**

```
boolean markSupported()
```

**Overrides:**

- [markSupported](#) in class [InputStream](#)

**Description:**

Checks if `ByteArrayInputStream` supports `mark/reset`.

Package [java.io](#)

**Class [ByteArrayInputStream](#)****Public Method `read`**

```
int read()
```

**Overrides:**

- [read](#) in class [InputStream](#)

```
int read(byte[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [InputStream](#)

**Description:****`read()` method:**

Reads the next byte from the input stream. The returned value is an `int` between 0 and 255. If the end of the stream is reached, -1 is returned.

**`read(byte[] buf, int offset, int len)` method:**

Reads up to *len* bytes from the stream into the array *buf*. If the end of the stream is reached, -1 is returned, otherwise the number of read bytes is returned.

Package [java.io](#)

## Class [ByteArrayInputStream](#)

### Public Method [reset](#)

```
void reset()
```

#### Overrides:

- [reset](#) in class [InputStream](#)

#### Description:

Resets the buffer to the marked position, by default 0.

Package [java.io](#)

## Class [ByteArrayInputStream](#)

### Public Method [skip](#)

```
int skip(int num_bytes)
```

#### Overrides:

- [skip](#) in class [InputStream](#)

#### Description:

Skips *num\_bytes* bytes from this input stream. Fewer bytes might be skipped if the end of the stream is reached. The number of actually skipped bytes is returned.

Package [java.io](#)

## Class [ByteArrayOutputStream](#)

extends [OutputStream](#) → [Object](#)

Implements an output stream for writing data into a byte array. The buffer grows automatically when data is written to it. The data can be retrieved with calls to *toByteArray* and *toString*.

### Public Constructors

- [ByteArrayOutputStream](#)

### Public Methods

- [reset](#)
- [size](#)
- [toByteArray](#)
- [toString](#)
- [write](#)
- [writeTo](#)

### Methods inherited from [java.io.OutputStream](#)

- [flush](#)



- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [ByteArrayOutputStream](#)

#### Public Constructor ByteArrayOutputStream

```
ByteArrayOutputStream()
```

```
ByteArrayOutputStream(int size)
```

#### Description:

##### constructor [ByteArrayOutputStream\(\)](#):

Create a new byte array output stream, with initial buffer size of 32 bytes.

##### constructor [ByteArrayOutputStream\(int size\)](#):

Create a new byte array output stream, with the specified initial buffer size.

#### Package [java.io](#)

### Class [ByteArrayOutputStream](#)

#### Public Method reset

```
void reset()
```

#### Description:

Reset the size of this byte array to zero.

#### Package [java.io](#)

### Class [ByteArrayOutputStream](#)

#### Public Method size

```
int size()
```

#### Description:

Returns the current size of the buffer.

Package [java.io](#)

## Class [ByteArrayOutputStream](#)

### Public Method toByteArray

```
byte[] toByteArray()
```

#### Description:

Creates a new byte array with the current size of the this output stream and the valid content of the buffer copied to it.

Package [java.io](#)

## Class [ByteArrayOutputStream](#)

### Public Method toString

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

#### Description:

Converts the buffer content into a string.

Package [java.io](#)

## Class [ByteArrayOutputStream](#)

### Public Method write

```
void write(byte[] buffer, int offset, int len)
```

#### Overrides:

- [write](#) in class [OutputStream](#)

```
void write(int oneByte)
```

#### Overrides:

- [write](#) in class [OutputStream](#)

#### Description:

##### **write(byte[] buffer, int offset, int add) method:**

Writes *len* bytes from the specified byte array starting at *offset* to this byte array output stream.

##### **write(int oneByte) method:**

Writes the specified byte to this byte array output stream.

Package [java.io](#)

## Class [ByteArrayOutputStream](#)

### Public Method [writeTo](#)

```
void writeTo(OutputStream out)
```

#### Throws:

- [IOException](#)

#### Description:

Writes the complete contents of this byte array output stream to the output stream *out*.

Package [java.io](#)

## Class [CharArrayReader](#)

extends [Reader](#) → [Object](#)

This class implements a character buffer that can be used as a character-input stream.

### Public Constructors

- [CharArrayReader](#)

### Public Methods

- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [ready](#)
- [reset](#)
- [skip](#)

### Methods inherited from [java.io.Reader](#)

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [CharArrayReader](#)

### Public Constructor CharArrayReader

```
CharArrayReader(char[] buf)
```

```
CharArrayReader(char[] buf, int offset, int length)
```

#### Description:

##### constructor CharArrayReader(char[] buf):

Constructs an *CharArrayReader* from the specified array of chars.

##### constructor CharArrayReader(char[] buf, int offset, int length):

Constructs an *CharArrayReader* from the specified array of chars. The *buf* parameter defines the size of the input buffer. The *offset* parameter defines the offset of the first character to read. The *length* parameter defines the number of chars to read.

Package [java.io](#)

## Class [CharArrayReader](#)

### Public Method close

```
void close()
```

#### Overrides:

- [close](#) in class [Reader](#)

#### Throws:

- [IOException](#)

#### Description:

This method close the stream.

#### Example

```
char[] ch = {'H','e','l','l','o'};
CharArrayReader car = new CharArrayReader(ch);
car.close();
```

After the construction of a *CharArrayReader*, the stream is closed.

Package [java.io](#)

## Class [CharArrayReader](#)

### Public Method mark

```
void mark(int readlimit)
```

**Overrides:**

- [mark](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method marks the present position in the stream. Subsequent calls to *reset* method will reposition the stream to this point.

**Example**

```
char[] ch = {'H','e','l','l','o'};
CharArrayReader car = new CharArrayReader(ch);
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
car.reset();
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
car.mark();
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
car.reset();
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
```

The *mark* method sets up a new start position. The *reset* method walks to the mark position after mark method call. The subsequent call of the *read* method starts from this point.

**Package [java.io](#)****Class [CharArrayReader](#)****Public Method markSupported**

```
boolean markSupported()
```

**Overrides:**

- [markSupported](#) in class [Reader](#)

**Description:**

This method checks whether this stream supports the mark operation or not.

**Example**

```
char[] ch = {'H','e','l','l','o'};
CharArrayReader car = new CharArrayReader(ch);
if (car.markSupported())
    Logger.log("mark is supported");
```

The *markSupported* method checks if the mark operation is supported.

Package [java.io](#)

## Class [CharArrayReader](#)

### Public Method `read`

```
int read()
```

#### Overrides:

- [read](#) in class [Reader](#)

```
int read(char[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [Reader](#)

#### Description:

##### `read()` method:

This method reads a single character.

##### `read(char[] buf, int offset, int len)` method:

This method reads the characters in to a destination char array buffer. The *offset* parameter defines the offset of characters to read. The *len* parameter defines the maximum number of characters to read.

#### Example

```
char[] ch = {'H','e','l','l','o'};
char[] buff = new char[5];
CharArrayReader car = new CharArrayReader(ch);
car.read(buff, 0, 2);
for (int i = 0; i < 2; i++)
    Logger.log("" + i + " " + buff[i]);
```

The *read* method reads the stream into the provided buffer for later use.

Package [java.io](#)

## Class [CharArrayReader](#)

### Public Method `ready`

```
boolean ready()
```

#### Overrides:

- [ready](#) in class [Reader](#)

**Description:**

This method checks whether this stream is ready to be read or not.

**Example**

```
char[] ch = {'H','e','l','l','o'};
CharArrayReader car = new CharArrayReader(ch);
if (car.ready())
    Logger.log("The stream is ready to be read!");
```

The *ready* method checks if the stream can be read.

Package [java.io](#)

**Class [CharArrayReader](#)****Public Method reset**

```
void reset()
```

**Overrides:**

- [reset](#) in class [Reader](#)

**Description:**

This method resets the stream to the most recent mark, or to the beginning if it has never been marked.

**Example**

```
char[] ch = {'H','e','l','l','o'};
CharArrayReader car = new CharArrayReader(ch);
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
car.reset();
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
car.mark();
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
car.reset();
LogWriter("" + (char) car.read());
LogWriter("" + (char) car.read());
```

The *mark* method sets up a new start position. The *reset* method walks to the mark position after mark method call. The subsequent call of the *read* method starts from this point.

**Package [java.io](#)****Class [CharArrayReader](#)****Public Method [skip](#)**

```
int skip(int num_chars)
```

**Overrides:**

- [skip](#) in class [Reader](#)

**Description:**

This method skips the specified number of characters.

**Example**

```
char[] ch = {'H','e','l','l','o'};
CharArrayReader car = new CharArrayReader(ch);
Logger.log("The first element of the stream: " + car.read());
car.skip(2);
Logger.log("The next element the fourth: " + car.read());
```

The *skip* method skips the specified characters.

**Package [java.io](#)****Class [CharArrayWriter](#)**

extends [Writer](#) → [Object](#)

This class implements a character buffer that can be used as a *Writer*. The buffer automatically grows when data is written to the stream. The data can be retrieved using the *toCharArray* and *toString* methods.

**Public Constructors**

- [CharArrayWriter](#)

**Public Methods**

- [close](#)
- [flush](#)
- [reset](#)
- [size](#)
- [toCharArray](#)
- [toString](#)
- [write](#)
- [writeTo](#)

**Methods inherited from [java.io.Writer](#)**

- [flush](#)
- [close](#)
- [write](#)



## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [CharArrayWriter](#)

#### **Public Constructor CharArrayWriter**

```
CharArrayWriter ()
```

```
CharArrayWriter (int size)
```

#### **Description:**

##### **constructor CharArrayWriter():**

This method constructs a new *CharArrayWriter*.

##### **constructor CharArrayWriter(int size):**

This method constructs a new *CharArrayWriter* with the specified initial size.

## Package [java.io](#)

### Class [CharArrayWriter](#)

#### **Public Method close**

```
void close ()
```

#### **Overrides:**

- [close](#) in class [Writer](#)

#### **Description:**

This method closes the stream and does not release the buffer.

#### **Example**

```
char[] ch = {'H','e','l','l','o'};
CharArrayWriter caw = new CharArrayWriter();
caw.close();
```

After the construction of a *CharArrayWriter*, the stream is closed.

Package [java.io](#)

## Class [CharArrayWriter](#)

### Public Method flush

```
void flush()
```

#### Overrides:

- [flush](#) in class [Writer](#)

#### Description:

This method flush the stream.

#### Example

```
char[] ch = {'H','e','l','l','o'};
CharArrayWriter caw = new CharArrayWriter();
caw.flush();
```

After the construction of a *CharArrayWriter*, the stream is flushed.

Package [java.io](#)

## Class [CharArrayWriter](#)

### Public Method reset

```
void reset()
```

#### Description:

This method resets the buffer. The buffer can be used again without throwing away the already allocated buffer.

#### Example

```
char[] ch = {'H','E','L','L','O'};
char[] chnew = {'h','e','l','l','o'};
CharArrayWriter caw = new CharArrayWriter();
caw.write(ch);
Logger.log("" + caw.toString());
caw.reset();
caw.write(chnew);
Logger.log("" + caw.toString());
```

After the construction of a *CharArrayWriter*, the stream is reseted.

Package [java.io](#)

## Class [CharArrayWriter](#)

### Public Method size

```
int size()
```

#### Description:

This method returns the current size of the buffer.

#### Example

```
char[] ch = {'H', 'E', 'L', 'L', 'O'};
CharArrayWriter caw = new CharArrayWriter();
caw.write(ch);
Logger.log(ch.size());
```

The *size* method returns current size of the buffer.

Package [java.io](#)

## Class [CharArrayWriter](#)

### Public Method toCharArray

```
char[] toCharArray()
```

#### Description:

This method returns a copy of the input data.

#### Example

```
char[] ch = {'H', 'E', 'L', 'L', 'O'};
CharArrayWriter caw = new CharArrayWriter();
caw.write(ch);
char[] data = caw.toCharArray();
for (int i = 0; i < data.length; i++)
    Console.print("" + data[i]);
Console.println("");
```

The *toCharArray* method returns a copy of the input data.

Package [java.io](#)

## Class [CharArrayWriter](#)

### Public Method toString

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

**Description:**

This method converts the input data to a string.

**Example**

```
char[] ch = {'H', 'E', 'L', 'L', 'O'};
CharArrayWriter caw = new CharArrayWriter();
caw.write(ch);
Console.println(caw.toString());
```

The *toString* method returns a string copy of the input data.

Package [java.io](#)

**Class [CharArrayWriter](#)****Public Method write**

```
void write(char[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(String str, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Description:****write(char[] buf, int offset, int len) method:**

This method writes the defined character array to the internal buffer. The *offset* parameter defines the offset into the character array. The *len* parameter defines the number of characters to write.

**write(int b) method:**

This method writes a character to the buffer.

**write(String str, int offset, int len) method:**

This method writes the defined string to the internal character buffer. The *offset* parameter defines the offset into the character of the string. The *len* parameter defines the number of characters to write.

## Example

```
char[] ch = {'H', 'E', 'L', 'L', 'O'};
CharArrayWriter caw = new CharArrayWriter();
caw.write(ch);
```

The `write` method writes the elements of the character array to the buffer.

Package [java.io](#)

## Class [CharArrayWriter](#)

### Public Method `writeTo`

```
void writeTo(Writer out)
```

#### Throws:

- [IOException](#)

#### Description:

This method writes the content of the buffer to another character stream.

## Example

```
char[] ch = {'H', 'E', 'L', 'L', 'O'};
caw1 = new CharArrayWriter();
caw2 = new CharArrayWriter();
caw1.write(ch);
caw1.writeTo(caw2);
Console.println(caw2.toString());
```

The `writeTo` method writes the content of the buffer to the other stream.

Package [java.io](#)

## Class `CharConversionException`

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Base class for character conversion exceptions.

### Public Constructors

- [CharConversionException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [CharConversionException](#)

#### Public Constructor CharConversionException

```
CharConversionException()
```

```
CharConversionException(String message)
```

#### Description:

##### constructor CharConversionException():

Create Exception null as its detailed message.

##### constructor CharConversionException(String message):

Create Exception with detailed message.

## Package [java.io](#)

### Interface [DataInput](#)

Interface for reading bytes from a binary stream into Java primitive types.

#### Public Methods

- [readBoolean](#)
- [readByte](#)
- [readChar](#)
- [readFloat](#)
- [readFully](#)
- [readInt](#)
- [readLine](#)
- [readShort](#)
- [readUnsignedByte](#)
- [readUnsignedShort](#)
- [readUTF](#)
- [skipBytes](#)

## Package [java.io](#)

### Interface [DataInput](#)

#### Public Method readBoolean

```
abstract boolean readBoolean()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads one input byte and returns true if that byte is nonzero, false otherwise.

Package [java.io](#)

**Interface [DataInput](#)****Public Method `readByte`**

```
abstract byte readByte()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads and returns on input byte. The byte value is between -128 and 127.

Package [java.io](#)

**Interface [DataInput](#)****Public Method `readChar`**

```
abstract char readChar()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads an input character and returns a char value.

Package [java.io](#)

**Interface [DataInput](#)****Public Method `readFloat`**

```
abstract float readFloat()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads four input bytes and returns a float value.

Package [java.io](#)

**Interface [DataInput](#)****Public Method [readFully](#)**

```
abstract void readFully(byte[] buf)
```

**Throws:**

- [EOFException](#)
- [IOException](#)

```
abstract void readFully(byte[] buf, int offset, int len)
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:****[readFully\(byte\[\] buf\)](#) method:**

Reads *b.length* bytes from the input stream and stores them in *buf*.

**[readFully\(byte\[\] buf, int offset, int len\)](#) method:**

Reads *len* bytes into the buffer *buf*, starting at the buffer-offset *offset*.

Package [java.io](#)

**Interface [DataInput](#)****Public Method [readInt](#)**

```
abstract int readInt()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads four input bytes and returns an int value.

Package [java.io](#)

**Interface [DataInput](#)****Public Method [readLine](#)**

```
abstract String readLine()
```



**Throws:**

- [IOException](#)

**Description:**

Reads the next line of text from the input stream, converting each byte into a character until a line terminator or end of file is encountered. If nothing could be read *null* is returned.

Package [java.io](#)

**Interface [DataInput](#)****Public Method readShort**

```
abstract short readShort()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads two input bytes and returns a short.

Package [java.io](#)

**Interface [DataInput](#)****Public Method readUnsignedByte**

```
abstract int readUnsignedByte()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads one input byte and converts it into an int in the range of 0 to 255.

Package [java.io](#)

**Interface [DataInput](#)****Public Method readUnsignedShort**

```
abstract int readUnsignedShort()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads two input bytes and returns an int in the range of 0 to 65535.

Package [java.io](#)

**Interface [DataInput](#)****Public Method [readUTF](#)**

```
abstract String readUTF()
```

**Throws:**

- [EOFException](#)
- [UTFDataFormatException](#)
- [IOException](#)

**Description:**

Reads a Unicode character encoded with UTF-8.

Package [java.io](#)

**Interface [DataInput](#)****Public Method [skipBytes](#)**

```
abstract int skipBytes(int n)
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Tries to skip over n input bytes, discarding the skipped bytes. Returns the actual number of skipped bytes, which might be smaller than n (e.g. because end of file was reached).

Package [java.io](#)

**Class [DataInputStream](#)**

extends [FilterInputStream](#) → [InputStream](#) → [Object](#)  
implements [DataInput](#)

A data input stream allows for reading primitive Java data types from an underlying input stream in a machine independent manner. Data input and data output streams represent Unicode strings in a UTF-8 format.

**Public Constructors**

- [DataInputStream](#)

**Public Methods**

- [read](#)

- [readBoolean](#)
- [readByte](#)
- [readChar](#)
- [readFloat](#)
- [readFully](#)
- [readInt](#)
- [readLine](#)
- [readShort](#)
- [readUnsignedByte](#)
- [readUnsignedShort](#)
- [readUTF](#)
- [skipBytes](#)

#### Methods inherited from [java.io.FilterInputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

#### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [DataInputStream](#)

#### **Public Constructor DataInputStream**

```
DataInputStream(InputStream in)
```

#### **Description:**

Creates a DataInputStream that uses the specified underlying InputStream.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method [read](#)

```
final int read(byte[] b)
```

#### Overrides:

- [read](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

```
final int read(byte[] b, int off, int len)
```

#### Overrides:

- [read](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

#### Description:

##### **read(byte[] b) method:**

Reads *b.length* bytes from underlying input stream into the buffer *b*. The total number of bytes read is returned. If there is no more data to read -1 is returned.

##### **read(byte[] b, int off, int len) method:**

Reads up to *len* bytes of data into the buffer *b*, starting at array offset *off*. The number of bytes actually read is returned. If *len* is 0 no bytes are read and 0 is returned. If there is no more data to read -1 is returned.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method [readBoolean](#)

```
final boolean readBoolean()
```

#### Throws:

- [IOException](#)

#### Description:

Reads one input byte and returns true if that byte is nonzero, false otherwise.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method readByte

```
final byte readByte()
```

#### Throws:

- [IOException](#)

#### Description:

Reads and returns on input byte. The byte value is between -128 and 127.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method readChar

```
final char readChar()
```

#### Throws:

- [IOException](#)
- [EOFException](#)

#### Description:

Reads an input character and returns a char value.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method readFloat

```
final float readFloat()
```

#### Throws:

- [IOException](#)

#### Description:

Reads four input bytes and returns a float value.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method readFully

```
final void readFully(byte[] b)
```

**Throws:**

- [IOException](#)

```
final void readFully(byte[] b, int off, int len)
```

**Throws:**

- [IOException](#)
- [EOFException](#)

**Description:****readFully(byte[] buf) method:**

Reads *b.length* bytes from the input stream and stores them in *buf*.

**readFully(byte[] buf, int offset, int len) method:**

Reads *len* bytes into the buffer *buf*, starting at the buffer-offset *offset*.

Package [java.io](#)

**Class [DataInputStream](#)****Public Method readInt**

```
final int readInt()
```

**Throws:**

- [IOException](#)
- [EOFException](#)

**Description:**

Reads four input bytes and returns an int value.

Package [java.io](#)

**Class [DataInputStream](#)****Public Method readLine**

```
final String readLine()
```

**Throws:**

- [IOException](#)

**Description:**

Reads the next line of text from the input stream, converting each byte into a character until a line terminator or end of file is encountered. If nothing could be read *null* is returned.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method [readShort](#)

```
final short readShort()
```

#### Throws:

- [IOException](#)
- [EOFException](#)

#### Description:

Reads two input bytes and returns a short.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method [readUnsignedByte](#)

```
final int readUnsignedByte()
```

#### Throws:

- [IOException](#)

#### Description:

Reads one input byte and converts it into an int in the range of 0 to 255.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method [readUnsignedShort](#)

```
final int readUnsignedShort()
```

#### Throws:

- [IOException](#)
- [EOFException](#)

#### Description:

Reads two input bytes and returns an int in the range of 0 to 65535.

Package [java.io](#)

## Class [DataInputStream](#)

### Public Method [readUTF](#)

```
final String readUTF()
```

**Throws:**

- [IOException](#)

```
final static String readUTF(DataInput in)
```

**Throws:**

- [IOException](#)

**Description:****readUTF() method:**

Reads a Unicode character from the underlying stream encoded with UTF-8 and returns it as a string.

**readUTF(DataInput in) method:**

Reads a Unicode character from the input stream *in* encoded with UTF-8. The character is returned as a string.

Package [java.io](#)

**Class [DataInputStream](#)****Public Method [skipBytes](#)**

```
final int skipBytes(int n)
```

**Throws:**

- [IOException](#)

**Description:**

Tries to skip over *n* input bytes, discarding the skipped bytes. Returns the actual number of skipped bytes, which might be smaller than *n* (e.g. because end of file was reached).

Package [java.io](#)

**Interface [DataOutput](#)**

Interface for converting data from Java primitive types to a binary stream.

**Public Methods**

- [write](#)
- [writeBoolean](#)
- [writeByte](#)
- [writeBytes](#)
- [writeChar](#)
- [writeChars](#)
- [writeFloat](#)
- [writeInt](#)
- [writeShort](#)
- [writeUTF](#)



**Package [java.io](#)****Interface [DataOutput](#)****Public Method [write](#)**

```
abstract void write(byte[] buf)
```

**Throws:**

- [IOException](#)

```
abstract void write(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)

```
abstract void write(int value)
```

**Throws:**

- [IOException](#)

**Description:****`write(byte[] buf)` method:**

Writes all bytes of *buf* to the output stream.

**`write(byte[] buf, int offset, int len)` method:**

Writes *len* bytes from *buf*, starting at offset *offset*, to the output stream.

**`write(int value)` method:**

Writes the lower eight bits of *value* to the output stream. The 24 higher bits are ignored.

**Package [java.io](#)****Interface [DataOutput](#)****Public Method [writeBoolean](#)**

```
abstract void writeBoolean(boolean value)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a boolean value to the underlying output stream as a byte value. The value *true* is (byte)1, the value *false* is represented as (byte)0.

Package [java.io](#)

## Interface [DataOutput](#)

### Public Method writeByte

```
abstract void writeByte(int value)
```

#### Throws:

- [IOException](#)

#### Description:

Writes the lower eight bits of *b* to the output stream. The 24 higher bits are ignored.

Package [java.io](#)

## Interface [DataOutput](#)

### Public Method writeBytes

```
abstract void writeBytes(String value)
```

#### Throws:

- [IOException](#)

#### Description:

Writes the specified string to the underlying stream as a sequence of bytes, using only the first byte of the UTF-8 representation. Take care with Unicode symbols that take more than one byte when represented in UTF-8 code.

Package [java.io](#)

## Interface [DataOutput](#)

### Public Method writeChar

```
abstract void writeChar(int value)
```

#### Throws:

- [IOException](#)

#### Description:

Writes a character to the output stream as a one byte value.

Package [java.io](#)

## Interface [DataOutput](#)

### Public Method writeChars

```
abstract void writeChars(String value)
```

**Throws:**

- [IOException](#)

**Description:**

Writes every character in the string *s* to the output stream, one byte per character.

Package [java.io](#)

**Interface [DataOutput](#)****Public Method writeFloat**

```
abstract void writeFloat(float value)
```

**Throws:**

- [IOException](#)

**Description:**

Converts the float argument to an int using the `floatToIntBits` method in class `Float`, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.

Package [java.io](#)

**Interface [DataOutput](#)****Public Method writeInt**

```
abstract void writeInt(int value)
```

**Throws:**

- [IOException](#)

**Description:**

Writes an int to the underlying output stream as four bytes, high byte first.

Package [java.io](#)

**Interface [DataOutput](#)****Public Method writeShort**

```
abstract void writeShort(int value)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a short to the underlying output stream as two bytes, high byte first.

**Package [java.io](#)****Interface [DataOutput](#)****Public Method [writeUTF](#)**

```
abstract void writeUTF(String value)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a string to the underlying output stream using UTF-8 encoding.

The first two bytes written give the number of bytes to follow, i.e. the number of bytes actually written (not the length of the string).

**Package [java.io](#)****Class [DataOutputStream](#)**

extends [FilterOutputStream](#) → [OutputStream](#) → [Object](#)

implements [DataOutput](#)

A [DataOutputStream](#) can be used to write primitive Java data types to an output stream in a system independent way. A data input stream can then be used to read the data back.

**Public Constructors**

- [DataOutputStream](#)

**Public Methods**

- [flush](#)
- [size](#)
- [write](#)
- [writeBoolean](#)
- [writeByte](#)
- [writeBytes](#)
- [writeChar](#)
- [writeChars](#)
- [writeFloat](#)
- [writeInt](#)
- [writeShort](#)
- [writeUTF](#)

**Methods inherited from [java.io.FilterOutputStream](#)**

- [close](#)
- [flush](#)
- [write](#)

**Methods inherited from [java.io.OutputStream](#)**

- [flush](#)

- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [DataOutputStream](#)

#### Public Constructor [DataOutputStream](#)

```
DataOutputStream(OutputStream out)
```

#### Description:

Creates a new data output stream to write data to the underlying output stream *out*.

#### Package [java.io](#)

### Class [DataOutputStream](#)

#### Public Method [flush](#)

```
void flush()
```

#### Overrides:

- [flush](#) in class [FilterOutputStream](#)

#### Throws:

- [IOException](#)

#### Description:

Flushes any buffered data from this stream to the underlying output stream.

#### Package [java.io](#)

### Class [DataOutputStream](#)

#### Public Method [size](#)

```
final int size()
```

#### Description:

Returns the number of bytes written to this data output stream so far.

Package [java.io](#)

## Class [DataOutputStream](#)

### Public Method write

```
void write(byte[] buf, int offset, int len)
```

#### Overrides:

- [write](#) in class [FilterOutputStream](#)

#### Throws:

- [IOException](#)

```
void write(int b)
```

#### Overrides:

- [write](#) in class [FilterOutputStream](#)

#### Throws:

- [IOException](#)

#### Description:

##### **write(byte[] buf, int offset, int len) method:**

Writes *len* bytes from *buf*, starting at *offset*, to the underlying output stream.

##### **write(int b) method:**

Writes the lower eight bits of *b* to the underlying output stream.

Package [java.io](#)

## Class [DataOutputStream](#)

### Public Method writeBoolean

```
final void writeBoolean(boolean b)
```

#### Throws:

- [IOException](#)

#### Description:

Writes a boolean value to the underlying output stream as a byte value. The value *true* is (byte)1, the value *false* is represented as (byte)0.

Package [java.io](#)

## Class [DataOutputStream](#)

### Public Method writeByte

```
final void writeByte(int b)
```

#### Throws:

- [IOException](#)

#### Description:

Writes the lower eight bits of *b* to the output stream. The 24 higher bits are ignored.

Package [java.io](#)

## Class [DataOutputStream](#)

### Public Method writeBytes

```
final void writeBytes(String s)
```

#### Throws:

- [IOException](#)

#### Description:

Writes the specified string to the underlying stream as a sequence of bytes, using only the first byte of the UTF-8 representation. Take care with Unicode symbols that take more than one byte when represented in UTF-8 code.

Package [java.io](#)

## Class [DataOutputStream](#)

### Public Method writeChar

```
final void writeChar(int c)
```

#### Throws:

- [IOException](#)

#### Description:

Writes a character to the underlying output stream as a one byte value.

Package [java.io](#)

## Class [DataOutputStream](#)

### Public Method writeChars

```
final void writeChars(String s)
```

**Throws:**

- [IOException](#)

**Description:**

Writes every character in the string *s* to the underlying output stream, one byte per character.

Package [java.io](#)

**Class [DataOutputStream](#)****Public Method writeFloat**

```
final void writeFloat(float f)
```

**Throws:**

- [IOException](#)

**Description:**

Converts the float argument to an int using the `floatToIntBits` method in class `Float`, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.

Package [java.io](#)

**Class [DataOutputStream](#)****Public Method writeInt**

```
final void writeInt(int i)
```

**Throws:**

- [IOException](#)

**Description:**

Writes an int to the underlying output stream as four bytes, high byte first.

Package [java.io](#)

**Class [DataOutputStream](#)****Public Method writeShort**

```
final void writeShort(int s)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a short to the underlying output stream as two bytes, high byte first.



**Package [java.io](#)****Class [DataOutputStream](#)****Public Method [writeUTF](#)**

```
final void writeUTF(String s)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a string to the underlying output stream using UTF-8 encoding.

The first two bytes written give the number of bytes to follow, i.e. the number of bytes actually written (not the length of the string).

**Package [java.io](#)****Class [EOFException](#)**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that an end of file or end of stream has been reached unexpectedly during input.

This exception is mainly used by data input streams, which generally expect a binary file in a specific format, and for which an end of stream is an unusual condition. Most other input streams return a special value on end of stream.

**Public Constructors**

- [EOFException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [EOFException](#)****Public Constructor EOFException**

```
EOFException()
```

```
EOFException(String message)
```

**Description:****constructor EOFException():**

Create Exception null as its detailed message.

**constructor EOFException(String message):**

Create Exception with detailed message.

**Package [java.io](#)****Interface Externalizable**

implements [Serializable](#)

Object serialization uses the [Serializable](#) and Externalizable interfaces. The writeExternal and readExternal methods of the Externalizable interface are implemented by a class to give the class complete control over the format and contents of the stream for an object and its supertypes. Each object to be stored is tested for the Externalizable interface. If the object supports Externalizable, the writeExternal method is called. If the object does not support Externalizable and does implement Serializable, the object is saved using ObjectOutputStream. When an Externalizable object is reconstructed, an instance is created using the public no-arg constructor, then the readExternal method called.

**Public Methods**

- [readExternal](#)
- [writeExternal](#)

**Package [java.io](#)****Interface [Externalizable](#)****Public Method readExternal**

```
abstract void readExternal(ObjectInput in)
```

**Throws:**

- [ClassNotFoundException](#)
- [IOException](#)

**Description:**

A class implementing the Externalizable interface has to implement this method in order to restore the contents of an object from the stream *in* by calling the methods of DataInput for its primitive types or calling writeObject of ObjectOutputStream for objects, strings and arrays.

**Package [java.io](#)****Interface [Externalizable](#)****Public Method [writeExternal](#)**

```
abstract void writeExternal(ObjectOutput out)
```

**Throws:**

- [IOException](#)

**Description:**

A class implementing the Externalizable interface has to implement this method in order to save the contents of this object to the stream *out* by calling the methods of DataOutput for its primitive types or calling readObject for objects, strings and arrays. The readExternal method must read the values in the same sequence and with the same types as were written by writeExternal.

**Package [java.io](#)****Class File**

extends [Object](#)

implements [Serializable](#), [Comparable](#)

This class provides an abstract, i.e. system independent, representation of file and directory pathnames. Each name in an abstract pathname except for the last denotes a directory; the last name may denote either a directory or a file (see also [MicroSD Card Related Commands](#)).

**Comments on the iLCD file support**

Please note that the iLCD panel only supports DOS filenames in the 8.3 format, i.e. max. 8 characters for name and max. 3 characters for extension.

A maximum of 4 files can be opened at the same time. Note that the same file can be opened only once. It's e.g. not possible to read and write to a file at the same time. Make sure that any stream accessing a file is closed before a new stream for the same file is created.

Note that file paths use slashes ("/") and not backslashes ("\"). The root directory is "/".

**Public Constructors**

- [File](#)

**Public Methods**

- [canRead](#)
- [canWrite](#)
- [compareTo](#)
- [createNewFile](#)
- [createTempFile](#)
- [delete](#)
- [equals](#)
- [exists](#)
- [format](#)
- [getAbsolutePath](#)

- [getCanonicalPath](#)
- [getFreeSpace](#)
- [getName](#)
- [getParent](#)
- [getPath](#)
- [getTotalSpace](#)
- [hashCode](#)
- [isAbsolute](#)
- [isDirectory](#)
- [isFile](#)
- [isMounted](#)
- [lastModified](#)
- [length](#)
- [list](#)
- [listFiles](#)
- [mkdir](#)
- [mkdirs](#)
- [renameTo](#)
- [toString](#)
- [unmount](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- static final String pathSeparator
- static final char pathSeparatorChar
- static final String separator
- static final char separatorChar

#### Package [java.io](#)

### Class [File](#)

#### **Public Constructor File**

```
File(File dir, String name)
```

```
File(String path)
```

#### Throws:

- [NullPointerException](#)

```
File(String path, String name)
```

#### Throws:

- [NullPointerException](#)

Please note that the iLCD panel only supports DOS filenames in the 8.3 format, i.e. max. 8 characters for name and max 3 characters for extension.

Note that file paths use slashes ("/") and not backslashes ("\"). The root directory is "/".

Package [java.io](#)

Class [File](#)

### **Public Method canRead**

```
boolean canRead()
```

#### **Description:**

Tests whether the application can read the file denoted by this abstract pathname.

Returns true if and only if the file specified by this abstract pathname exists and can be read by the application.

Package [java.io](#)

Class [File](#)

### **Public Method canWrite**

```
boolean canWrite()
```

#### **Description:**

Tests whether the application can modify to the file denoted by this abstract pathname.

Returns true if and only if the file system actually contains a file denoted by this abstract pathname and the application is allowed to write to the file; false otherwise.

Package [java.io](#)

Class [File](#)

### **Public Method compareTo**

```
int compareTo(File that)
```

```
int compareTo(Object that)
```

#### **Description:**

##### **compareTo(File that) method:**

Compares two abstract pathnames lexicographically; case is ignored. Returns 0 if the argument is equal to this abstract pathname, a value less than zero if this abstract pathname is lexicographically less than the argument, or a value greater than zero if this abstract pathname is lexicographically greater than the argument.

##### **compareTo(Object that) method:**

Compares this abstract pathname to another object. If the other object is an abstract pathname, then this function behaves like compareTo(File). Otherwise, it throws a ClassCastException, since abstract

pathnames can only be compared to abstract pathnames.

Package [java.io](#)

Class [File](#)

### Public Method createNewFile

```
boolean createNewFile()
```

**Throws:**

- [IOException](#)

**Description:**

Creates a new, empty file named by the abstract pathname if and only if a file with this name does not yet exist.

Returns true if the named file does not exist and was successfully created; false if the named file already exists.

Note that any directories required must exist otherwise there will be an IOException.

Package [java.io](#)

Class [File](#)

### Public Method createTempFile

```
static File createTempFile(String prefix, String suffix)
```

**Throws:**

- [IOException](#)

```
static File createTempFile(String prefix, String suffix, File dir)
```

**Throws:**

- [IOException](#)
- [IllegalArgumentException](#)

**Description:**

#### **createTempFile(String prefix, String suffix) method:**

Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name. Invoking this method is equivalent to invoking createTempFile(prefix, suffix, null).

#### **createTempFile(String prefix, String suffix, File dir) method:**

Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name. If this method returns successfully then it is guaranteed that:

1. The file denoted by the returned abstract pathname did not exist before this method was invoked.

- Neither this method nor any of its variants will return the same abstract pathname again in the current invocation of the virtual machine.

The prefix argument must be at least three characters long. The suffix argument may be null, in which case the suffix ".tmp" will be used. Please note that the iLCD panel only supports DOS filenames in the 8.3 format, i.e. max. 8 characters for name and max 3 characters for extension.

If the directory argument is null then the system-dependent default temporary-file directory will be used.

Package [java.io](#)

Class [File](#)

### Public Method delete

```
boolean delete()
```

#### Description:

Deletes the file or directory denoted by this abstract pathname. If this pathname denotes a directory, then the directory must be empty in order to be deleted.

Package [java.io](#)

Class [File](#)

### Public Method equals

```
boolean equals(Object that)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

This method checks if this abstract pathname is equivalent to the given object. This method returns *true* if the argument is not *null* and is an abstract pathname that denotes the same file or directory as this abstract pathname.

#### Example

```
File file1 = new File("test/directory");
File file2 = new File("test/directory");
if (file1.equals(file2))
    Logger.log("These files are equivalent!");
```

The example above checks the pathnames of both files for equality.

Package [java.io](#)

Class [File](#)

### Public Method exists

```
boolean exists()
```

**Description:**

Tests whether the file (or directory) denoted by this abstract pathname exists.

**Example**

```
File file = new File("DATA.RII");
if (file.exists())
    Logger.log("this file exists");
```

The example above returns the absolute pathname.

Package [java.io](#)

Class [File](#)

**Public Method format**

```
static boolean format()
```

**Description:**

Formats the MicroSD card at runtime. All data on the MicroSD card will be deleted.

Note that this is not a standard Java method but iLCD panel related.

**See also:**

[Format MicroSD Card](#)

Package [java.io](#)

Class [File](#)

**Public Method getAbsolutePath**

```
String getAbsolutePath()
```

**Description:**

This method returns the absolute pathname string of this abstract pathname.

**Example**

```
File file = new File("test/directory");
Logger.log("get absolute path: " + file.getAbsolutePath());
```

The example above returns the absolute pathname.

Package [java.io](#)

Class [File](#)

**Public Method getCanonicalPath**

```
String getCanonicalPath()
```



**Throws:**

- [IOException](#)

**Description:**

This method returns the canonical pathname string of this abstract pathname.

**Example**

```
File file = new File("test/directory");
Logger.log("get canonical path: " + file.getCanonicalPath());
```

The example above returns the canonical pathname.

Package [java.io](#)

Class [File](#)

**Public Method getFreeSpace**

```
int getFreeSpace()
```

**Description:**

Returns free space on Micro SD Card in MB.

Package [java.io](#)

Class [File](#)

**Public Method getName**

```
String getName()
```

**Description:**

The *getName* method returns the last name of the file or directory. An empty string is returned if the pathname's name is empty.

**Example**

```
File file = new File("test/directory");
Logger.log("get name: " + file.getName());
```

The example above returns the last name of the directory.

Package [java.io](#)

Class [File](#)

**Public Method getParent**

```
String getParent()
```

**Description:**

This method returns the pathname string of this abstract pathname's parent, or *null* if this pathname does not name a parent directory.

**Example**

```
File file = new File("test/directory");
Logger.log("get parent: " + file.getParent());
```

The example above returns the directory path from the parent directory.

Package [java.io](#)

Class [File](#)

**Public Method [getPath](#)**

```
String getPath()
```

**Description:**

This method converts this abstract pathname into a pathname string.

**Example**

```
File file = new File("test/directory");
Logger.log("get path: " + file.getPath());
```

The example above returns the directory path.

Package [java.io](#)

Class [File](#)

**Public Method [getTotalSpace](#)**

```
int getTotalSpace()
```

**Description:**

Returns total space on Micro SD Card in MB.

Package [java.io](#)

Class [File](#)

**Public Method [hashCode](#)**

```
int hashCode()
```

**Overrides:**

- [hashCode](#) in class [Object](#)

**Description:**

Calculates the hash code for this abstract pathname. The result is the same as calling hashCode() for the related string.

Package [java.io](#)

Class [File](#)

**Public Method isAbsolute**

```
boolean isAbsolute()
```

**Description:**

Tests whether this abstract pathname is absolute, i.e. if it begins with "/".

Package [java.io](#)

Class [File](#)

**Public Method isDirectory**

```
boolean isDirectory()
```

**Description:**

Checks whether the file denoted by this abstract pathname exists and is a directory.

Package [java.io](#)

Class [File](#)

**Public Method isFile**

```
boolean isFile()
```

**Description:**

Tests whether the file denoted by this abstract pathname exists and is a normal file. A file is normal if it is not a directory.

Package [java.io](#)

Class [File](#)

**Public Method isMounted**

```
static boolean isMounted()
```

**Description:**

Checks if the file system (SD Card) is mounted.

Note that this is not a standard Java method but iLCD panel related.

**Package [java.io](#)****Class [File](#)****Public Method lastModified**

```
Date lastModified()
```

**Description:**

Returns a [Date](#) object containing information about the time that the file denoted by this abstract pathname was last modified. Returns null if the file doesn't exist.

**Package [java.io](#)****Class [File](#)****Public Method length**

```
int length()
```

**Description:**

Returns the length of the file denoted by this abstract pathname in bytes or 0 if the file doesn't exist. For directories the value is undefined.

Note that this value might become negative when the file is bigger than 2147483647 (0x7FFFFFFF) bytes.

(Conversion to float: float f = ret < 0 ? 4294967296f - (float)ret : (float)ret;)

**Package [java.io](#)****Class [File](#)****Public Method list**

```
String[] list()
```

```
String[] list(FileNameFilter filter)
```

**Description:****list() method:**

Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.

If this abstract pathname does not denote a directory, then this method returns null. Otherwise an array of strings is returned, one for each file or directory in the directory. Names denoting the directory itself and the directory's parent directory are not included in the result. Each string is a file name rather than a complete path.

There is no guarantee that the name strings in the resulting array will appear in any specific order; they are not, in particular, guaranteed to appear in alphabetical order.

**list(FilenameFilter filter) method:**

Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter. The behavior of this method is the same as that of the list() method, except that the strings in the returned array must satisfy the filter. If the given filter is null then all names are accepted. Otherwise, a name satisfies the filter if and only if the value true results when the FilenameFilter.accept(java.io.File, java.lang.String) method of the filter is invoked on this abstract pathname and the name of a file or directory in the directory that it denotes.

Package [java.io](#)

Class [File](#)

**Public Method listFiles**

```
File[] listFiles()  
  
File[] listFiles(FileFilter filter)  
  
File[] listFiles(FilenameFilter filter)
```

**Description:****listFiles() method:**

Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

If this abstract pathname does not denote a directory, then this method returns null. Otherwise an array of File objects is returned, one for each file or directory in the directory.

There is no guarantee that the name strings in the resulting array will appear in any specific order; they are not, in particular, guaranteed to appear in alphabetical order.

**listFiles(FileFilter filter) method:**

Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter. The behavior of this method is the same as that of the listFiles() method, except that the pathnames in the returned array must satisfy the filter. If the given filter is null then all pathnames are accepted. Otherwise, a pathname satisfies the filter if and only if the value true results when the FilenameFilter.accept(java.io.File, java.lang.String) method of the filter is invoked on the pathname.

**listFiles(FilenameFilter filter) method:**

Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter. The behavior of this method is the same as that of the listFiles() method, except that the pathnames in the returned array must satisfy the filter. If the given filter is null then all pathnames are accepted. Otherwise, a pathname satisfies the filter if and only if the value true results when the FilenameFilter.accept(java.io.File, java.lang.String) method of the filter is invoked on this abstract pathname and the name of a file or directory in the directory that it denotes.

**Package [java.io](#)**  
**Class [File](#)****Public Method mkdir**

```
boolean mkdir()
```

**Description:**

Creates the directory named by this abstract pathname and returns true iff the directory was created, i.e. when the directory already exists the return value is false.

**Package [java.io](#)**  
**Class [File](#)****Public Method mkdirs**

```
boolean mkdirs()
```

**Description:**

Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories. Note that if this operation fails it may have succeeded in creating some of the necessary parent directories. Returns true iff the directory was created, along with all necessary parent directories.

**Package [java.io](#)**  
**Class [File](#)****Public Method renameTo**

```
boolean renameTo(File dest)
```

**Description:**

Renames the file denoted by this abstract pathname. Renames true if and only if the renaming succeeded.

**Package [java.io](#)**  
**Class [File](#)****Public Method toString**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

Returns the pathname string of this abstract pathname. This is just the string returned by the `getPath()` method.

**Package [java.io](#)****Class [File](#)****Public Method [unmount](#)**

```
static boolean unmount()
```

**Description:**

Closes all files and safely removes the MicroSD card at runtime.

Note that this is not a standard Java method but iLCD panel related.

**See also:**

[Unmount MicroSD Card](#)

**Package [java.io](#)****Class [FileDescriptor](#)**

extends [Object](#)

Instances of the file descriptor class serve as an opaque handle to the underlying machine-specific structure representing an open file, an open socket, or another source or sink of bytes. The main practical use for a file descriptor is to create a [FileInputStream](#) or [FileOutputStream](#) to contain it. Applications should not create their own file descriptors.

**Public Constructors**

- [FileDescriptor](#)

**Public Methods**

- [sync](#)
- [valid](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Public Fields**

- static final [FileDescriptor](#) err
- static final [FileDescriptor](#) in
- static final [FileDescriptor](#) out

Package [java.io](#)

## Class [FileDescriptor](#)

### Public Constructor [FileDescriptor](#)

```
FileDescriptor ()
```

#### Description:

Constructs an (invalid) `FileDescriptor` object.

Package [java.io](#)

## Class [FileDescriptor](#)

### Public Method [sync](#)

```
void sync ()
```

#### Throws:

- [SyncFailedException](#)

#### Description:

Force all system buffers to synchronize with the underlying device. This method returns after all modified data and attributes of this `FileDescriptor` have been written to the relevant device(s). In particular, if this `FileDescriptor` refers to a physical storage medium, such as a file in a file system, `sync` will not return until all in-memory modified copies of buffers associated with this `FileDescriptor` have been written to the physical medium. `sync` is meant to be used by code that requires physical storage (such as a file) to be in a known state. For example, a class that provided a simple transaction facility might use `sync` to ensure that all changes to a file caused by a given transaction were recorded on a storage medium. `sync` only affects buffers downstream of this `FileDescriptor`. If any in-memory buffering is being done by the application (for example, by a `BufferedOutputStream` object), those buffers must be flushed into the `FileDescriptor` (for example, by invoking `OutputStream.flush`) before that data will be affected by `sync`.

Package [java.io](#)

## Class [FileDescriptor](#)

### Public Method [valid](#)

```
boolean valid ()
```

#### Description:

Tests if this file descriptor object is valid, i.e. if the file descriptor object represents a valid, open file, socket, or other active I/O connection.

Package [java.io](#)

## Interface [FileFilter](#)

Defines a filter interface for abstract pathnames.

Instances of this interface can be passed to the `listFiles` method of the [File](#) class.



## Public Methods

- [accept](#)

Package [java.io](#)

## Interface [FileFilter](#)

### Public Method [accept](#)

```
abstract boolean accept(File pathname)
```

#### Description:

Checks if the abstract pathname *pathname* should be included in a pathname list.

Package [java.io](#)

## Class [FileInputStream](#)

extends [InputStream](#) → [Object](#)

A [FileInputStream](#) obtains input bytes from a file in a file system.

### Public Constructors

- [FileInputStream](#)

### Public Methods

- [available](#)
- [close](#)
- [getFD](#)
- [read](#)
- [skip](#)

### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [FileInputStream](#)

### **Public Constructor [FileInputStream](#)**

```
FileInputStream(File file)
```

#### **Throws:**

- [SecurityException](#)
- [FileNotFoundException](#)

```
FileInputStream(FileDescriptor fd)
```

#### **Throws:**

- [SecurityException](#)

```
FileInputStream(String name)
```

#### **Throws:**

- [SecurityException](#)
- [FileNotFoundException](#)

#### **Description:**

#### **Important Note**

Make sure that any stream accessing a file is closed before a new stream for the same file is created.

#### **constructor [FileInputStream](#)(File file):**

Creates a [FileInputStream](#) by opening a connection to an actual file, the file named by the File object file in the file system. A new [FileDescriptor](#) object is created to represent this file connection.

If the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a [FileNotFoundException](#) is thrown.

#### **constructor [FileInputStream](#)(FileDescriptor fd):**

Creates a [FileInputStream](#) by using the file descriptor fdObj, which represents an existing connection to an actual file in the file system.

#### **constructor [FileInputStream](#)(String name):**

Creates a [FileInputStream](#) by opening a connection to an actual file, the file named by the path name name in the file system. A new [FileDescriptor](#) object is created to represent this file connection.

If the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a [FileNotFoundException](#) is thrown.

Package [java.io](#)  
Class [FileInputStream](#)

**Public Method available**

```
int available()
```

**Overrides:**

- [available](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Returns the number of bytes that can be read from this file input stream without blocking.

Package [java.io](#)  
Class [FileInputStream](#)

**Public Method close**

```
void close()
```

**Overrides:**

- [close](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Closes this file input stream and releases any system resources (e.g. file handles) associated with the stream.

Package [java.io](#)  
Class [FileInputStream](#)

**Public Method getFD**

```
final FileDescriptor getFD()
```

**Throws:**

- [IOException](#)

**Description:**

Returns the FileDescriptor object that represents the connection to the actual file in the file system being used by this FileInputStream.

Package [java.io](#)  
Class [FileInputStream](#)

**Public Method read**

```
int read()
```

**Overrides:**

- [read](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

```
int read(byte[] buf)
```

**Overrides:**

- [read](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

**read() method:**

Reads a byte of data from this input stream. This method blocks if no input is yet available.

Returns the next byte of data, or -1 if the end of the file is reached.

**read(byte[] buf) method:**

Reads up to `b.length` bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

Returns the total number of bytes read into the buffer. Note that this number might be smaller than `buf.length` if the end of the file was reached. If there is no more data to read because EOF was reached, -1 is returned.

**read(byte[] buf, int offset, int len) method:**

Reads up to len bytes of data from this input stream into an array of bytes at array position offset. This method blocks until some input is available.

Returns the total number of bytes read into the buffer. Note that this number might be smaller than len if the end of the file was reached. If there is no more data to read because EOF was reached, -1 is returned.

Package [java.io](#)

**Class [FileInputStream](#)****Public Method skip**

```
int skip(int num_bytes)
```

**Overrides:**

- [skip](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Skips over and discards n bytes of data from the input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly 0. The actual number of bytes skipped is returned.

Package [java.io](#)

**Interface [FilenameFilter](#)**

Instances of this interface are used to filter filenames and can be passed to the list method of class [File](#).

**Public Methods**

- [accept](#)

Package [java.io](#)

**Interface [FilenameFilter](#)****Public Method accept**

```
abstract boolean accept(File dir, String name)
```

**Description:**

Checks if the specified file should be included in a file list.

## Package [java.io](#)

### Class **FileNotFoundException**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that an attempt to open the file denoted by a specified pathname has failed.

This exception will be thrown by the `FileInputStream`, `FileOutputStream`, and `RandomAccessFile` constructors when a file with the specified pathname does not exist. It will also be thrown by these constructors if the file does exist but for some reason is inaccessible, for example when an attempt is made to open a read-only file for writing.

#### Public Constructors

- [FileNotFoundException](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [FileNotFoundException](#)

#### **Public Constructor `FileNotFoundException`**

```
FileNotFoundException()
```

```
FileNotFoundException(String message)
```

#### **Description:**

##### **constructor `FileNotFoundException()`:**

Create Exception null as its detailed message.

##### **constructor `FileNotFoundException(String message)`:**

Create Exception with detailed message.

## Package [java.io](#)

### Class **FileOutputStream**

extends [OutputStream](#) → [Object](#)

A file output stream is an output stream for writing data to a File or to a FileDescriptor. Only one FileOutputStream can write to a file at a time.

#### Public Constructors

- [FileOutputStream](#)

#### Public Methods

- [close](#)
- [getFD](#)
- [write](#)

#### Methods inherited from [java.io.OutputStream](#)

- [flush](#)
- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class **[FileOutputStream](#)**

#### **Public Constructor [FileOutputStream](#)**

```
FileOutputStream(File file)
```

#### Throws:

- [SecurityException](#)
- [IOException](#)

```
FileOutputStream(FileDescriptor fd)
```

#### Throws:

- [SecurityException](#)

```
FileOutputStream(String name)
```

**Throws:**

- [SecurityException](#)
- [IOException](#)

```
FileOutputStream(String name, boolean append)
```

**Throws:**

- [SecurityException](#)
- [IOException](#)

**Description:****Important Note**

Make sure that any stream accessing a file is closed before a new stream for the same file is created.

**constructor FileOutputStream(File file):**

Creates a file output stream to write to the file represented by the specified File object. A new FileDescriptor object is created to represent this file connection. Note that any existing file will be overwritten, i.e. deleted and created anew.

**constructor FileOutputStream(FileDescriptor fd):**

Creates an output file stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system. Note that any existing file will be overwritten, i.e. deleted and created anew.

**constructor FileOutputStream(String name):**

Creates an output file stream to write to the file with the specified name. A new FileDescriptor object is created to represent this file connection. Note that any existing file will be overwritten, i.e. deleted and created anew.

**constructor FileOutputStream(String name, boolean append):**

Creates an output file stream to write to the file with the specified name. If the second argument is true, then the data will be written to the end of the file rather than the beginning. A new FileDescriptor object is created to represent this file connection.

Package [java.io](#)

**Class [FileOutputStream](#)****Public Method close**

```
void close ()
```

**Overrides:**

- [close](#) in class [OutputStream](#)



**Throws:**

- [IOException](#)

**Description:**

Closes this file output stream and releases any system resources associated with this stream. This file output stream may no longer be used for writing bytes.

Package [java.io](#)

**Class [FileOutputStream](#)****Public Method [getFD](#)**

```
final FileDescriptor getFD()
```

**Throws:**

- [IOException](#)

**Description:**

Returns the file descriptor associated with this stream.

Package [java.io](#)

**Class [FileOutputStream](#)****Public Method [write](#)**

```
void write(byte[] buf)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)

```
void write(byte[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Note that the OutputStream has to be closed in order to ensure that the data is written to the file.

**write(byte[] buf) method:**

Writes b.length bytes from the specified byte array to this file output stream.

**write(byte[] buf, int offset, int len) method:**

Writes len bytes from the specified byte array starting at offset off to this file output stream.

**write(int b) method:**

Writes the specified byte to this file output stream. Implements the write method of OutputStream.

**Package [java.io](#)****Class FileReader**

extends [InputStreamReader](#) → [Reader](#) → [Object](#)

Convenience class for reading character files.

**Public Constructors**

- [FileReader](#)

**Methods inherited from [java.io.InputStreamReader](#)**

- [getEncoding](#)
- [close](#)
- [ready](#)
- [read](#)

**Methods inherited from [java.io.Reader](#)**

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)

- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [FileReader](#)

### Public Constructor [FileReader](#)

```
FileReader(File file)
```

#### Throws:

- [FileNotFoundException](#)

```
FileReader(FileDescriptor fd)
```

```
FileReader(String name)
```

#### Throws:

- [FileNotFoundException](#)

#### Description:

##### constructor [FileReader](#)(File file)

Creates a new [FileReader](#), specifying the File to read from.

##### constructor [FileReader](#)(FileDescriptor fd)

Creates a new [FileReader](#), specifying the FileDescriptor to read from.

##### constructor [FileReader](#)(String name)

Creates a new [FileReader](#), specifying the file to read from.

Package [java.io](#)

## Class [FileWriter](#)

extends [OutputStreamWriter](#) → [Writer](#) → [Object](#)

Convenience class for writing character files.

### Public Constructors

- [FileWriter](#)

### Methods inherited from [java.io.OutputStreamWriter](#)

- [close](#)
- [flush](#)

- [getEncoding](#)
- [write](#)

#### Methods inherited from [java.io.Writer](#)

- [flush](#)
- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [FileWriter](#)

#### Public Constructor [FileWriter](#)

```
FileWriter(File file)
```

#### Throws:

- [SecurityException](#)
- [IOException](#)

```
FileWriter(FileDescriptor fd)
```

#### Throws:

- [SecurityException](#)

```
FileWriter(String name)
```

#### Throws:

- [IOException](#)

```
FileWriter(String name, boolean append)
```

#### Throws:

- [IOException](#)

#### Description:

##### constructor [FileWriter](#)(File file):

Creates a [FileWriter](#) object for a specified File object.

**constructor `FileWriter(FileDescriptor fd)`:**

Creates a `FileWriter` object for a specified `FileDescriptor`.

**constructor `FileWriter(String name)`:**

Creates a `FileWriter` object for a file specified by its name.

**constructor `FileWriter(String name, boolean append)`:**

Creates a `FileWriter` object for a file specified by its name. Data written to the file will be appended if *append* is true.

**Package [java.io](#)****Class `FilterInputStream`**

extends [InputStream](#) → [Object](#)

A `FilterInputStream` contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality. The class `FilterInputStream` itself simply overrides all methods of `InputStream` with versions that pass all requests to the contained input stream. Subclasses of `FilterInputStream` may further override some of these methods and may also provide additional methods and fields.

Note that the constructor of this class is protected, i.e. in order to use it a subclass has to be derived.

**Public Methods**

- [available](#)
- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [reset](#)
- [skip](#)

**Methods inherited from [java.io.InputStream](#)**

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)  
Class [FilterInputStream](#)

**Public Method available**

```
int available()
```

**Overrides:**

- [available](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Returns the number of bytes that can be read from this input stream without blocking.

Package [java.io](#)  
Class [FilterInputStream](#)

**Public Method close**

```
void close()
```

**Overrides:**

- [close](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Closes this input stream and releases any system resources associated with the stream.

Package [java.io](#)  
Class [FilterInputStream](#)

**Public Method mark**

```
void mark(int readlimit)
```

**Overrides:**

- [mark](#) in class [InputStream](#)

**Description:**

Marks the current position in this input stream. A subsequent call to the reset method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

The readlimit argument tells this input stream to allow that many bytes to be read before the mark position gets invalidated.

Stream marks are intended to be used in situations where reading ahead is necessary.

Package [java.io](#)

## Class [FilterInputStream](#)

### Public Method [markSupported](#)

```
boolean markSupported()
```

#### Overrides:

- [markSupported](#) in class [InputStream](#)

#### Description:

Checks if this input stream supports the mark and reset methods.

Package [java.io](#)

## Class [FilterInputStream](#)

### Public Method [read](#)

```
int read()
```

#### Overrides:

- [read](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

```
int read(byte[] buf)
```

#### Overrides:

- [read](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

**Description:****read() method:**

Reads a byte of data from this input stream. This method blocks if no input is yet available.

**read(byte[] buf) method:**

Reads up to `b.length` bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

Returns the total number of bytes read into the buffer. Note that this number might be smaller than `buf.length` if the end of the stream was reached. If there is no more data to read because the end of the stream was reached, `-1` is returned.

**read(byte[] buf, int offset, int len) method:**

Reads up to `len` bytes of data from this input stream into an array of bytes at array position `offset`. This method blocks until some input is available.

Returns the total number of bytes read into the buffer. Note that this number might be smaller than `len` if the end of the stream was reached. If there is no more data to read because the end of the stream was reached, `-1` is returned.

Package [java.io](#)

**Class [FilterInputStream](#)****Public Method reset**

```
void reset ()
```

**Overrides:**

- [reset](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Repositions this stream to the position marked with the `mark` method.

Stream marks are intended to be used in situations where you need to read ahead to check what data comes next in the stream.

Package [java.io](#)

**Class [FilterInputStream](#)****Public Method skip**

```
int skip(int num_bytes)
```



**Overrides:**

- [skip](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Skips over n bytes of data. The actual number of bytes skipped is returned.

**Package [java.io](#)****Class [FilterOutputStream](#)**

extends [OutputStream](#) → [Object](#)

Superclass for classes that filter output streams. Such a class is used to transform data before passing it to the underlying output stream.

**Public Constructors**

- [FilterOutputStream](#)

**Public Methods**

- [close](#)
- [flush](#)
- [write](#)

**Methods inherited from [java.io.OutputStream](#)**

- [flush](#)
- [close](#)
- [write](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [FilterOutputStream](#)****Public Constructor [FilterOutputStream](#)**

```
FilterOutputStream(OutputStream out)
```

**Description:**

Creates an output stream filter for the underlying output stream *out*.

Package [java.io](#)

## Class [FilterOutputStream](#)

### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [OutputStream](#)

#### Throws:

- [IOException](#)

#### Description:

Closes this output stream and releases all system resources associated with it. Data will be flushed before closing the underlying output stream.

Package [java.io](#)

## Class [FilterOutputStream](#)

### Public Method flush

```
void flush ()
```

#### Overrides:

- [flush](#) in class [OutputStream](#)

#### Throws:

- [IOException](#)

#### Description:

Flushes any buffered data to the underlying output stream by calling its flush method.

Package [java.io](#)

## Class [FilterOutputStream](#)

### Public Method write

```
void write (byte[] buf)
```

#### Overrides:

- [write](#) in class [OutputStream](#)

#### Throws:

- [IOException](#)

```
void write(byte[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)

**Description:****write(byte[] buf) method:**

Writes `b.length` bytes from the specified byte array to this file output stream.

**write(byte[] buf, int offset, int len) method:**

Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.

**write(int b) method:**

Writes the specified byte to this output stream. Implements the `write` method of `OutputStream`.

**Package [java.io](#)****Abstract Class [FilterReader](#)**

extends [Reader](#) → [Object](#)

Abstract class for filtered reading of character streams.

**Public Methods**

- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [ready](#)
- [reset](#)
- [skip](#)

**Methods inherited from [java.io.Reader](#)**

- [read](#)

- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [FilterReader](#)

#### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [Reader](#)

#### Throws:

- [IOException](#)

#### Description:

Close the stream.

#### Package [java.io](#)

### Class [FilterReader](#)

#### Public Method mark

```
void mark (int readlimit)
```

#### Overrides:

- [mark](#) in class [Reader](#)

#### Throws:

- [IOException](#)

#### Description:

Marks the present position in the stream.

Package [java.io](#)

## Class [FilterReader](#)

### Public Method markSupported

```
boolean markSupported()
```

#### Overrides:

- [markSupported](#) in class [Reader](#)

#### Description:

Checks if this stream supports mark method calls.

Package [java.io](#)

## Class [FilterReader](#)

### Public Method read

```
int read()
```

#### Overrides:

- [read](#) in class [Reader](#)

#### Throws:

- [IOException](#)

```
int read(char[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [Reader](#)

#### Throws:

- [IOException](#)

#### Description:

##### **read() method:**

Reads a single character.

##### **read(char[] buf, int offset, int len) method:**

Reads *len* characters from the char array *buf*, starting at offset *offset*.

Package [java.io](#)  
Class [FilterReader](#)

**Public Method [ready](#)**

```
boolean ready()
```

**Overrides:**

- [ready](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Checks if this stream is ready to be read.

Package [java.io](#)  
Class [FilterReader](#)

**Public Method [reset](#)**

```
void reset()
```

**Overrides:**

- [reset](#) in class [Reader](#)

**Throws:**

- [IOException](#)

Package [java.io](#)  
Class [FilterReader](#)

**Public Method [skip](#)**

```
int skip(int num_chars)
```

**Overrides:**

- [skip](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

Skip *n* characters in this stream.

## Package [java.io](#)

### Abstract Class [FilterWriter](#)

extends [Writer](#) → [Object](#)

Abstract class for writing filtered character streams.

#### Public Methods

- [close](#)
- [flush](#)
- [write](#)

#### Methods inherited from [java.io.Writer](#)

- [flush](#)
- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [FilterWriter](#)

#### Public Method [close](#)

```
void close ()
```

#### Overrides:

- [close](#) in class [Writer](#)

#### Throws:

- [IOException](#)

#### Description:

Close the stream.

## Package [java.io](#)

### Class [FilterWriter](#)

#### Public Method [flush](#)

```
void flush ()
```

**Overrides:**

- [flush](#) in class [Writer](#)

**Throws:**

- [IOException](#)

**Description:**

Flush the stream.

Package [java.io](#)

**Class [FilterWriter](#)****Public Method write**

```
void write(char[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Throws:**

- [IOException](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Throws:**

- [IOException](#)

```
void write(String str, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Throws:**

- [IOException](#)

**Description:**

**write(char[] buf, int offset, int len) method:**

Writes *len* characters from the specified array starting at offset *off*.



**write(int b) method:**

Writes a single character *b*.

**write(String str, int offset, int len) method:**

Writes a part of a string, starting at *offset*, writing *len* characters.

**Package [java.io](#)****Abstract Class [InputStream](#)**

extends [Object](#)

Superclass for classes implementing an input stream of bytes, i.e. for getting data byte by byte from a data source without any translation.

**Public Constructors**

- [InputStream](#)

**Public Methods**

- [available](#)
- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [reset](#)
- [skip](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [InputStream](#)****Public Constructor [InputStream](#)**

```
InputStream ( )
```

**Description:**

Creates an input stream object.

Package [java.io](#)

## Class [InputStream](#)

### Public Method available

```
int available()
```

#### Throws:

- [IOException](#)

#### Description:

Returns the number of bytes that can be read or skipped without blocking. Actually this method always returns 0 and has to be overwritten by subclasses.

Package [java.io](#)

## Class [InputStream](#)

### Public Method close

```
void close()
```

#### Throws:

- [IOException](#)

#### Description:

Closes this input stream and releases any used system resources.

The InputStream method actually does nothing and should be overwritten.

Package [java.io](#)

## Class [InputStream](#)

### Public Method mark

```
void mark(int readlimit)
```

#### Description:

This method does nothing and has to be overwritten by a subclass.

The overwritten method should mark the current position in this input stream. A subsequent call to the reset method puts the stream back to the marked position.

The readlimit argument specifies how many bytes can be read before the mark position gets invalidated.

Package [java.io](#)  
Class [InputStream](#)

**Public Method `markSupported`**

```
boolean markSupported()
```

**Description:**

This method always returns false and should be overwritten by subclasses.

Checks if this input stream supports the mark and reset methods.

Package [java.io](#)  
Class [InputStream](#)

**Public Method `read`**

```
abstract int read()
```

**Throws:**

- [IOException](#)

```
int read(byte[] buf)
```

**Throws:**

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)

**Description:**

**`read()` method:**

This method is abstract and has to be implemented in a subclass.

Reads the next byte from this input stream. The value byte is returned as an int in the range of 0 and 255. If no byte is available because the end of the stream is reached, the value -1 is returned. This method blocks until input data is available.

**`read(byte[] buf)` method:**

Tries to read *buf.length* bytes into the buffer array *buf*. The actual number of bytes read is returned. Note that it might be smaller than *buf.length*, e.g. when the end of the stream has been reached. If the end of the stream has been reached and there is no more data to be read, -1 is returned. This method blocks until data is available.

**read(byte[] buf, int offset, int len) method:**

Reads up to *len* bytes from this input stream into the buffer array *buf*, starting at the buffer offset *offset*. The actual number of bytes read is returned. This number might be smaller than *len*. If the end of the stream has been reached and there is no more data to be read, -1 is returned.

**Package [java.io](#)****Class [InputStream](#)****Public Method [reset](#)**

```
void reset ()
```

**Throws:**

- [IOException](#)

**Description:**

This method does nothing and should be implemented by subclasses.

Repositions this stream to the position marked by calling the mark method.

**Package [java.io](#)****Class [InputStream](#)****Public Method [skip](#)**

```
int skip(int num_bytes)
```

**Throws:**

- [IOException](#)

**Description:**

Skips over and discards *num\_bytes* bytes from this input stream. The number of actually skipped bytes may be smaller than *num\_bytes*, possibly 0 (e.g. when reaching end of file). If *num\_bytes* is negative than no bytes are skipped.

The skip method of *InputStream* creates a byte array and then repeatedly reads into it until *num\_bytes* bytes have been read or the end of the stream has been reached. Subclasses should provide a more efficient implementation of this method.

**Package [java.io](#)****Class [InputStreamReader](#)**

extends [Reader](#) → [Object](#)

An [InputStreamReader](#) is a bridge from byte stream to character streams: It reads bytes and translates them into characters according to a specified character encoding. The encoding that is used may be specified by name, or the default encoding may be accepted.

Each invocation of one of an *InputStreamReader*'s *read* methods may cause one or more bytes to be read from the underlying byte input stream. To enable the efficient conversion of a bytes to

characters, more bytes may be read ahead from the underlying stream than are necessary to satisfy the current read operation.

Only utf8 is supported by an iLCD panel.

### Public Constructors

- [InputStreamReader](#)

### Public Methods

- [close](#)
- [getEncoding](#)
- [read](#)
- [ready](#)

### Methods inherited from [java.io.Reader](#)

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.io](#)

## Class [InputStreamReader](#)

### Public Constructor [InputStreamReader](#)

```
InputStreamReader(InputStream in)
```

```
InputStreamReader(InputStream in, String encoding_name)
```

### Throws:

- [UnsupportedEncodingException](#)

### Description:

#### constructor [InputStreamReader\(InputStream in\)](#):

This method constructs an *InputStreamReader* that uses the default (utf8) character encoding.

**constructor `InputStreamReader(InputStream in, String encoding_name)`:**

This method constructs an *InputStreamReader* that uses the specified encoding name.

Note that only "utf8" is supported as `encoding_name`. So this constructor is practically the same as the above.

Package [java.io](#)

**Class [InputStreamReader](#)****Public Method close**

```
void close ()
```

**Overrides:**

- [close](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method closes this input stream reader.

**Example**

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
InputStreamReader inr = new InputStreamReader(in);

inr.close();
```

The example above close this stream.

Package [java.io](#)

**Class [InputStreamReader](#)****Public Method getEncoding**

```
String getEncoding()
```

**Description:**

Returns the canonical name of the character encoding being used by this stream. If this *InputStreamReader* was created with the *InputStreamReader(InputStream, String)* constructor then the returned encoding name, being canonical, may differ from the encoding name passed to the constructor. May return null if the stream has been closed.

Note that this method will always return "UTF8".

## Example

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
InputStreamReader inr = new InputStreamReader(in);

Console.println("get encoding: " + inr.getEncoding());
```

The example above prints the encoding to the console.

Package [java.io](#)

## Class [InputStreamReader](#)

### Public Method read

```
int read(char[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [Reader](#)

#### Throws:

- [IOException](#)

#### Description:

This method read characters into a portion of an array.

## Example

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
InputStreamReader inr = new InputStreamReader(in);

char[] charBuff = new char[5];
inr.read(charBuff, 0, 5);
```

The example above reads five bytes into a character array.

Package [java.io](#)

## Class [InputStreamReader](#)

### Public Method ready

```
boolean ready()
```

#### Overrides:

- [ready](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method tell whether this stream is ready to be read. An *InputStreamReader* is ready if its input buffer is not empty, or if bytes are available to be read from the underlying byte stream.

**Example**

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
InputStreamReader inr = new InputStreamReader(in);

char[] charBuff = new char[5];
if (inr.ready())
    inr.read(charBuff, 0, 5);
```

The example above reads five bytes into a character array if the stream is ready.

**Package [java.io](#)****Class [InterruptedIOException](#)**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that an I/O operation has been interrupted. An [InterruptedIOException](#) is thrown to indicate that an input or output transfer has been terminated because the thread performing it was terminated.

**Public Constructors**

- [InterruptedIOException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Public Fields**

- `int bytesTransferred`



**Package [java.io](#)****Class [InterruptedIOException](#)****Public Constructor [InterruptedIOException](#)**

```
InterruptedIOException()
```

```
InterruptedIOException(String message)
```

**Description:****constructor [InterruptedIOException](#)():**

Create Exception null as its detailed message.

**constructor [InterruptedIOException](#)(String message):**

Create Exception with detailed message.

**Package [java.io](#)****Class [InvalidClassException](#)**

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

implements [Serializable](#)

Thrown when the Serialization runtime detects one of the following problems with a Class:

- The serial version of the class does not match that of the class descriptor read from the stream
- The class contains unknown datatypes
- The class does not have an accessible no-arg constructor

**Public Constructors**

- [InvalidClassException](#)

**Public Methods**

- [getMessage](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)

- [notifyAll](#)

## Public Fields

- String classname

Package [java.io](#)

## Class [InvalidClassException](#)

### Public Constructor InvalidClassException

```
InvalidClassException(String message)
```

```
InvalidClassException(String classname, String message)
```

### Description:

#### constructor InvalidClassException():

Create Exception with detailed message.

#### constructor InvalidClassException(String classname, String message):

Create Exception with detailed message and the name of the invalid class.

Package [java.io](#)

## Class [InvalidClassException](#)

### Public Method getMessage

```
String getMessage()
```

### Overrides:

- [getMessage](#) in class [Throwable](#)

### Description:

Return detailed error message and class name.

Package [java.io](#)

## Class InvalidObjectException

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

implements [Serializable](#)

Indicates that one or more deserialized objects failed validation tests. The constructor argument should provide the reason for the failure.

### Public Constructors

- [InvalidObjectException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [InvalidObjectException](#)****Public Constructor [InvalidObjectException](#)**

```
InvalidObjectException(String message)
```

**Description:**

Create Exception with detailed message giving the reason for the validation tests fail.

**Package [java.io](#)****Class [IOException](#)**

extends [Exception](#) → [Throwable](#) → [Object](#)

Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.

**Public Constructors**

- [IOException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)

- [notifyAll](#)

Package [java.io](#)

## Class [IOException](#)

### Public Constructor IOException

```
IOException()
```

```
IOException(String message)
```

#### Description:

##### constructor `IOException()`:

Create Exception null as its detailed message.

##### constructor `IOException(String message)`:

Create Exception with detailed message.

Package [java.io](#)

## Class `LineNumberInputStream`

extends [FilterInputStream](#) → [InputStream](#) → [Object](#)

This class is an input stream filter which provides the added functionality of keeping track of the current line number.

A line sequence of bytes ending with a carriage return character ( `'\r'` ), a newline character ( `'\n'` ), or a carriage return character followed immediately by a line feed character. In all three cases, the line terminating character(s) are returned as a single newline character.

### Public Constructors

- [LineNumberInputStream](#)

### Public Methods

- [available](#)
- [getLineNumber](#)
- [mark](#)
- [read](#)
- [reset](#)
- [setLineNumber](#)
- [skip](#)

### Methods inherited from [java.io.FilterInputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)

- [close](#)

#### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class [LineNumberInputStream](#)

#### Public Constructor [LineNumberInputStream](#)

```
LineNumberInputStream(InputStream in)
```

#### Description:

This method constructs a newline number input stream that reads its input from the specified input stream.

#### Package [java.io](#)

### Class [LineNumberInputStream](#)

#### Public Method available

```
int available()
```

#### Overrides:

- [available](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

#### Description:

This method returns the number of bytes that can be read from this input stream without blocking.

## Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
...
int read = lis.available();
```

The example above returns the number of bytes that can be read from the input stream without blocking.

Package [java.io](#)

## Class [LineNumberInputStream](#)

### Public Method [getLineNumber](#)

```
int getLineNumber()
```

#### Description:

This method returns the current line number.

## Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
...
lnis.getLineNumber();
```

The example above returns the current line number.

Package [java.io](#)

## Class [LineNumberInputStream](#)

### Public Method [mark](#)

```
void mark(int readlimit)
```

#### Overrides:

- [mark](#) in class [FilterInputStream](#)

#### Description:

This method marks the present position in the stream. Subsequent calls to the *reset* method will attempt to reposition the stream to this point, and will reset the line number appropriately.

## Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
```

```
...
lnis.mark(10);
```

The example above marks the current position in the stream.

Package [java.io](#)

## Class [LineNumberInputStream](#)

### Public Method [read](#)

```
int read()
```

#### Overrides:

- [read](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

#### Description:

##### [read\(\)](#) method:

The *read* method reads the next byte of data from this input stream. The value byte is returned as an *int* in the range 0 to 255. If no byte is available, the end of the stream is detected, or an exception is thrown.

##### [read\(byte\[\] buf, int offset, int len\)](#) method:

This method reads up to *len* bytes of data from this input stream into an array of bytes. This method blocks until some input is available.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
...
int read = lnis.read();
```

The example above reads the current byte from the stream.

Package [java.io](#)

## Class [LineNumberInputStream](#)

### **Public Method `reset`**

```
void reset()
```

#### **Overrides:**

- [reset](#) in class [FilterInputStream](#)

#### **Throws:**

- [IOException](#)

#### **Description:**

This method resets the stream to the most recent mark.

#### **Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
...
lnis.reset();
```

The example above resets the stream to the most recent mark.

Package [java.io](#)

## Class [LineNumberInputStream](#)

### **Public Method `setLineNumber`**

```
void setLineNumber(int line_number)
```

#### **Description:**

This method sets the current line number.

#### **Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
...
lnis.setLineNumber();
```

The example above sets the current line number.



**Package [java.io](#)****Class [LineNumberInputStream](#)****Public Method [skip](#)**

```
int skip(int num_bytes)
```

**Overrides:**

- [skip](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:**

This method skips the specified characters.

**Example**

```
File file = new File("testFile");
InputStream is = FileInputStream(file);
LineNumberInputStream lnis = new LineNumberInputStream(is);
...
lnis.skip(3);
```

The example above skips the specified number of lines.

**Package [java.io](#)****Class [LineNumberReader](#)**

extends [BufferedReader](#) → [Reader](#) → [Object](#)

A buffered character input stream that keeps track of line numbers. A line is considered to be terminated by any of one of a linefeed (`'\n'`), a carriage return (`'\r'`), or carriage return followed immediately by a linefeed (`'\r\n'`).

**Public Constructors**

- [LineNumberReader](#)

**Public Methods**

- [getLineNumber](#)
- [mark](#)
- [read](#)
- [readLine](#)
- [reset](#)
- [setLineNumber](#)
- [skip](#)

**Methods inherited from [java.io.BufferedReader](#)**

- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [read](#)
- [readLine](#)
- [skip](#)

**Methods inherited from [java.io.Reader](#)**

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [LineNumberReader](#)****Public Constructor LineNumberReader**

```
LineNumberReader(Reader in)
```

```
LineNumberReader(Reader in, int size)
```

**Description:****constructor [LineNumberReader\(Reader in\)](#)**

Creates a new [LineNumberReader](#), using the default input buffer size.

**constructor [LineNumberReader\(Reader in, int size\)](#)**

Creates a new [FileReader](#), reading characters into a buffer of the given size.

**Package [java.io](#)****Class [LineNumberReader](#)****Public Method [getLineNumber](#)**

```
int getLineNumber()
```

**Description:**

This method returns the current line number.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
Logger.log("current line number: " + lnr.getLineNumber());
```

The example above reads the current line number and writes the result to the logger.

Package [java.io](#)

**Class [LineNumberReader](#)****Public Method mark**

```
void mark(int readlimit)
```

**Overrides:**

- [mark](#) in class [BufferedReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method marks the present position in the stream. Subsequent calls to the *reset* method will attempt to reposition the stream to this point, and will reset the line number appropriately.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
lnr.mark(10);
```

The example above marks the current position in the stream.

Package [java.io](#)

**Class [LineNumberReader](#)****Public Method read**

```
int read()
```

**Overrides:**

- [read](#) in class [BufferedReader](#)

**Throws:**

- [IOException](#)

```
int read(char[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [BufferedReader](#)

**Throws:**

- [IOException](#)

**Description:****read() method:**

The *read* method reads a single character. Line terminators are compressed into single newline ('\n') characters.

**read(byte[] buf, int offset, int len) method:**

This method reads characters into a portion of an array.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
int read = lnr.read();
```

The example above reads the current character from the stream.

Package [java.io](#)

**Class [LineNumberReader](#)****Public Method readLine**

```
String readLine()
```

**Overrides:**

- [readLine](#) in class [BufferedReader](#)

**Throws:**

- [IOException](#)

**Description:**

Read a line of text. A line is considered to be terminated by any one of a line feed (`'\n'`), a carriage return (`'\r'`), or a carriage return followed immediately by a line feed.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
String readLine = lnr.readLine();
```

The example above reads a line from the stream till a line terminator is reached.

Package [java.io](#)

Class [LineNumberReader](#)

**Public Method `reset`**

```
void reset()
```

**Overrides:**

- [reset](#) in class [BufferedReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method resets the stream to the most recent mark.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
lnr.reset();
```

The example above resets the current position in the stream to the most recent mark.

Package [java.io](#)

Class [LineNumberReader](#)

**Public Method `setLineNumber`**

```
void setLineNumber(int line_number)
```

**Description:**

This method sets the current line number.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
lnr.setLineNumber();
```

The example above sets the current line number.

Package [java.io](#)

**Class [LineNumberReader](#)****Public Method [skip](#)**

```
int skip(int num_chars)
```

**Overrides:**

- [skip](#) in class [BufferedReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method skips the specified characters.

**Example**

```
File file = new File("testFile");
Reader read = new FileReader(file);
LineNumberReader lnr = new LineNumberReader(read);
...
lnr.skip(3);
```

The example above skips the specified number of lines.

Package [java.io](#)

**Class [NotActiveException](#)**

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)  
implements [Serializable](#)

Thrown when serialization or deserialization is not active.

## Public Constructors

- [NotActiveException](#)

## Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [NotActiveException](#)

#### Public Constructor NotActiveException

```
NotActiveException()
```

```
NotActiveException(String message)
```

#### Description:

##### constructor `NotActiveException()`:

Create Exception null as its detailed message.

##### constructor `NotActiveException(String message)`:

Create Exception with detailed message.

## Package [java.io](#)

### Class `NotSerializableException`

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)  
implements [Serializable](#)

Thrown when an instance is required to have a `Serializable` interface. The serialization runtime or the class of the instance can throw this exception.

## Public Constructors

- [NotSerializableException](#)

## Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [NotSerializableException](#)

#### **Public Constructor [NotSerializableException](#)**

```
NotSerializableException ()
```

```
NotSerializableException (String message)
```

#### **Description:**

##### **constructor [NotSerializableException](#)():**

Create Exception null as its detailed message.

##### **constructor [NotSerializableException](#)(String message):**

Create Exception with detailed message. The message should give also the class name.

## Package [java.io](#)

### Interface [ObjectInput](#)

implements [DataInput](#)

Interface extending the [DataInput](#) interface to include reading objects. [DataInput](#) defines methods for the input of primitive types. [ObjectInput](#) extends that interface to include also methods for reading objects, arrays and strings.

#### **Public Methods**

- [available](#)
- [close](#)
- [read](#)
- [readObject](#)
- [skip](#)



Package [java.io](#)  
Interface [ObjectInput](#)

**Public Method available**

```
abstract int available()
```

**Throws:**

- [IOException](#)

**Description:**

Returns the number of bytes that can be read without blocking.

Package [java.io](#)  
Interface [ObjectInput](#)

**Public Method close**

```
abstract void close()
```

**Throws:**

- [IOException](#)

**Description:**

Closes the input stream. Must be called to release any resources associated with the stream.

Package [java.io](#)  
Interface [ObjectInput](#)

**Public Method read**

```
abstract int read()
```

**Throws:**

- [IOException](#)

```
abstract int read(byte[] buf)
```

**Throws:**

- [IOException](#)

```
abstract int read(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)

**Description:****read() method:**

Reads one byte of data. This method will block until data is available. If the end of the stream is reached and there is no more data to read, -1 is returned.

**read(byte[] buf) method:**

Reads *buf.length* bytes into the byte array *buf*. This method will block until data is available. If the end of the stream is reached and there is no more data to read, -1 is returned.

**read(byte[] buf, int offset, int len) method:**

Reads *len* bytes into the byte array *buf*, starting at offset *offset*. If the end of the stream is reached and there is no more data to read, -1 is returned.

Package [java.io](#)

**Interface [ObjectInput](#)****Public Method readObject**

```
abstract Object readObject ()
```

**Throws:**

- [ClassNotFoundException](#)
- [IOException](#)

**Description:**

Read and return an object. The class that implements this interface defines where the object is read from.

Package [java.io](#)

**Interface [ObjectInput](#)****Public Method skip**

```
abstract int skip(int num_bytes)
```

**Throws:**

- [IOException](#)

**Description:**

Skips n bytes of input. Returns the actual number of bytes skipped.

Package [java.io](#)

**Interface [ObjectOutput](#)**

implements [DataOutput](#)

Interface extending the `DataOutput` interface to include writing of objects. `DataOutput` includes methods for the output of primitive types, `ObjectOutput` extends that interface to include objects, arrays and strings.

### Public Methods

- [close](#)
- [flush](#)
- [write](#)
- [writeObject](#)

Package [java.io](#)

## Interface [ObjectOutput](#)

### Public Method close

```
abstract void close()
```

#### Throws:

- [IOException](#)

#### Description:

Closes this stream. This method must be called to release any resources associated with the stream.

Package [java.io](#)

## Interface [ObjectOutput](#)

### Public Method flush

```
abstract void flush()
```

#### Throws:

- [IOException](#)

#### Description:

Flushes this stream writing any buffered output bytes.

Package [java.io](#)

## Interface [ObjectOutput](#)

### Public Method write

```
abstract void write(byte[] buf)
```

#### Throws:

- [IOException](#)

```
abstract void write(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)

```
abstract void write(int b)
```

**Throws:**

- [IOException](#)

**Description:****write(byte[] buf) method:**

Writes an array of bytes. This method will block until the bytes are actually written.

**write(byte[] buf, int offset, int len) method:**

Writes *len* bytes from the byte array *buf*, starting at *offset*. This method will block until the bytes are actually written.

**write(int b) method:**

Writes one byte. This method will block until the byte is actually written.

Package [java.io](#)

**Interface [ObjectOutput](#)****Public Method [writeObject](#)**

```
abstract void writeObject(Object obj)
```

**Throws:**

- [IOException](#)

**Description:**

Write an object to the underlying output stream. The class implementing this interface defines how the object is written.

Package [java.io](#)

**Abstract Class [ObjectStreamException](#)**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

implements [Serializable](#)

Superclass of all exceptions specific to Object Stream classes.

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)

- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.io](#)

### Class `OptionalDataException`

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)  
implements [Serializable](#)

Unexpected data appeared in an `ObjectInputStream` trying to read an `Object`. Occurs when the stream contains primitive data instead of the object that is expected by `readObject`. The EOF flag in the exception is true indicating that no more primitive data is available. The count field contains the number of bytes available to read.

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- boolean eof
- int length

#### Package [java.io](#)

### Abstract Class `OutputStream`

extends [Object](#)

Abstract class to be used as superclass for classes implementing output streams of bytes. An output stream accepts output bytes and sends to an underlying data sink.

## Public Constructors

- [OutputStream](#)

## Public Methods

- [close](#)
- [flush](#)
- [write](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [OutputStream](#)

#### Public Constructor OutputStream

```
OutputStream()
```

#### Description:

Creates an output stream object.

## Package [java.io](#)

### Class [OutputStream](#)

#### Public Method close

```
void close()
```

#### Throws:

- [IOException](#)

#### Description:

The close method of *OutputStream* does nothing, i.e. it has to be implemented in a derived class.

Closes this output stream and releases all corresponding system resources. A closed output stream cannot be opened again.

## Package [java.io](#)

### Class [OutputStream](#)

#### Public Method flush

```
void flush()
```

**Throws:**

- [IOException](#)

**Description:**

The flush method of *OutputStream* does nothing, i.e. it has to be implemented in a derived class.

Flushes this output stream and forces any buffered data to be written to its destination.

Package [java.io](#)

**Class [OutputStream](#)****Public Method [write](#)**

```
void write(byte[] buf)
```

**Throws:**

- [IOException](#)

```
void write(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)

```
abstract void write(int b)
```

**Throws:**

- [IOException](#)

**Description:****[write\(byte\[\] buf\)](#) method:**

Writes *buf.length* bytes from the specified buffer array to this output stream.

**[write\(byte\[\] buf, int offset, int len\)](#) method:**

Writes *len* bytes from the specified buffer array to this output stream, starting at *offset*.

**[write\(int b\)](#) method:**

Writes one byte to this output stream. The byte written consists of the eight lower bits of *b*. The 24 higher bits are ignored.

At least this method has to be implemented by a derived class.

Package [java.io](#)

**Class [OutputStreamWriter](#)**

extends [Writer](#) → [Object](#)

An `OutputStreamWriter` translates character streams to byte streams according to a specified encoding.

Only utf8 is supported by an iLCD panel.

### Public Constructors

- [OutputStreamWriter](#)

### Public Methods

- [close](#)
- [flush](#)
- [getEncoding](#)
- [write](#)

### Methods inherited from [java.io.Writer](#)

- [flush](#)
- [close](#)
- [write](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [OutputStreamWriter](#)

### Public Constructor OutputStreamWriter

```
OutputStreamWriter(OutputStream out)
```

```
OutputStreamWriter(OutputStream out, String encoding_scheme)
```

### Throws:

- [UnsupportedEncodingException](#)

### Description:

#### constructor `OutputStreamWriter(OutputStream out)`:

This method constructs an `OutputStreamWriter` that uses the default (utf8) character encoding.

#### constructor `OutputStreamWriter(OutputStream out, String encoding_scheme)`:

This method constructs an `OutputStreamWriter` that uses the specified encoding name.



Note that only "utf8" is supported as encoding\_scheme. So this constructor is practically the same as the above.

Package [java.io](#)

## Class [OutputStreamWriter](#)

### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [Writer](#)

#### Throws:

- [IOException](#)

#### Description:

Closes this output stream.

Package [java.io](#)

## Class [OutputStreamWriter](#)

### Public Method flush

```
void flush ()
```

#### Overrides:

- [flush](#) in class [Writer](#)

#### Throws:

- [IOException](#)

#### Description:

Flushes all buffered bytes to their output destination.

Package [java.io](#)

## Class [OutputStreamWriter](#)

### Public Method getEncoding

```
String getEncoding ()
```

#### Description:

Returns the canonical name of the character encoding being used by this stream. If this *OutputStreamWriter* was created with the *OutputStreamWriter(OutputStream, String)* constructor then the returned encoding name may differ from the encoding name passed to the constructor. May also return *null* if the stream was closed.

Note that this method will always return "UTF8".

Package [java.io](#)

## Class [OutputStreamWriter](#)

### Public Method write

```
void write(char[] buf, int offset, int len)
```

#### Overrides:

- [write](#) in class [Writer](#)

#### Throws:

- [IOException](#)

#### Description:

Writes *len* bytes from a character array, starting at array offset *offset*.

Package [java.io](#)

## Class [PipedInputStream](#)

extends [InputStream](#) → [Object](#)

A pipe input stream should be connected to a pipe output stream. If a pipe input stream is connected to a pipe output stream, the data bytes are written to the pipe output stream. The piped input stream contains a buffer that decoupling read from write operations.

### Public Constructors

- [PipedInputStream](#)

### Public Methods

- [available](#)
- [close](#)
- [connect](#)
- [read](#)

### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)

- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [PipedInputStream](#)

### Public Constructor PipedInputStream

```
PipedInputStream()
```

```
PipedInputStream(PipedOutputStream source)
```

#### Throws:

- [IOException](#)

#### Description:

##### constructor PipedInputStream():

This method constructs a *PipedInputStream* so that it is not yet connected. It must be connected to a *PipedOutputStream* before being used.

##### constructor PipedInputStream(PipedOutputStream source):

This method constructs a *PipedInputStream* so that is connected to the piped output stream *source*. Data bytes written to *source* will then be available as input from this stream.

Package [java.io](#)

## Class [PipedInputStream](#)

### Public Method available

```
int available()
```

#### Overrides:

- [available](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

#### Description:

It returns the number of bytes that can be read from this input stream without blocking.

#### Example

```
PipedOutputStream pos = new PipedOutputStream();  
PipedInputStream pis = new PipedInputStream(pos);
```

```
byte[] b = {10,20};
pos.write(b, 0, 2);
if (pis.available() > 0)
    Console.println("read data: " + pis.read());
```

The *available* method checks if data can be read.

Package [java.io](#)

## Class [PipedInputStream](#)

### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

#### Description:

This method closes this piped input stream and releases any system resources associated with this stream.

#### Example

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream(pos);
pos.close();
pis.close();
```

The *close* method closes the piped input and output stream.

Package [java.io](#)

## Class [PipedInputStream](#)

### Public Method connect

```
void connect (PipedOutputStream source)
```

#### Throws:

- [IOException](#)

#### Description:

This method connects this piped input stream to the piped output stream *source*. An *IOException* will be thrown, if this object is already connected to some other piped output stream.

## Example

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream();

pis.connect(pos);
```

The `connect` method connects the piped input stream with the piped output stream.

Package [java.io](#)

## Class [PipedInputStream](#)

### Public Method `read`

```
int read()
```

#### Overrides:

- [read](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [InputStream](#)

#### Throws:

- [IOException](#)
- [InterruptedException](#)

#### Description:

##### `read()` method:

This method reads the next byte of data from this piped input stream. The value byte is returned as an int in the range 0 to 255. -1 is returned, if no byte is available because the end of the stream has been reached. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown. If a thread was providing data bytes to the connected piped output stream, but the thread is no longer alive, then an *IOException* is thrown.

##### `read(byte[] buf, int offset, int len)` method:

This method reads up to *len* bytes of data from this piped input stream into an array of bytes. Less than *len* bytes will be read if the end of the data stream is reached. This method blocks until at least one byte of input is available. If a thread was providing data bytes to the connected piped output stream, but the thread is no longer alive, then an *IOException* is thrown.

## Example

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream(pos);

byte[] b = {10,20};
pos.write(b, 0, 2);
if (pis.available() > 0)
    Console.println("data: " + pis.read());
else
    Console.println("no data");
```

The piped input *read* method reads the data from the piped output stream and prints it to the console.

## Package [java.io](#)

### Class **PipedOutputStream**

extends [OutputStream](#) → [Object](#)

A piped output stream can be connected to a piped input stream to create a communications pipe. The piped output stream is the sending end of the pipe. Data is written to a *PipedOutputStream* object by one thread and data is read from the connected *PipedInputStream* by some other thread.

#### Public Constructors

- [PipedOutputStream](#)

#### Public Methods

- [close](#)
- [connect](#)
- [flush](#)
- [write](#)

#### Methods inherited from [java.io.OutputStream](#)

- [flush](#)
- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [PipedOutputStream](#)

### Public Constructor [PipedOutputStream](#)

```
PipedOutputStream()
```

```
PipedOutputStream(PipedInputStream sink)
```

#### Throws:

- [IOException](#)

#### Description:

##### constructor [PipedOutputStream\(\)](#):

This method constructs a piped output stream that is not yet connected to a piped input stream. It must be connected to a piped input stream, either by the receiver or the sender, before being used.

##### constructor [PipedOutputStream\(PipedInputStream sink\)](#):

This method constructs a piped output stream connected to the specified piped input stream *sink*. Data bytes written to this stream will then be available as input from *sink*.

Package [java.io](#)

## Class [PipedOutputStream](#)

### Public Method [close](#)

```
void close()
```

#### Overrides:

- [close](#) in class [OutputStream](#)

#### Description:

The *close* method closes this piped output stream and releases any system resources associated with this stream. This stream may no longer be used for writing bytes.

#### Example

```
PipedOutputStream pos = new PipedOutputStream();  
PipedInputStream pis = new PipedInputStream(pos);  
pos.close();  
pis.close();
```

The *close* method closes the piped input and output stream.

Package [java.io](#)

## Class [PipedOutputStream](#)

### Public Method connect

```
void connect(PipedInputStream sink)
```

#### Throws:

- [IOException](#)

#### Description:

This method connects this piped output stream to a receiver. If this object is already connected to some other piped input stream, an *IOException* is thrown.

#### Example

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream();

pos.connect(pis);
```

The *connect* method connects the piped output stream with the piped input stream.

Package [java.io](#)

## Class [PipedOutputStream](#)

### Public Method flush

```
void flush()
```

#### Overrides:

- [flush](#) in class [OutputStream](#)

#### Description:

This method flushes this piped output stream and forces any buffered output bytes to be written out. This will notify any readers that bytes are waiting in the pipe.

#### Example

```
PipedOutputStream pos = new PipedOutputStream();
...
pos.flush();
```

The example above flushes the piped output stream.



**Package [java.io](#)****Class [PipedOutputStream](#)****Public Method [write](#)**

```
void write(byte[] b, int off, int len)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [OutputStream](#)

**Throws:**

- [IOException](#)

**Description:****`write(byte[] b, int off, int len)` method:**

This method writes *len* bytes from the specified byte array starting at offset *off* to this piped output stream. If a thread was reading data bytes from the connected piped input stream, but the thread is no longer alive, then an *IOException* is thrown.

**`write(int b)` method:**

This method writes the specified *byte* to the piped output stream. If a thread was reading data byte from the connected piped input stream, but the thread is no longer alive, then an *IOException* is thrown.

**Example**

```
PipedOutputStream pos = new PipedOutputStream();  
byte[] b = {10,20};  
pos.write(b, 0, 2);
```

The example above writes the byte array to the piped output stream.

**Package [java.io](#)****Class [PipedReader](#)**

extends [Reader](#) → [Object](#)

The *PipedReader* class provides character-input streams.

## Public Constructors

- [PipedReader](#)

## Public Methods

- [close](#)
- [connect](#)
- [read](#)
- [ready](#)

## Methods inherited from [java.io.Reader](#)

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

## Class [PipedReader](#)

### Public Constructor PipedReader

```
PipedReader ()
```

```
PipedReader (PipedWriter source)
```

### Throws:

- [IOException](#)

### Description:

#### constructor PipedReader():

This method constructs a *PipedReader* so that it is not yet connected. It must be connected to a *PipedWriter* before being used.

#### constructor PipedReader(PipedWriter source):

This method constructs a *PipedReader* so that is connected to the piped writer *source*. Data written to *source* will then be available as input from this stream.

**Package [java.io](#)****Class [PipedReader](#)****Public Method close**

```
void close ()
```

**Overrides:**

- [close](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method closes this piped stream and releases any system resources associated with this stream.

**Example**

```
PipedReader pr = new PipedReader();
PipedWriter pw = new PipedWriter(pr);
pr.close();
pw.close();
```

The `close` method closes the piped reader and writer stream.

**Package [java.io](#)****Class [PipedReader](#)****Public Method connect**

```
void connect(PipedWriter source)
```

**Throws:**

- [IOException](#)

**Description:**

This method connects this piped reader to the piped writer `source`. An *IOException* will be thrown, if this object is already connected to some other piped writer.

**Example**

```
PipedReader pr = new PipedReader();
PipedWriter pw = new PipedWriter();

pr.connect(pw);
```

The `connect` method connects the piped reader stream with the piped writer stream.

**Package [java.io](#)****Class [PipedReader](#)****Public Method [read](#)**

```
int read()
```

**Overrides:**

- [read](#) in class [Reader](#)

**Throws:**

- [IOException](#)

```
int read(char[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [Reader](#)

**Throws:**

- [IOException](#)
- [InterruptedIOException](#)

**Description:****read() method:**

This method reads the next character from this piped stream. If no character is available because the end of the stream has been reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown. If a thread was providing data characters to the connected piped writer, but the thread is no longer alive, then an *IOException* is thrown.

**read(byte[] buf, int offset, int len) method:**

This method reads up to *len* bytes of data from this piped input stream into an array of bytes. Less than *len* bytes will be read if the end of the data stream is reached. This method blocks until at least one character of input is available. If a thread was providing data characters to the connected piped writer, but the thread is no longer alive, then an *IOException* is thrown.

**Example**

```
PipedReader pr = new PipedReader();
PipedWriter pw = new PipedWriter(pr);

byte[] b = {10,20};
pw.write(b, 0, 2);
if (pr.ready())
    Console.println("data: " + pr.read());
else
    Console.println("stream is not ready");
```

The piped input `read` method reads the data from the piped stream and prints it to the console.

Package [java.io](#)

Class [PipedReader](#)

**Public Method** `ready`

```
boolean ready()
```

**Overrides:**

- [ready](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method tell whether this stream is ready to be read. A piped character stream is ready if the circular buffer is not empty.

**Example**

```
PipedReader pr = new PipedReader();
PipedWriter pw = new PipedWriter(pr);

byte[] b = {10,20};
pw.write(b, 0, 2);
if (pr.ready())
    Console.println("data: " + pr.read());
else
    Console.println("stream is not ready");
```

The example reads the data from the piped stream and prints it to the console if the stream is ready.

Package [java.io](#)

Class [PipedWriter](#)

extends [Writer](#) → [Object](#)

The `PipedWriter` class provides character-output streams.

**Public Constructors**

- [PipedWriter](#)

**Public Methods**

- [close](#)
- [connect](#)
- [flush](#)
- [write](#)

## Methods inherited from [java.io.Writer](#)

- [flush](#)
- [close](#)
- [write](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [PipedWriter](#)

#### Public Constructor PipedWriter

```
PipedWriter ()
```

```
PipedWriter(PipedReader sink)
```

#### Throws:

- [IOException](#)

#### Description:

##### constructor PipedWriter():

This method constructs a *PipedWriter* so that it is not yet connected. It must be connected to a *PipedReader* before being used.

##### constructor PipedWriter(PipedReader sink):

This method constructs a *PipedWriter* so that is connected to the piped writer *source*. Data written to *sink* will then be available as input from this stream.

## Package [java.io](#)

### Class [PipedWriter](#)

#### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [Writer](#)

#### Description:

This method closes this piped stream and releases any system resources associated with this stream.

## Example

```
PipedReader pr = new PipedReader();
PipedWriter pw = new PipedWriter(pr);
pr.close();
pw.close();
```

The `close` method closes the piped reader and writer stream.

Package [java.io](#)

Class [PipedWriter](#)

### Public Method `connect`

```
void connect(PipedReader sink)
```

#### Throws:

- [IOException](#)

#### Description:

This method connects this piped writer to the piped reader. An *IOException* will be thrown, if this object is already connected to some other piped reader.

## Example

```
PipedReader pr = new PipedReader();
PipedWriter pw = new PipedWriter();

pw.connect(pr);
```

The `connect` method connects the piped writer stream with the piped reader stream.

Package [java.io](#)

Class [PipedWriter](#)

### Public Method `flush`

```
void flush()
```

#### Overrides:

- [flush](#) in class [Writer](#)

#### Description:

This method flushes this piped stream and forces any buffered output characters to be written out. This will notify any readers that characters are waiting in the pipe.

## Example

```
PipedWriter pw = new PipedWriter();
...
pw.flush();
```

The example above flushes the piped writer stream.

Package [java.io](#)

## Class [PipedWriter](#)

### Public Method write

```
void write(char b)
```

#### Throws:

- [IOException](#)

```
void write(char[] b, int off, int len)
```

#### Overrides:

- [write](#) in class [Writer](#)

#### Throws:

- [IOException](#)

#### Description:

##### **write(byte[] b, int off, int len) method:**

This method writes *len* characters from the specified character array starting at offset *off* to this piped output stream. If a thread was reading data characters from the connected piped input stream, but the thread is no longer alive, then an *IOException* is thrown.

##### **write(int b) method:**

This method writes the specified *char* to the piped output stream. If a thread was reading data character from the connected piped input stream, but the thread is no longer alive, then an *IOException* is thrown.

## Example

```
PipedWriter pw = new PipedWriter();
byte[] b = {10,20};
pw.write(b, 0, 2);
```

The example above writes the character array to the piped output stream.



## Package [java.io](#)

### Class **PrintStream**

extends [FilterOutputStream](#) → [OutputStream](#) → [Object](#)

Adds to an output stream the ability to print representations of various data values via calling the `print` and `println` methods. A `PrintStream` never throws an *IOException*, instead an internal flag is set that can be tested via the `checkError` method. Optionally a `PrintStream` can be created so as to flush automatically, i.e. the `flush` method is automatically called after a byte array is written, `println` is called or a newline character is written.

#### Public Constructors

- [PrintStream](#)

#### Public Methods

- [checkError](#)
- [close](#)
- [flush](#)
- [print](#)
- [println](#)
- [write](#)

#### Methods inherited from [java.io.FilterOutputStream](#)

- [close](#)
- [flush](#)
- [write](#)

#### Methods inherited from [java.io.OutputStream](#)

- [flush](#)
- [close](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class [PrintStream](#)

#### **Public Constructor `PrintStream`**

```
PrintStream(OutputStream out)
```

```
PrintStream(OutputStream out, boolean auto_flush)
```

**Description:****constructor `PrintStream(OutputStream out)`:**

Create a new print stream. This stream will not flush automatically. Values are printed to the output stream *out*.

**constructor `PrintStream(OutputStream out, boolean auto_flush)`:**

Create a new print stream, writing to the output stream *out*. If *auto\_flush* is true the output buffer will be flushed whenever a byte array is written, the `println` method is called or a newline character is written.

**Description:****constructor `PrintStream(OutputStream out)`:**

Constructs a new print stream. This stream will not flush automatically.

**constructor `PrintStream(OutputStream out, boolean auto_flush)`:**

Constructs a new print stream. If the *auto\_flush* flag is true, the output buffer will be flushed whenever a byte array is written, one of the `println` methods is invoked, or a newline character or byte (`'\n'`) is written.

Package [java.io](#)

**Class [PrintStream](#)****Public Method `checkError`**

```
boolean checkError()
```

**Description:**

Flush the stream and check its error state. The internal error state is set to true when the underlying output stream throws an `IOException`, so this method returns true if and only if this stream has encountered an `IOException` or the protected method `setError` has been called.

**Description:**

This method flushes the stream and check its error state. The internal error state is set to true when the underlying output stream throws an `IOException` other than `InterruptedIOException`, and when the `setError` method is invoked.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintStream ps = new PrintStream(fos);
...
if (ps.checkError())      Console.println("Error");
```

The example above checks a occurred error.

**Package [java.io](#)**  
**Class [PrintStream](#)****Public Method close**

```
void close()
```

**Overrides:**

- [close](#) in class [FilterOutputStream](#)

**Description:**

Close this stream. This is done by flushing the stream and then closing the underlying output stream.

**Description:**

This method closes the stream. This is done by flushing the stream and then closing the underlying output stream.

**Example**

```
File file = new File("testFile");  
String fileString = "Hello World";  
OutputStream fos = new FileOutputStream(file);  
PrintStream pw = new PrintStream(fos);  
...  
ps.close();
```

The example above closes the stream.

**Package [java.io](#)**  
**Class [PrintStream](#)****Public Method flush**

```
void flush()
```

**Overrides:**

- [flush](#) in class [FilterOutputStream](#)

**Description:****Description:**

Flushes this stream, i.e. writes any buffered output bytes to the underlying output stream and then flushing that stream.

**Description:**

This method flushes the stream. This is done by writing any buffered output bytes to the underlying output stream and then flushing that stream.

## Example

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintStream ps = new PrintStream(fos);
...
ps.flush();
```

The example above flushes the stream.

Package [java.io](#)

## Class [PrintStream](#)

### Public Method print

```
void print(boolean b)
void print(char c)
void print(char[] s)
void print(float f)
void print(int i)
void print(Object obj)
void print(String s)
```

### Description:

#### **print(boolean b) method:**

Print a boolean value. The return value of *String.valueOf(boolean)* is translated into bytes according to the default platform character encoding and these bytes are written as with *write(int)*.

#### **print(char c) method:**

Print a character. The character is translated into the platform's default character encoding and these bytes are written as with *write(int)*.

#### **print(char[] s) method:**

Print an array of characters. The characters are converted into bytes according to the default encoding of the platform and these bytes are written as with *write(int)*.

#### **print(float f) method:**

Print a floating-point number. The string produced by *String.valueOf(float)* is translated into bytes according to the default encoding of the platform and these bytes are written as with *write(int)*.

**print(int i) method:**

Print an integer number. The string produced by *String.valueOf(int)* is translated into bytes according to the default encoding of the platform and these bytes are written as with *write(int)*.

**print(Object obj) method:**

Print an object. The string produced by *String.valueOf(Object)* is translated into bytes according to the default encoding of the platform and these bytes are written as with *write(int)*.

**print(String s) method:**

Print a string. If the argument is *null* the string *"null"* is printed, otherwise the characters of the string are converted into bytes according to the default encoding of the platform and these bytes are written as with *write(int)*.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintStream ps = new PrintStream(fos);
...
ps.print(fileString);
```

The example above prints the specified string to the file.

Package [java.io](#)

**Class [PrintStream](#)****Public Method println**

```
void println()
void println(boolean b)
void println(char c)
void println(char[] s)
void println(float f)
void println(int i)
void println(Object obj)
void println(String s)
```

**Description:****println() method:**

Terminate the current line by writing the line separator string as defined by the system property *line.separator* (See method [getProperty](#) in class [System](#)).

**println(boolean b) method:**

Print a boolean and then terminate the line. Works like calling `print(boolean)` and then `println()`.

**println(char c) method:**

Print a character and then terminate the line. Works like calling `print(char)` and then `println()`.

**println(char[] s) method:**

Print a character array and then terminate the line. Works like calling `print(char[])` and then `println()`.

**println(float f) method:**

Print a float value and then terminate the line. Works like calling `print(float)` and then `println()`.

**println(int i) method:**

Print an integer value and then terminate the line. Works like calling `print(int)` and then `println()`.

**println(Object obj) method:**

Print an object and then terminate the line. Works like calling `print(Object)` and then `println()`.

**println(String s) method:**

Print a String and then terminate the line. Works like calling `print(String)` and then `println()`.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintStream ps = new PrintStream(fos);
...
ps.println(fileString);
```

The example above prints the specified string to the file.

Package [java.io](#)

Class [PrintStream](#)

**Public Method write**

```
void write(byte[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [FilterOutputStream](#)

```
void write(int b)
```

**Overrides:**

- [write](#) in class [FilterOutputStream](#)

**Description:****write(byte[] buf, int offset, int len) method:**

Writes *len* bytes from *buf*, starting at offset *offset* to this stream. If automatic flushing is enabled then the *flush* method is called. The bytes will be written without additional encoding.

**write(int b) method:**

Writes the specified byte to this stream. If the byte is a newline and automatic flushing is enabled the *flush* method will be called. The bytes will be written without additional encoding.

```
File file = new File("testFile");
int fileInt = 100;
OutputStream fos = new FileOutputStream(file);
PrintStream ps = new PrintStream(fos);
...
ps.write(fileInt);
```

The example above prints the specified character to the file.

**Package [java.io](#)****Class [PrintWriter](#)**

extends [Writer](#) → [Object](#)

Class for printing formatted output to a text output stream using methods like `print` and `println`. Implements all the print methods found in [PrintStream](#). In contrast to the [PrintStream](#) class automatic flushing (if enabled) will only be done when one of the `println`-methods are called, rather than whenever a newline character is output.

**Public Constructors**

- [PrintWriter](#)

**Public Methods**

- [checkError](#)
- [close](#)
- [flush](#)
- [print](#)
- [println](#)
- [write](#)

### Methods inherited from [java.io.Writer](#)

- [flush](#)
- [close](#)
- [write](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.io](#)

## Class [PrintWriter](#)

### Public Constructor [PrintWriter](#)

```
PrintWriter(OutputStream out)
PrintWriter(OutputStream out, boolean autoflush)
PrintWriter(Writer wr)
PrintWriter(Writer wr, boolean autoflush)
```

### Description:

#### **constructor [PrintWriter\(OutputStream out\):](#)**

Constructs a new print writer. This stream will not flush automatically.

#### **constructor [PrintWriter\(OutputStream out, boolean autoflush\):](#)**

Constructs a new print writer. If the *autoflush* flag is true, the *println* method will flush the output buffer.

#### **constructor [PrintWriter\(Writer wr\):](#)**

Constructs a new print writer. This stream will not flush automatically.

#### **constructor [PrintWriter\(Writer wr, boolean autoflush\):](#)**

Constructs a new print writer. If the *autoflush* flag is true, the *println* method will flush the output buffer.

### Package [java.io](#)

## Class [PrintWriter](#)

### Public Method [checkError](#)

```
boolean checkError()
```



**Description:**

This method flushes the stream and check its error state. Once the stream encounters an error, this routine will return true on all successive calls.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintWriter pw = new PrintWriter(fos);
...
if (pw.checkError()) Console.println("Error");
```

The example above checks a occurred error.

Package [java.io](#)

Class [PrintWriter](#)

**Public Method close**

```
void close()
```

**Overrides:**

- [close](#) in class [Writer](#)

**Description:**

This method closes this stream.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintWriter pw = new PrintWriter(fos);
...
pw.close();
```

The example above closes the stream.

Package [java.io](#)

Class [PrintWriter](#)

**Public Method flush**

```
void flush()
```

**Overrides:**

- [flush](#) in class [Writer](#)

**Description:**

This method flushes this stream.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintWriter pw = new PrintWriter(fos);
...
pw.flush();
```

The example above flushes the stream.

Package [java.io](#)

Class [PrintWriter](#)

**Public Method print**

```
void print(boolean bool)
void print(char ch)
void print(char[] charArray)
void print(float fnum)
void print(int inum)
void print(Object obj)
void print(String str)
```

**Description:****print(boolean bool) method:**

This method prints a boolean value.

**print(char ch) method:**

This method prints a character value.

**print(char[] charArray) method:**

This method prints an array of characters.

**print(float fnum) method:**

This method prints a floating point number.

**print(int inum) method:**

This method prints an integer value.

**print(Object obj) method:**

This method prints an object.

**print(String str) method:**

This method prints a string. If the argument is *null* then the string "*null*" is printed.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintWriter pw = new PrintWriter(fos);
...
pw.print(fileString);
```

The example above prints the specified string to the file.

Package [java.io](#)

**Class [PrintWriter](#)****Public Method println**

```
void println()
void println(boolean bool)
void println(char ch)
void println(char[] charArray)
void println(float fnum)
void println(int inum)
void println(Object obj)
void println(String str)
```

**Description:****println() method:**

This method terminates the current line by writing a line separator string.

**println(boolean bool) method:**

This method prints a boolean value and terminates the line with a line terminator.

**println(char ch) method:**

This method prints a character and terminates the line with a line terminator.

**println(char[] charArray) method:**

This method prints a character array and terminates the print operation with a line terminator.

**println(float fnum) method:**

This method prints a floating point number and terminates the line with a line terminator.

**println(int inum) method:**

This method prints an integer number and terminates the line with a line terminator.

**println(Object obj) method:**

This method prints an object and terminates the line with a line terminator.

**println(String str) method:**

This method prints a string and terminates the line with a line terminator.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World";
OutputStream fos = new FileOutputStream(file);
PrintWriter pw = new PrintWriter(fos);
...
pw.println(fileString);
```

The example above prints the specified string to the file.

**Package [java.io](#)****Class [PrintWriter](#)****Public Method write**

```
void write(char[] charArray)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(char[] charArray, int offset, int count)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(int ch)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(String str)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(String str, int offset, int count)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Description:****write(char[] charArray) method:**

This method writes the specified character array to this stream. If the byte is a newline and automatic flushing is enabled then the *flush* method will be invoked.

**write(char[] charArray, int offset, int count) method:**

This method writes *count* characters from the specified character array starting at offset *off* to this stream. If automatic flushing is enabled then the *flush* method will be invoked.

**write(int chr) method:**

This method writes the specified character to this stream. If the byte is a newline and automatic flushing is enabled then the *flush* method will be invoked.

**write(String str) method:**

This method writes the specified string to this stream. If the byte is a newline and automatic flushing is enabled then the *flush* method will be invoked.

**write(String str, int offset, int count) method:**

This method writes *count* characters from the specified string starting at offset *off* to this stream. If automatic flushing is enabled then the *flush* method will be invoked.

**Example**

```
File file = new File("testFile");  
int fileInt = 100;
```

```
OutputStream fos = new FileOutputStream(file);
PrintWriter pw = new PrintWriter(fos);
...
pw.write(fileInt);
```

The example above prints the specified character to the file.

## Package [java.io](#)

### Class **PushbackInputStream**

extends [FilterInputStream](#) → [InputStream](#) → [Object](#)

A `PushbackInputStream` adds the ability to "*push back*" or "*unread*" one byte. This is useful in situation where it is convenient for a fragment of code to read an indefinite number of bytes that are delimited by a particular byte value. After reading the terminating byte, the code fragment can "*unread*" it, so that the next read operation on the input stream will reread the byte that was pushed back.

#### Public Constructors

- [PushbackInputStream](#)

#### Public Methods

- [available](#)
- [close](#)
- [markSupported](#)
- [read](#)
- [reset](#)
- [skip](#)
- [unread](#)

#### Methods inherited from [java.io.FilterInputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

#### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)

- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [PushbackInputStream](#)

### Public Constructor PushbackInputStream

```
PushbackInputStream(InputStream in)
```

```
PushbackInputStream(InputStream in, int bufsize)
```

#### Description:

##### constructor [PushbackInputStream\(InputStream in\)](#):

Constructs a *PushbackInputStream* and saves its arguments, the input stream *in*, for later use.

##### constructor [PushbackInputStream\(InputStream in, int bufsize\)](#):

Constructs a *PushbackInputStream* with a pushback buffer of the specified *size*, and saves its argument, the input stream *in*, for later use.

Package [java.io](#)

## Class [PushbackInputStream](#)

### Public Method available

```
int available()
```

#### Overrides:

- [available](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

#### Description:

This method returns the number of bytes that can be read from this input stream without blocking. This method calls the *available* method of the underlying input stream and returns that value plus the number of bytes that have been pushed back.

#### Example

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
PushbackInputStream pis = new PushbackInputStream(in, 20);
Logger.log("available bytes: " + pis.available());
```

The example above writes the available bytes which can be read from the input stream without blocking to the logger.

Package [java.io](#)

## Class [PushbackInputStream](#)

### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

#### Description:

This method closes this stream and releases any system resources associated with the stream.

#### Example

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
PushbackInputStream pis = new PushbackInputStream(in, 20);

pis.close();
```

The *close* method closes the stream.

Package [java.io](#)

## Class [PushbackInputStream](#)

### Public Method markSupported

```
boolean markSupported ()
```

#### Overrides:

- [markSupported](#) in class [FilterInputStream](#)

#### Description:

This method tests if this input stream supports the *mark* and *reset* methods, which it does not.

#### Example

```
File file = new File("testFile");
InputStream in = new FileInputStream(file);
PushbackInputStream pis = new PushbackInputStream(in, 20);
```



```
if (pis.markSupported())
    Logger.log("This stream supports the mark method");
```

The example above checks if the input stream supports the *mark* method.

Package [java.io](#)

## Class [PushbackInputStream](#)

### Public Method [read](#)

```
int read()
```

#### Overrides:

- [read](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

#### Overrides:

- [read](#) in class [FilterInputStream](#)

#### Throws:

- [IOException](#)

#### Description:

##### [read\(\)](#) method:

This method reads the next bytes from this input stream. The value byte is returned as an *int* in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value *-1* is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

##### [read\(byte\[\] buf, int offset, int len\)](#) method:

This method reads up to *len* bytes of data from this input stream into an array of bytes. This method reads any pushed back byte first. Afterwards it reads from the underlying input stream if fewer than *len* bytes have been read. This method blocks until at least 1 byte of input is available.

Package [java.io](#)

## Class [PushbackInputStream](#)

### Public Method [reset](#)

```
void reset()
```

**Overrides:**

- [reset](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:**

Repositions this stream to the position marked by calling the *mark* method.

Package [java.io](#)

**Class [PushbackInputStream](#)****Public Method skip**

```
int skip(int num_bytes)
```

**Overrides:**

- [skip](#) in class [FilterInputStream](#)

**Throws:**

- [IOException](#)

**Description:**

This method skips over and discards *n* bytes of data from this input stream. If *n* is negative, no bytes are skipped.

The *skip* method of *PushbackInputStream* first skips over the bytes in the pushback buffer, if any. It then calls the *skip* method of the underlying input stream if more bytes need to be skipped. The actual number of bytes skipped is returned.

Package [java.io](#)

**Class [PushbackInputStream](#)****Public Method unread**

```
void unread(byte[] buf)
```

**Throws:**

- [IOException](#)

```
void unread(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)

```
void unread(int b)
```

**Throws:**

- [IOException](#)

**Description:****unread(byte[] buf) method:**

This method pushes back an array of bytes by copying it to the front of the pushback buffer. After this method returns, the next byte to be read will have the value `buf[0]`, the byte after that will have the value `buf[1]`, and so forth.

**unread(byte[] buf, int offset, int len) method:**

This method pushes back a portion of an array of bytes by copying it to the front of the pushback buffer. After this method returns, the next byte to be read will have the value `buf[offset]`, the byte after that will have the value `buf[offset+1]`, and so forth.

**unread(int b) method:**

This method pushes back a byte by copying it to the front of the pushback buffer. After this method returns, the next byte to be read will have the value (byte)b.

**Package [java.io](#)****Class PushbackReader**

extends [FilterReader](#) → [Reader](#) → [Object](#)

A character stream reader that allows characters to be pushed back into the stream.

**Public Constructors**

- [PushbackReader](#)

**Public Methods**

- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [ready](#)
- [reset](#)
- [skip](#)
- [unread](#)

**Methods inherited from [java.io.FilterReader](#)**

- [mark](#)
- [markSupported](#)
- [reset](#)
- [ready](#)
- [skip](#)
- [read](#)
- [close](#)

**Methods inherited from [java.io.Reader](#)**

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [PushbackReader](#)****Public Constructor PushbackReader**

```
PushbackReader(Reader in)
```

```
PushbackReader(Reader in, int bufsize)
```

**Throws:**

- [IllegalArgumentException](#)

**Description:****constructor [PushbackReader\(Reader in\)](#):**

Constructs a new pushback reader with a one character pushback buffer.

**constructor [PushbackReader\(Reader in, int bufsize\)](#):**

Constructs a new pushback reader with a pushback buffer of the specified size.

**Package [java.io](#)****Class [PushbackReader](#)****Public Method close**

```
void close()
```

**Overrides:**

- [close](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method closes this stream.

**Example**

```
File file = new File("testFile");
Reader in = new FileReader(file);
PushbackReader pr = new PushbackReader(in, 20);
...
pr.close();
```

The `close` method closes the stream.

Package [java.io](#)

**Class [PushbackReader](#)****Public Method mark**

```
void mark(int read_limit)
```

**Overrides:**

- [mark](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method marks the present position in the stream. The `mark` for class `PushbackReader` always throws an exception.

**Example**

```
File file = new File("testFile");
Reader in = new FileReader(file);
PushbackReader pr = new PushbackReader(in, 20);
...
pr.mark(2);
```

The example above marks the current position in the stream.

**Package [java.io](#)****Class [PushbackReader](#)****Public Method `markSupported`**

```
boolean markSupported()
```

**Overrides:**

- [markSupported](#) in class [FilterReader](#)

**Description:**

This method tells whether this stream supports the *mark* operation, which it does not.

**Example**

```
File file = new File("testFile");
Reader in = new FileReader(file);
PushbackReader pr = new PushbackReader(in, 20);
...
if (pr.markSupported())
    Logger.log("This stream supports the mark!");
```

The *markSupported* method checks if the mark is supported.

**Package [java.io](#)****Class [PushbackReader](#)****Public Method `read`**

```
int read()
```

**Overrides:**

- [read](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)

```
int read(char[] b, int offset, int len)
```

**Overrides:**

- [read](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****read() method:**

This method reads a single character.

**read(byte[] buf, int offset, int len) method:**

This method reads characters into a portion of an array.

**Example**

```
File file = new File("testFile");
Reader in = new FileReader(file);
PushbackReader pr = new PushbackReader(in, 20);
...
int data = pr.read();
```

The example above reads the next single character.

Package [java.io](#)

**Class [PushbackReader](#)****Public Method ready**

```
boolean ready()
```

**Overrides:**

- [ready](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method tells whether this stream is ready to be read.

**Example**

```
File file = new File("testFile");
Reader in = new FileReader(file);
PushbackReader pr = new PushbackReader(in, 20);
...
if (pr.ready())
    Logger.log("This stream is ready!");
```

The *ready* method checks if the stream is ready.

Package [java.io](#)  
Class [PushbackReader](#)

**Public Method [reset](#)**

```
void reset()
```

**Overrides:**

- [reset](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)

**Description:**

This method throws always an exception.

Package [java.io](#)  
Class [PushbackReader](#)

**Public Method [skip](#)**

```
int skip(int num_chars)
```

**Overrides:**

- [skip](#) in class [FilterReader](#)

**Throws:**

- [IOException](#)

**Description:**

Skips *num\_chars* characters.

Package [java.io](#)  
Class [PushbackReader](#)

**Public Method [unread](#)**

```
void unread(char[] buf)
```

**Throws:**

- [IOException](#)

```
void unread(char[] b, int offset, int len)
```

**Throws:**

- [IOException](#)



```
void unread(int b)
```

**Throws:**

- [IOException](#)

**Description:****unread(char[] buf) method:**

Push back an array of characters by copying it to the front of the pushback buffer. After this method returns, the next character to be read will have the value `cbuf[0]`, the character after that will have the value `cbuf[1]`, and so forth.

**unread(char[] b, int offset, int len) method:**

Push back a portion of an array of characters by copying it to the front of the pushback buffer. After this method returns, the next character to be read will have the value `cbuf[0]`, the character after that will have the value `cbuf[1]`, and so forth.

**unread(int b) method:**

This method push back a single character.

**Example**

```
File file = new File("testFile");
Reader in = new FileReader(file);
PushbackReader pr = new PushbackReader(in, 20);
...
pr.unread(data);
```

The example above pushes back a single character.

**Package [java.io](#)****Class `RandomAccessFile`**

extends [Object](#)

implements [DataOutput](#), [DataInput](#)

Supports reading and writing to a random access file. A random access file behaves like a large array of bytes stored in the file system. The file pointer is an index into this array. Input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. Output operations that write past the current end of the implied array cause the array to be extended. The file pointer can be read by the `getFilePointer` method and set by the `seek` method.

It is generally true of all the reading routines in this class that if end-of-file is reached before the desired number of bytes has been read, an `EOFException` (which is a kind of `IOException`) is thrown. If any byte cannot be read for any reason other than end-of-file, an `IOException` other than `EOFException` is thrown. In particular, an `IOException` may be thrown if the stream has been closed.

## Public Constructors

- [RandomAccessFile](#)

## Public Methods

- [close](#)
- [getFD](#)
- [getFilePointer](#)
- [length](#)
- [read](#)
- [readBoolean](#)
- [readByte](#)
- [readChar](#)
- [readFloat](#)
- [readFully](#)
- [readInt](#)
- [readLine](#)
- [readShort](#)
- [readUnsignedByte](#)
- [readUnsignedShort](#)
- [readUTF](#)
- [seek](#)
- [setLength](#)
- [skipBytes](#)
- [write](#)
- [writeBoolean](#)
- [writeByte](#)
- [writeBytes](#)
- [writeChar](#)
- [writeChars](#)
- [writeFloat](#)
- [writeInt](#)
- [writeShort](#)
- [writeUTF](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Constructor RandomAccessFile

```
RandomAccessFile(File file, String mode)
```

### Throws:

- [IllegalArgumentException](#)
- [SecurityException](#)

- [IOException](#)

```
RandomAccessFile(String name, String mode)
```

**Throws:**

- [IllegalArgumentException](#)
- [SecurityException](#)
- [IOException](#)

**Description:****Important Note**

Make sure that any stream accessing a file is closed before a new stream for the same file is created.

**constructor RandomAccessFile(File file, String mode):**

Creates a random access file stream to read from, and optionally write to, the file specified by the File argument. A new FileDescriptor object is created to represent this file connection. RandomAccessFile supports the notion of a file pointer. The file pointer indicates the current location in the file. When the file is first created, the file pointer is set to 0, indicating the beginning of the file.

The mode argument must either be "r" or "rw" for read or read and write access.

**constructor RandomAccessFile(String name, String mode):**

Creates a random access file stream to read from, and optionally write to, a file with the specified name. A new FileDescriptor object is created to represent the connection to the file. RandomAccessFile supports the notion of a file pointer. The file pointer indicates the current location in the file. When the file is first created, the file pointer is set to 0, indicating the beginning of the file.

The mode argument must either be "r" or "rw" for read or read and write access.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method close**

```
void close ()
```

**Throws:**

- [IOException](#)

**Description:**

Closes this random access file stream and releases any system resources associated with the stream. A closed random access file cannot perform input or output operations and cannot be reopened.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method getFD

```
final FileDescriptor getFD ()
```

#### Throws:

- [IOException](#)

#### Description:

Returns the opaque file descriptor object associated with this stream.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method getFilePointer

```
int getFilePointer ()
```

#### Throws:

- [IOException](#)

#### Description:

Returns the current offset in bytes from the beginning of the file, at which the next read or write occurs.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method length

```
int length ()
```

#### Throws:

- [IOException](#)

#### Description:

Returns the length of this file in bytes.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method read

```
int read ()
```

**Throws:**

- [IOException](#)

```
int read(byte[] buf)
```

**Throws:**

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****read() method:**

Reads a byte of data from this file. The file pointer will be adjusted by the number of bytes read. The byte is returned as an integer in the range 0 to 255 (0x00-0x0ff). The method blocks if no input is yet available. This method behaves the same way as the `InputStream.read()` method of `InputStream`.

**read(byte[] buf) method:**

Reads up to `b.length` bytes of data from this file into an array of bytes. The file pointer will be adjusted by the number of bytes read. Returns the total number of bytes read into the buffer. Note that this number might be smaller than `buf.length` if the end of the file was reached. If there is no more data to read because EOF was reached, -1 is returned.

The method blocks until at least one byte of input is available. This method behaves the same way as the `InputStream.read(byte[])` method of `InputStream`.

**read(byte[] buf, int offset, int len) method:**

Reads up to `len` bytes of data from this file into an array of bytes. The file pointer will be adjusted by the number of bytes read. Returns the total number of bytes read into the buffer. Note that this number might be smaller than `len` if the end of the file was reached. If there is no more data to read because EOF was reached, -1 is returned.

The method blocks until at least one byte of input is available. This method behaves in the exactly the same way as the `InputStream.read(byte[], int, int)` method of `InputStream`.

**Package [java.io](#)****Class [RandomAccessFile](#)****Public Method [readBoolean](#)**

```
final boolean readBoolean()
```

**Throws:**

- [EOFException](#)

- [IOException](#)

**Description:**

Reads a boolean from this file. This method reads a single byte from the file, starting at the current file pointer. A value of 0 represents false. Any other value represents true. This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method [readByte](#)**

```
final byte readByte()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads a signed eight-bit value from this file, starting from the current file pointer.

This method blocks until the byte is read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method [readChar](#)**

```
final char readChar()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads a Unicode character from this file. This method reads two bytes from the file, starting at the current file pointer.

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method [readFloat](#)**

```
final float readFloat()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads a float from this file. This method reads an int value, starting at the current file pointer, as if by the readInt method and then converts that int to a float using the intBitsToFloat method in class Float.

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method readFully**

```
final void readFully(byte[] buf)
```

**Throws:**

- [EOFException](#)
- [IOException](#)

```
final void readFully(byte[] buf, int offset, int len)
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:****readFully(byte[] buf) method:**

Reads b.length bytes from this file into the byte array, starting at the current file pointer. This method reads repeatedly from the file until the requested number of bytes are read. This method blocks until the requested number of bytes are read, the end of the stream is detected, or an exception is thrown.

**readFully(byte[] buf, int offset, int len) method:**

Reads exactly len bytes from this file into the byte array, starting at the current file pointer. This method reads repeatedly from the file until the requested number of bytes are read. This method blocks until the requested number of bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method readInt**

```
final int readInt()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads a signed 32-bit integer from this file. This method reads 4 bytes from the file, starting at the current file pointer.

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method [readLine](#)**

```
final String readLine()
```

**Throws:**

- [IOException](#)

**Description:**

Reads the next line of text from this file. This method successively reads bytes from the file, starting at the current file pointer, until it reaches a line terminator or the end of the file. Each byte is converted into a character by taking the byte's value for the lower eight bits of the character and setting the high eight bits of the character to zero. This method does not, therefore, support the full Unicode character set.

A line of text is terminated by a carriage-return character ('\r'), a newline character ('\n'), a carriage-return character immediately followed by a newline character, or the end of the file. Line-terminating characters are discarded and are not included as part of the string returned.

This method blocks until a newline character is read, a carriage return and the byte following it are read (to see if it is a newline), the end of the file is reached, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method [readShort](#)**

```
final short readShort()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads a signed 16-bit number from this file. The method reads two bytes from this file, starting at the current file pointer.



This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

Class [RandomAccessFile](#)

**Public Method [readUnsignedByte](#)**

```
final int readUnsignedByte()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads a signed 16-bit number from this file. The method reads two bytes from this file, starting at the current file pointer.

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

Class [RandomAccessFile](#)

**Public Method [readUnsignedShort](#)**

```
final int readUnsignedShort()
```

**Throws:**

- [EOFException](#)
- [IOException](#)

**Description:**

Reads an unsigned 16-bit number from this file. This method reads two bytes from the file, starting at the current file pointer.

This method blocks until the two bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

Class [RandomAccessFile](#)

**Public Method [readUTF](#)**

```
final String readUTF()
```

**Throws:**

- [EOFException](#)
- [UTFDataFormatException](#)
- [IOException](#)

**Description:**

The first two bytes are read, starting from the current file pointer, as if by `readUnsignedShort`. This value gives the number of following bytes that are in the encoded string, not the length of the resulting string. The following bytes are then interpreted as bytes encoding characters in the UTF-8 format and are converted into characters.

This method blocks until all the bytes are read, the end of the stream is detected, or an exception is thrown.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method `seek`**

```
void seek(int pos)
```

**Throws:**

- [IOException](#)

**Description:**

Sets the file-pointer offset, measured from the beginning of this file (pos 0), at which the next read or write occurs. The offset may be set beyond the end of the file. Setting the offset beyond the end of the file does not change the file length. The file length will change only by writing after the offset has been set beyond the end of the file (bytes between will eventually be filled with 0).

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method `setLength`**

```
void setLength(int newlen)
```

**Throws:**

- [IOException](#)

**Description:**

Sets the length of this file.

If the present length of the file as returned by the `length` method is greater than the `newLength` argument then the file will be truncated. In this case, if the file offset as returned by the `getFilePointer` method is greater than `newLength` then after this method returns the offset will be equal to `newLength`.

If the present length of the file as returned by the `length` method is smaller than the `newLength` argument then the file will be extended. In this case, the contents of the extended portion of the file are not defined.

**Package [java.io](#)****Class [RandomAccessFile](#)****Public Method skipBytes**

```
int skipBytes(int n)
```

**Throws:**

- [IOException](#)

**Description:**

Attempts to skip over n bytes of input discarding the skipped bytes.

This method may skip over some smaller number of bytes, possibly zero. This may result from any of a number of conditions; reaching end of file before n bytes have been skipped is only one possibility. This method never throws an EOFException. The actual number of bytes skipped is returned. If n is negative, no bytes are skipped.

**Package [java.io](#)****Class [RandomAccessFile](#)****Public Method write**

```
void write(byte[] buf)
```

**Throws:**

- [IOException](#)

```
void write(byte[] buf, int offset, int len)
```

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

```
void write(int b)
```

**Throws:**

- [IOException](#)

**Description:****`write(byte[] buf)` method:**

Writes b.length bytes from the specified byte array to this file, starting at the current file pointer. The file pointer will be adjusted by the number of bytes written.

**write(byte[] buf, int offset, int len) method:**

Writes len bytes to this file, taken from the specified byte array starting at offset off in the array. The file pointer will be adjusted by the number of bytes written.

**write(int b) method:**

Writes the specified byte to this file. The write starts at the current file pointer. The file pointer will be adjusted by the number of bytes written.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method writeBoolean**

```
final void writeBoolean(boolean b)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a boolean to the file as a one-byte value. The value true is written out as the value (byte)1; the value false is written out as the value (byte)0. The write starts at the current position of the file pointer.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method writeByte**

```
final void writeByte(int b)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a byte to the file as a one-byte value. The write starts at the current position of the file pointer.

Package [java.io](#)

**Class [RandomAccessFile](#)****Public Method writeBytes**

```
final void writeBytes(String s)
```

**Throws:**

- [IOException](#)

**Description:**

Writes the string to the file as a sequence of bytes. Each character in the string is written out, in sequence, by discarding its high eight bits. The write starts at the current position of the file pointer.

Package [java.io](#)

Class [RandomAccessFile](#)

**Public Method writeChar**

```
final void writeChar(int c)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a char to the file as a two-byte value, high byte first. The write starts at the current position of the file pointer.

Package [java.io](#)

Class [RandomAccessFile](#)

**Public Method writeChars**

```
final void writeChars(String s)
```

**Throws:**

- [IOException](#)

**Description:**

Writes a string to the file as a sequence of characters. Each character is written to the data output stream as if by the writeChar method. The write starts at the current position of the file pointer.

Package [java.io](#)

Class [RandomAccessFile](#)

**Public Method writeFloat**

```
final void writeFloat(float f)
```

**Throws:**

- [IOException](#)

**Description:**

Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first. The write starts at the current position of the file pointer.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method writeInt

```
final void writeInt(int i)
```

#### Throws:

- [IOException](#)

#### Description:

Writes an int to the file as four bytes, high byte first. The write starts at the current position of the file pointer.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method writeShort

```
final void writeShort(int s)
```

#### Throws:

- [IOException](#)

#### Description:

Writes a short to the file as two bytes, high byte first. The write starts at the current position of the file pointer.

Package [java.io](#)

## Class [RandomAccessFile](#)

### Public Method writeUTF

```
final void writeUTF(String s)
```

#### Throws:

- [IOException](#)

#### Description:

Writes a string to the file using UTF-8 encoding.

First, two bytes are written to the file, starting at the current file pointer, as if by the writeShort method giving the number of bytes to follow. This value is the number of bytes actually written out, not the length of the string. Following the length, each character of the string is output, in sequence, using the UTF-8 encoding for each character.

## Package [java.io](#)

### Abstract Class `Reader`

extends [Object](#)

Abstract class for reading character streams. A subclass has to implement the methods `read` and `close`.

#### Public Methods

- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [ready](#)
- [reset](#)
- [skip](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.io](#)

### Class `Reader`

#### Public Method `close`

```
abstract void close()
```

#### Throws:

- [IOException](#)

#### Description:

This method closes this reader. Once a stream has been closed, further `read`, `ready`, `mark`, or `reset` method invocations will throw an `IOException`.

## Package [java.io](#)

### Class `Reader`

#### Public Method `mark`

```
void mark(int readLimit)
```

#### Throws:

- [IOException](#)

**Description:**

Mark the current position in the stream. A call to the reset method will attempt to reposition the stream to this point.

The parameter *readLimit* specifies the limit on the number of characters that may be read while still preserving the mark. After reading this many characters, attempting to reset the stream may fail.

Package [java.io](#)

Class [Reader](#)

**Public Method `markSupported`**

```
boolean markSupported()
```

**Description:**

This method tell whether this stream supports the *mark* operation. The default implementation returns always false. Subclasses should override this method.

Package [java.io](#)

Class [Reader](#)

**Public Method `read`**

```
int read()
```

**Throws:**

- [IOException](#)

```
int read(char[] buf)
```

**Throws:**

- [IOException](#)

```
abstract int read(char[] buf, int offset, int count)
```

**Throws:**

- [IOException](#)

**Description:****`read()` method:**

The *read* method reads a single character and will block until this character is available, an I/O error occurs, or the end of the stream is reached. Subclasses that intend to support efficient single characters input should override this method.



**read(char[] buffer) method:**

This method reads characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

**read(char[] buffer, int offset, int len) method:**

This method reads characters into a portion of an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

**Package [java.io](#)  
Class [Reader](#)****Public Method ready**

```
boolean ready()
```

**Throws:**

- [IOException](#)

**Description:**

This method tell whether the stream is ready to be read.

**Package [java.io](#)  
Class [Reader](#)****Public Method reset**

```
void reset()
```

**Throws:**

- [IOException](#)

**Description:**

This method resets the stream. If the stream has been marked, then attempt to reposition it at the mark. If the stream has not been marked, then attempt to reset it in some way appropriate to the particular stream.

**Package [java.io](#)  
Class [Reader](#)****Public Method skip**

```
int skip(int count)
```

**Throws:**

- [IOException](#)

**Description:**

This method skip the specified number of characters. This method will block until some characters are available, an I/O error occurs, or the end of the stream is reached.

Package [java.io](#)

**Class [SequenceInputStream](#)**

extends [InputStream](#) → [Object](#)

A *SequenceInputStream* represents the logical concatenation of other input streams. It starts out with an ordered collection of input streams and reads from the first until end of file is reached, whereupon it reads from the second one, and so on, until end of file is reached on the last of the contained input stream.

**Public Constructors**

- [SequenceInputStream](#)

**Public Methods**

- [available](#)
- [close](#)
- [read](#)

**Methods inherited from [java.io.InputStream](#)**

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

**Class [SequenceInputStream](#)****Public Constructor [SequenceInputStream](#)**

```
SequenceInputStream(Enumeration stream_list)
```

```
SequenceInputStream(InputStream first_stream, InputStream second_stream)
```

**Description:****constructor `SequenceInputStream(Enumeration stream_list)`:**

Initializes a newly created *SequenceInputStream* by remembering the argument, which must be an *Enumeration* that produces object whose run-time type is *InputStream*. The input stream that are produced by the enumeration will be read, in order, to provide the bytes to be read from this *SequenceInputStream*. After each input stream from the enumeration is exhausted, it is closed by calling its *close* method.

**constructor `SequenceInputStream(InputStream first_stream, InputStream second_stream)`:**

Initializes a newly created *SequenceInputStream* by remembering the two arguments, which will be read in order, first *first\_stream* and then *second\_stream*, to provide the bytes to be read from this *SequenceInputStream*.

Package [java.io](#)

**Class [SequenceInputStream](#)****Public Method available**

```
int available()
```

**Overrides:**

- [available](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:**

This method returns the number of bytes available on the current stream.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World testFile\r";
writeToFile(file, fileString);
InputStream in = new FileInputStream(file);

File file2 = new File("testFil2");
String fileString2 = "Hello World testFile-2\r";
writeToFile(file2, fileString2);
InputStream in2 = new FileInputStream(file2);

SequenceInputStream sis = new SequenceInputStream(in, in2);
Logger.log("available data: " + sis.available());
```

The example above returns the number of bytes which are available on the current stream.

Package [java.io](#)

## Class [SequenceInputStream](#)

### Public Method close

```
void close ()
```

#### Overrides:

- [close](#) in class [InputStream](#)

#### Throws:

- [IOException](#)

#### Description:

This method closes this stream and releases any system resources associated with the stream. A closed *SequenceInputStream* cannot perform input operations and cannot be reopened.

If this stream was created from an enumeration, all remaining elements are requested from the enumeration and closed before the *close* method returns.

#### Example

```
File file = new File("testFile");
String fileString = "Hello World testFile\r";
writeToFile(file,fileString);
InputStream in = new FileInputStream(file);

File file2 = new File("testFil2");
String fileString2 = "Hello World testFile-2\r";
writeToFile(file2,fileString2);
InputStream in2 = new FileInputStream(file2);

SequenceInputStream sis = new SequenceInputStream(in, in2);
... sis.close();
```

The example above closes the stream.

Package [java.io](#)

## Class [SequenceInputStream](#)

### Public Method read

```
int read ()
```

#### Overrides:

- [read](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

```
int read(byte[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [InputStream](#)

**Throws:**

- [IOException](#)

**Description:****read() method:**

Reads the next byte of data from this input stream. The byte is returned as an *int* in the range 0 to 255. If no byte is available because the end of the stream is reached, the value -1 is returned. This method blocks until input data is available, the end of the stream is detected, or an exception is thrown.

This method tries to read one character from the current substream. If it reaches the end of the stream, it calls the *close* method of the current substream and begins reading from the next substream.

**read(byte[] buf, int offset, int len) method:**

This method reads up to *len* bytes of data from this input stream into an array of bytes. This method blocks until at least 1 byte of input is available. If the first argument is *null*, up to *len* bytes are read and discarded.

The *read* method of *SequenceInputStream* tries to read the data from the current substream. If it fails to read any characters because the substream has reached the end of the stream, it calls the *close* method of the current substream and begins reading from the next substream.

**Example**

```
File file = new File("testFile");
String fileString = "Hello World testFile\r";
writeToFile(file, fileString);
InputStream in = new FileInputStream(file);

File file2 = new File("testFil2");
String fileString2 = "Hello World testFile-2\r";
writeToFile(file2, fileString2);
InputStream in2 = new FileInputStream(file2);

SequenceInputStream sis = new SequenceInputStream(in, in2);
int data = 0;
....
data = sis.read();
```

The example above reads one byte from this stream.

## Package [java.io](#)

### Interface **Serializable**

Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object. After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

The classes `ObjectInputStream` and `ObjectOutputStream` are high-level streams that contain the methods for serializing and deserializing an object.

Serializability of a class is enabled by the class implementing the `java.io.Serializable` interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

## Package [java.io](#)

### Class **StreamCorruptedException**

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

implements [Serializable](#)

Thrown when control information that was read from an object stream violates internal consistency checks.

#### Public Constructors

- [StreamCorruptedException](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [StreamCorruptedException](#)

### Public Constructor StreamCorruptedException

```
StreamCorruptedException ()
```

```
StreamCorruptedException (String message)
```

#### Description:

##### constructor `StreamCorruptedException()`:

Create Exception null as its detailed message.

##### constructor `StreamCorruptedException(String message)`:

Create Exception with detailed message.

Package [java.io](#)

## Class `StreamTokenizer`

extends [Object](#)

The `StreamTokenizer` class takes an input stream and parse it into tokens, allowing the tokens to be read one at a time. The parsing process is controlled by a table and a number of flags that can be set to various states. The stream tokenizer can recognize *identifiers*, *numbers*, *quoted strings*, and various *comment styles*.

Valid input values for the stream tokenizer are between `'\u0000'` and `'\u00FF'`. The input values are regarded as characters with five possible attributes: *whitespace*, *alphabetic*, *numeric*, *string quote*, and *comment character*. Each character can have zero or more of these attributes.

Instances of the string tokenizer can have the following flags:

- Line terminators are to be returned as tokens or treated as whitespace that merely separates tokens.
- C-style comments are to be recognized and skipped.
- C++-style comments are to be recognized and skipped.
- Characters of identifiers are converted to lowercase.

### Public Constructors

- [StreamTokenizer](#)

### Public Methods

- [commentChar](#)
- [eolIsSignificant](#)
- [lineno](#)
- [lowerCaseMode](#)
- [nextToken](#)
- [ordinaryChar](#)

- [ordinaryChars](#)
- [parseNumbers](#)
- [pushBack](#)
- [quoteChar](#)
- [reset](#)
- [resetSyntax](#)
- [slashSlashComments](#)
- [slashStarComments](#)
- [toString](#)
- [whitespaceChars](#)
- [wordChars](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- float nval
- String sval
- static final int TT\_EOF
- static final int TT\_EOL
- static final int TT\_NUMBER
- static final int TT\_WORD
- int ttype

### Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Constructor [StreamTokenizer](#)

```
StreamTokenizer(InputStream i)
```

```
StreamTokenizer(Reader r)
```

### Description:

#### constructor [StreamTokenizer\(InputStream i\)](#):

This method constructs a stream tokenizer that parses the specified input stream. The stream tokenizer is initialized to the following default state:

- All byte values 'A' through 'Z', 'a' through 'z', and '\u00A0' through '\u00FF' are considered to be alphabetic.
- All byte values '\u0000' through '\u0020' are considered to be whitespace.
- '/' is a comment character.
- Single quote '\'' and double quote '"' are string quote characters.
- Numbers are parsed.
- Ends of lines are treated as whitespace, not as separate tokens.
- C-style and C++-style comments are not recognized.



**constructor StreamTokenizer(Reader r):**

Constructs a tokenizer that parses the given character stream.

Package [java.io](#)

**Class [StreamTokenizer](#)****Public Method [commentChar](#)**

```
void commentChar(int ch)
```

**Description:**

Specified that the character argument starts a single-line comment. All characters from the comment character to the end of the line are ignored by this stream tokenizer.

**Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
st.commentChar('a');
```

The example above reads a file from the MicroSD card as a input stream and sets up the stream tokenizer *commentChar* 'a' which will be skipped when the file is read.

Package [java.io](#)

**Class [StreamTokenizer](#)****Public Method [eolIsSignificant](#)**

```
void eolIsSignificant(boolean flag)
```

**Description:**

Determines whether or not ends of line are treated as tokens. If the flag argument is true, this tokenizer treats end-of-lines as tokens. The *nextToken* method returns *TT\_EOL* and also sets the *ttype* field to this value when an end of line is read.

A line is a sequence of characters ending with either a carriage-return character (`'\r'`) or a newline character (`'\n'`). In addition, a carriage-return character followed immediately by a newline character is treated as a single end-of-line token.

If the *flag* is false, end-of-line characters are treated as whitespace and serve only to separate tokens.

**Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
st.eolIsSignificant(true);
```

All end of line characters are treated as token with the *eolIsSignificant* method.

Package [java.io](#)  
Class [StreamTokenizer](#)

**Public Method `lineno`**

```
int lineno()
```

**Description:**

This method returns the current line number.

**Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
Logger.log("line number: " + st.lineno());
```

The example above reads the line number and writes the current line number to the logger.

Package [java.io](#)  
Class [StreamTokenizer](#)

**Public Method `lowerCaseMode`**

```
void lowerCaseMode(boolean fl)
```

**Description:**

The *lowerCaseMode* method determines whether or not word token are automatically lowercase.

**Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
st.lowerCaseMode(true);
```

All following word token are treated as lowercase.

Package [java.io](#)  
Class [StreamTokenizer](#)

**Public Method `nextToken`**

```
int nextToken()
```

**Throws:**

- [IOException](#)

**Description:**

This method parses the next token from the input stream of this tokenizer. The type of the next token is returned in the *ttype* field. Additional information about the token may be in the *nval* field or the *sval* field of this tokenizer. Typical clients of this class first set up the syntax tables and then sit in a loop calling *nextToken* to parse successive tokens until *TT\_EOF* is returned.

**Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
int token = st.nextToken();
```

The example above parses the next token from the input stream.

Package [java.io](#)

**Class [StreamTokenizer](#)****Public Method [ordinaryChar](#)**

```
void ordinaryChar(int ch)
```

**Description:**

Specifies that the character argument is 'ordinary' in this tokenizer. It removes any special significance the character has as a comment character, word component, string delimiter, whitespace, or number character. When such a character is encountered by the parser, the parser treats it as a single-character token and sets *ttype* field to the character value.

**Example**

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.ordinaryChar('h');
```

The character which is specified by the *ordinaryChar* method is treated as token.

Package [java.io](#)

**Class [StreamTokenizer](#)****Public Method [ordinaryChars](#)**

```
void ordinaryChars(int low, int hi)
```

**Description:**

Specifies that all characters *c* in the range  $low \leq c \leq high$  are 'ordinary' in this tokenizer.

## Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.ordinaryChars('a', 'b');
```

The character which are specified by the *ordinaryChars* method are treated as token.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method parseNumbers

```
void parseNumbers()
```

#### Description:

Specifies that numbers should be parsed by this tokenizer. The syntax table of this tokenizer is modified so that each of the following characters has the 'numeric' attribute.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
st.parseNumbers();
```

All numbers are parsed by this tokenizer with the example above.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method pushBack

```
void pushBack()
```

#### Description:

Causes the next call to the *nextToken* method of this tokenizer to return the current value in the *ttype* field.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.pushBack();
```

The example above causes the next call to the *nextToken* method to return the current value in the *ttype* field.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method quoteChar

```
void quoteChar(int ch)
```

#### Description:

Specifies that matching pairs of this character delimited string contains in this tokenizer.

When the *nextToken* method encounters a string constant, the *ttype* field is set to the string delimiter and the *sval* field is set to the body of the string.

If a string quote character is encountered, then a string is recognized, consisting of all characters after the string quote character, up to the next occurrence of that same string quote character, or a line terminator, or end of file. The usual escape sequence `'\t'` is recognized and converted to single character as the string is parsed.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.quoteChar('a');
```

The example above defines the `'a'` character as string quote character.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method reset

```
void reset()
```

#### Description:

Resets this stream tokenizer to it's default values.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method resetSyntax

```
void resetSyntax()
```

#### Description:

Resets this tokenizer's syntax table so that all characters are 'ordinary'. See the *ordinaryChar* method for more information on a character being ordinary.

## Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.resetSyntax();
```

All characters are treated as 'ordinary' character after the `resetSyntax` method call.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method `slashSlashComments`

```
void slashSlashComments(boolean flag)
```

#### Description:

This method determines whether or not the tokenizer recognizes C++-style comments. If the flag argument is `true`, this stream tokenizer recognizes C++-style comments. Any occurrence of two consecutive slash characters (`'/'`) is treated as the beginning of a comment that extends to the end of line. If the flag argument is `false`, then C++-style comments are not treated specially.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.slashSlashComments(true);
```

The example above sets up the stream tokenizer to recognize C++-style comments.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method `slashStarComments`

```
void slashStarComments(boolean flag)
```

#### Description:

This method determines whether or not the tokenizer recognizes C-style comments. If the flag argument is `true`, this stream tokenizer recognizes C-style comments. All text between successive occurrences of `/*` and `*/` are discarded. If the flag argument is `false`, then C-style comments are not treated specially.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
```

```
...
st.slashStarComments(true);
```

The example above sets up the stream tokenizer to recognize C-style comments.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method `toString`

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

#### Description:

This method returns the string representation of the current stream token.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
Logger.log(st.toString());
```

The example above writes the string representation of the current stream token to the logger.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method `whitespaceChars`

```
void whitespaceChars(int low, int hi)
```

#### Description:

This method specifies that all characters  $c$  in the range  $low \leq c \leq high$  are word constituents. A word token consists of a word constituent followed by zero or more constituents or number constituents.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.whitespaceChars('a', 'c');
```

All characters between 'a' and 'c' characters and also the 'a' and 'c' characters regarded as word constituents after the `whitespaceChars` method call.

Package [java.io](#)

## Class [StreamTokenizer](#)

### Public Method [wordChars](#)

```
void wordChars(int low, int hi)
```

#### Description:

This method specifies that all characters  $c$  in the range  $low \leq c \leq high$  are word constituents. A word token consists of a word constituent followed by zero or more word constituents or number constituents.

#### Example

```
File file = new File("testFile");
InputStream is = new FileInputStream(file);
StreamTokenizer st = new StreamTokenizer(is);
...
st.wordChars('a', 'c');
```

All characters between 'a' and 'c' characters and also the 'a' and 'c' characters regarded as word constituents after the `wordChars` method call.

Package [java.io](#)

## Class [StringBufferInputStream](#)

extends [InputStream](#) → [Object](#)

This class allows an application to create an input stream in which the bytes read are supplied by the contents of a string.

### Public Constructors

- [StringBufferInputStream](#)

### Public Methods

- [available](#)
- [read](#)
- [reset](#)
- [skip](#)

### Methods inherited from [java.io.InputStream](#)

- [mark](#)
- [markSupported](#)
- [reset](#)
- [available](#)
- [skip](#)
- [read](#)
- [close](#)



**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [StringBufferInputStream](#)****Public Constructor StringBufferInputStream**

```
StringBufferInputStream(String s)
```

**Description:**

This method constructs a string input stream to read data from the specified string.

**Package [java.io](#)****Class [StringBufferInputStream](#)****Public Method available**

```
int available()
```

**Overrides:**

- [available](#) in class [InputStream](#)

**Description:**

Returns the number of bytes that can be read from the input stream without blocking.

**Example**

```
String str = "Hello World";
StringBufferInputStream sbis = new StringBufferInputStream(str);

Logger.log("Number of bytes that can be read without blocking: " +
sbis.available());
```

The example above counts the number of bytes that can be read without blocking and writes the result to the logger.

**Package [java.io](#)****Class [StringBufferInputStream](#)****Public Method read**

```
int read()
```

**Overrides:**

- [read](#) in class [InputStream](#)

```
int read(byte[] buf, int offset, int len)
```

**Overrides:**

- [read](#) in class [InputStream](#)

**Description:****read() method:**

This method reads the next byte of data from this input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.

The *read* method of *StringBufferedInputStream* cannot block. It returns the low eight bits of the next character in this input stream's buffer.

**read(byte[] buf, int offset, int len) method:**

This method reads up to *len* bytes of data from this input stream into an array of bytes.

The *read* method of *StringBufferedInputStream* cannot block. It copies the low eight bits from the characters in this input stream's buffer into byte array argument.

**Example**

```
String str = "Hello World";
StringBufferInputStream sbis = new StringBufferInputStream(str);
byte[] b = new byte[5];
sbis.read(b, 0, 5);
Console.print("input data: ");
for (int i = 0; i < 5; i++)
    Console.print("" + (char)b[i]);
```

The example above reads the first 5 elements of the string *str* and writes it to the console.

**Package [java.io](#)****Class [StringBufferInputStream](#)****Public Method reset**

```
void reset()
```

**Overrides:**

- [reset](#) in class [InputStream](#)

**Description:**

This method resets the input stream to begin reading from the first character of this input stream's underlying buffer.

**Example**

```
String str = "Hello World";
StringBufferInputStream sbis = new StringBufferInputStream(str);

Console.println("read the first element of the string: " +
sbis.read());
Console.println("read the second element of the string: " +
sbis.read());
sbis.reset();
Console.println("read the first element of the string: " +
sbis.read());
```

The example above reads the first two elements of the String, resets the stream to the beginning of the string, reads only the first element again, and prints it to the console.

Package [java.io](#)

**Class [StringBufferInputStream](#)****Public Method [skip](#)**

```
int skip(int num_bytes)
```

**Overrides:**

- [skip](#) in class [InputStream](#)

**Description:**

This method skips *num\_bytes* bytes of input from this input stream. Fewer bytes might be skipped if the end of the input stream is reached.

**Example**

```
String str = "Hello World";
StringBufferInputStream sbis = new StringBufferInputStream(str);

sbis.skip(6);
Console.println("six elements are skipped: " + sbis.read());
```

The example above skips 6 elements of the string *str*, reads the seventh element, and prints it to the console.

Package [java.io](#)

**Class [StringReader](#)**

extends [Reader](#) → [Object](#)

The *StringReader* class provides a character stream whose source is a string.

### Public Constructors

- [StringReader](#)

### Public Methods

- [close](#)
- [mark](#)
- [markSupported](#)
- [read](#)
- [ready](#)
- [reset](#)
- [skip](#)

### Methods inherited from [java.io.Reader](#)

- [read](#)
- [close](#)
- [markSupported](#)
- [mark](#)
- [reset](#)
- [ready](#)
- [skip](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [StringReader](#)

### Public Constructor [StringReader](#)

```
StringReader(String buffer)
```

#### Description:

This method constructs a *StringReader* object.

Package [java.io](#)

## Class [StringReader](#)

### Public Method [close](#)

```
void close()
```

**Overrides:**

- [close](#) in class [Reader](#)

**Description:**

This method closes this string reader.

**Example**

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

sr.close();
```

The `close` method closes the string reader.

Package [java.io](#)

**Class [StringReader](#)****Public Method mark**

```
void mark(int readAheadLimit)
```

**Overrides:**

- [mark](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method mark the present position in the stream. Subsequent calls to `reset` will reposition the stream to this point.

**Example**

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

int dim = 5;
char[] b = new char[dim];

sr.reset();
sr.read(b, 0, dim);
for (int i = 0; i < dim; i++)
    Console.print("" + b[i]);
Console.println("");

if (sr.markSupported())
    sr.mark(5);
```

```
sr.reset();
sr.read(b, 0, dim);
for (int i = 0; i < dim; i++)
    Console.print("" + b[i]);
Console.println("");

sr.close();
```

A new position is assigned in the stream with the mark method.

Package [java.io](#)

## Class [StringReader](#)

### **Public Method markSupported**

```
boolean markSupported()
```

#### Overrides:

- [markSupported](#) in class [Reader](#)

#### Description:

This method checks if mark operation is supported.

#### Example

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

if (sr.markSupported())
{
    Logger.log("This stream supports the mark operation!");
    sr.mark(5);
}
```

The *markSupported* method checks if the mark operation is supported.

Package [java.io](#)

## Class [StringReader](#)

### **Public Method read**

```
int read()
```

#### Overrides:

- [read](#) in class [Reader](#)

#### Throws:

- [IOException](#)

```
int read(char[] b, int off, int len)
```

**Overrides:**

- [read](#) in class [Reader](#)

**Throws:**

- [IOException](#)
- [ArrayIndexOutOfBoundsException](#)

**Description:****read() method:**

The *read* method reads a single character.

**read(byte[] buf, int off, int len) method:**

This method reads characters into a portion of an array.

**Example**

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

int dim = 6;
char[] b = new char[dim];
sr.read(b, 0, dim);
for (int i = 0; i < dim; i++)
    Console.print("" + b[i]);
Console.println("");

sr.close();
```

The example above reads six characters from the stream and prints it to the console.

Package [java.io](#)

**Class [StringReader](#)****Public Method ready**

```
boolean ready()
```

**Overrides:**

- [ready](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method tell whether this stream is ready to be read.

**Example**

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

int dim = 6;
char[] b = new char[dim];

if (sr.ready())
    Console.println("number of characters: " + sr.read(b, 0, dim));
else
    Console.println("stream is not ready");
```

The example checks if the stream is ready.

Package [java.io](#)

**Class [StringReader](#)****Public Method reset**

```
void reset()
```

**Overrides:**

- [reset](#) in class [Reader](#)

**Throws:**

- [IOException](#)

**Description:**

This method resets the stream to the most recent mark, or to the beginning of the stream if it has never been marked.

**Example**

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

sr.mark(5);
sr.reset();
Console.println("" + sr.read());

sr.close();
```

A new position is assigned in the stream with the *mark* method after the *reset* method call.



Package [java.io](#)

## Class [StringReader](#)

### Public Method [skip](#)

```
int skip(int n)
```

#### Overrides:

- [skip](#) in class [Reader](#)

#### Throws:

- [IOException](#)

#### Description:

This method skips the specified characters.

#### Example

```
String buffer = "Hello World";
StringReader sr = new StringReader(buffer);

sr.skip(6);
if (sr.read() == 'W')
    Console.println("skip operation was successfully");

sr.close();
```

The example above skips 6 characters.

Package [java.io](#)

## Class [StringWriter](#)

extends [Writer](#) → [Object](#)

The *StringWriter* class provides a character stream that collects its output in a string buffer, which can then be used to construct a string.

### Public Constructors

- [StringWriter](#)

### Public Methods

- [close](#)
- [flush](#)
- [getBuffer](#)
- [toString](#)
- [write](#)

**Methods inherited from [java.io.Writer](#)**

- [flush](#)
- [close](#)
- [write](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [StringWriter](#)****Public Constructor StringWriter**

```
StringWriter()
```

```
StringWriter(int size)
```

**Description:****constructor [StringWriter\(\)](#):**

This method constructs a new string writer, using the default initial string-buffered size.

**constructor [StringWriter\(int size\)](#):**

This method constructs a new string writer, using the specified size for the internal buffer.

**Package [java.io](#)****Class [StringWriter](#)****Public Method close**

```
void close()
```

**Overrides:**

- [close](#) in class [Writer](#)

**Description:**

This method closes this string reader.

**Example**

```
StringWriter sw = new StringWriter();  
  
sw.close();
```

The `close` method closes the string writer.

Package [java.io](#)

Class [StringWriter](#)

**Public Method `flush`**

```
void flush()
```

**Overrides:**

- [flush](#) in class [Writer](#)

**Description:**

This method flushes the string stream

**Example**

```
StringWriter sw = new StringWriter();  
...  
sw.flush();
```

The example above flushes the string writer stream.

Package [java.io](#)

Class [StringWriter](#)

**Public Method `getBuffer`**

```
StringBuffer getBuffer()
```

**Description:**

This method returns the current string buffer value.

**Example**

```
StringWriter sw = new StringWriter();  
...  
sw.getBuffer();
```

The example above returns the current buffer value.

Package [java.io](#)

Class [StringWriter](#)

**Public Method `toString`**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

This method returns the buffer's current value as a string.

**Example**

```
StringWriter sw = new StringWriter ();  
sw.write("Hello World");  
Logger.log(sw.toString());
```

The example above writes the buffer's current value to the logger.

Package [java.io](#)

**Class [StringWriter](#)****Public Method write**

```
void write(char[] buf, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(int c)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(String str)
```

**Overrides:**

- [write](#) in class [Writer](#)

```
void write(String str, int offset, int len)
```

**Overrides:**

- [write](#) in class [Writer](#)

**Description:****write(char[] buf, int offset, int len) method:**

This method writes a portion of an array of characters.

**write(int c) method:**

This method writes a single character.

**write(String str) method:**

This method writes a whole string.

**write(String str, int offset, int len) method:**

This method is used to write a portion of a string.

**Example**

```
StringWriter sw = new StringWriter();
char[] ch = {10,20};
sw.write(ch, 0, 2);
```

The example above writes the character array to the String output stream.

**Package [java.io](#)****Class [SyncFailedException](#)**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that a sync operation has failed.

**Public Constructors**

- [SyncFailedException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [SyncFailedException](#)****Public Constructor [SyncFailedException](#)**

```
SyncFailedException()
```

```
SyncFailedException(String message)
```

**Description:****constructor SyncFailedException():**

Create Exception null as its detailed message.

**constructor SyncFailedException(String message):**

Create Exception with detailed message.

**Package [java.io](#)****Class [UnsupportedEncodingException](#)**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that the character encoding is not supported.

**Public Constructors**

- [UnsupportedEncodingException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [UnsupportedEncodingException](#)****Public Constructor [UnsupportedEncodingException](#)**

```
UnsupportedEncodingException ()
```

```
UnsupportedEncodingException (String message)
```

**Description:****constructor [UnsupportedEncodingException](#)():**

Create Exception null as its detailed message.

**constructor `UnsupportedEncodingException(String message)`:**

Create Exception with detailed message.

**Package [java.io](#)****Class `UTFDataFormatException`**

extends [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that a malformed UTF-8 string has been read in a data input stream or by any class that implements the data input interface. See the `writeUTF` method for the format in which UTF-8 strings are read and written.

**Public Constructors**

- [UTFDataFormatException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.io](#)****Class [UTFDataFormatException](#)****Public Constructor `UTFDataFormatException`**

```
UTFDataFormatException()
```

```
UTFDataFormatException(String message)
```

**Description:****constructor `UTFDataFormatException()`:**

Create Exception null as its detailed message.

**constructor `UTFDataFormatException(String message)`:**

Create Exception with detailed message.

## Package [java.io](#)

### Class **WriteAbortedException**

extends [ObjectStreamException](#) → [IOException](#) → [Exception](#) → [Throwable](#) → [Object](#)  
implements [Serializable](#)

Signals that one of the ObjectStreamExceptions was thrown during a write operation.

#### Public Constructors

- [WriteAbortedException](#)

#### Public Methods

- [getMessage](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- Exception detail

## Package [java.io](#)

### Class **[WriteAbortedException](#)**

#### **Public Constructor WriteAbortedException**

```
WriteAbortedException(String msg, Exception detail)
```

#### **Description:**

Thrown during a read operation when one of the ObjectStreamExceptions was thrown during a write operation. The exception that terminated the write operation can be found in the detail field. The stream is reset to it's initial state and all references to objects already deserialized are discarded.



Package [java.io](#)

## Class [WriteAbortedException](#)

### Public Method [getMessage](#)

```
String getMessage()
```

#### Overrides:

- [getMessage](#) in class [Throwable](#)

#### Description:

Produce the message and include the message from the nested exception, if there is one.

Package [java.io](#)

## Abstract Class [Writer](#)

extends [Object](#)

This is a abstract class for writing to character streams.

#### Public Methods

- [close](#)
- [flush](#)
- [write](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.io](#)

## Class [Writer](#)

### Public Method [close](#)

```
abstract void close()
```

#### Throws:

- [IOException](#)

#### Description:

This method closes this stream and flushing it first.

Package [java.io](#)  
Class [Writer](#)

### Public Method flush

```
abstract void flush()
```

#### Throws:

- [IOException](#)

#### Description:

This method flushs the stream.

Package [java.io](#)  
Class [Writer](#)

### Public Method write

```
void write(char[] buf)
```

#### Throws:

- [IOException](#)

```
abstract void write(char[] buf, int offset, int len)
```

#### Throws:

- [IOException](#)

```
void write(int b)
```

#### Throws:

- [IOException](#)

```
void write(String str)
```

#### Throws:

- [IOException](#)

```
void write(String str, int offset, int len)
```

#### Throws:

- [IOException](#)

**Description:****write(char []buf) method:**

This method writes an array of character.

**write(char[] buf, int offset, int len) method:**

This method writes a portion of an array of characters.

**write(int b) method:**

This method writes a single character.

**write(String str) method:**

This method writes a whole string.

**write(String str, int offset, int len) method:**

This method is used to write a portion of a string.

**Package java.lang**

Contains fundamental classes considered an essential part of the Java Language.

**Interfaces**

- [Cloneable](#)
- [Comparable](#)
- [Runnable](#)

**Classes**

- [AbstractMethodError](#)
- [ArithmeticException](#)
- [ArrayIndexOutOfBoundsException](#)
- [ArrayStoreException](#)
- [Boolean](#)
- [Byte](#)
- [Character](#)
- [ClassCastException](#)
- [ClassNotFoundException](#)
- [CloneNotSupportedException](#)
- [Error](#)
- [Exception](#)
- [Float](#)
- [IllegalArgumentException](#)
- [IllegalMonitorStateException](#)
- [IllegalStateException](#)
- [IncompatibleClassChangeError](#)
- [IndexOutOfBoundsException](#)
- [Integer](#)
- [InternalError](#)
- [InterruptedException](#)

- [LinkageError](#)
- [Math](#)
- [NegativeArraySizeException](#)
- [NoClassDefFoundError](#)
- [NoSuchFieldError](#)
- [NoSuchMethodError](#)
- [NullPointerException](#)
- [Number](#)
- [NumberFormatException](#)
- [Object](#)
- [OutOfMemoryError](#)
- [RuntimeException](#)
- [SecurityException](#)
- [Short](#)
- [StackOverflowError](#)
- [String](#)
- [StringBuffer](#)
- [StringIndexOutOfBoundsException](#)
- [System](#)
- [Thread](#)
- [Throwable](#)
- [UnsupportedOperationException](#)
- [VerifyError](#)
- [VirtualMachineError](#)

## Package [java.lang](#)

### Class **AbstractMethodError**

extends [IncompatibleClassChangeError](#) → [LinkageError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown when an application tries to call an abstract method. Normally, this error is caught by the compiler. It can only occur at run time if the definition of a class has changed since the current method compiled.

#### Public Constructors

- [AbstractMethodError](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [AbstractMethodError](#)

### Public Constructor AbstractMethodError

```
AbstractMethodError()
```

```
AbstractMethodError(String s)
```

#### Description:

##### constructor [AbstractMethodError\(\)](#):

Constructs an [AbstractMethodError](#) with no detail message.

##### constructor [AbstractMethodError\(String s\)](#):

Constructs an [AbstractMethodError](#) with the specified detail message.

Package [java.lang](#)

## Class [ArithmeticException](#)

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown when an exceptional arithmetic condition (e.g. division by zero) has occurred.

### Public Constructors

- [ArithmeticException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [ArithmeticException](#)

### Public Constructor ArithmeticException

```
ArithmeticException()
```

```
ArithmeticException(String s)
```

**Description:****constructor ArithmeticException():**

Constructs an ArithmeticException with no detail message.

**constructor ArithmeticException(String s):**

Constructs an ArithmeticException with the specified detail message.

Package [java.lang](#)

**Class ArrayIndexOutOfBoundsException**

extends [IndexOutOfBoundsException](#) → [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

**Public Constructors**

- [ArrayIndexOutOfBoundsException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

**Class [ArrayIndexOutOfBoundsException](#)****Public Constructor [ArrayIndexOutOfBoundsException](#)**

```
ArrayIndexOutOfBoundsException()
```

```
ArrayIndexOutOfBoundsException(int index)
```

```
ArrayIndexOutOfBoundsException(String s)
```

**Description:****constructor `ArrayIndexOutOfBoundsException()`:**

Constructs an `ArrayIndexOutOfBoundsException` with no detail message.

**constructor `ArrayIndexOutOfBoundsException(int index)`:**

Constructs a new `ArrayIndexOutOfBoundsException` class with an argument indicating the illegal index.

**constructor `ArrayIndexOutOfBoundsException(String s)`:**

Constructs an `ArrayIndexOutOfBoundsException` with the specified detail message.

**Package [java.lang](#)****Class `ArrayStoreException`**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.

**Public Constructors**

- [ArrayStoreException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [ArrayStoreException](#)****Public Constructor `ArrayStoreException`**

```
ArrayStoreException()
```

```
ArrayStoreException(String s)
```

**Description:****constructor `ArrayStoreException()`:**

Constructs an `ArrayStoreException` with no detail message.

**constructor `ArrayStoreException(String s)`:**

Constructs an `ArrayStoreException` with the specified detail message.

**Package [java.lang](#)****Class `Boolean`**

extends [Object](#)

The `Boolean` class wraps a value of the primitive type `boolean` in an object. An object of type `Boolean` contains a single field whose type is `boolean`.

**Public Constructors**

- [Boolean](#)

**Public Methods**

- [booleanValue](#)
- [equals](#)
- [hashCode](#)
- [toString](#)
- [valueOf](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Public Fields**

- static final Boolean FALSE
- static final Boolean TRUE

**Package [java.lang](#)****Class [Boolean](#)****Public Constructor `Boolean`**

```
Boolean(boolean value)
```

```
Boolean(String s)
```



**Description:****constructor Boolean(boolean value):**

Creates a new *Boolean* object which represents the value argument.

**constructor Boolean(String s):**

Allocates a *Boolean* object representing the value true if the string argument is not null and is equal, ignoring case, to the string "true". Otherwise, allocate a *Boolean* object representing the value false.

**Example**

```
Boolean bool = new Boolean("true");
```

A new *Boolean* object is created with the example above.

Package [java.lang](#)

Class [Boolean](#)

**Public Method booleanValue**

```
boolean booleanValue()
```

**Description:**

Returns the value of this *Boolean* object as a *boolean* primitive.

**Example**

```
Boolean bool = new Boolean("true");
if (bool.booleanValue())
    Logger.log("bool value is ture");
```

The example above returns a byte value and writes it to the logger.

Package [java.lang](#)

Class [Boolean](#)

**Public Method equals**

```
boolean equals(Object obj)
```

**Overrides:**

- [equals](#) in class [Object](#)

**Description:**

Returns *true* if and only if the argument is not *null* and is a *Boolean* object that represents the same *boolean* value as this object.

## Example

```
Boolean eqA = new Boolean("true");
Boolean eqB = new Boolean("true");
if (eqA.equals(eqB))
    Logger.log("correct: A eq B");
```

The comparison between *eqA* and *eqB* delivers a true result.

Package [java.lang](#)

Class [Boolean](#)

### Public Method hashCode

```
int hashCode()
```

#### Overrides:

- [hashCode](#) in class [Object](#)

#### Description:

Returns a *hashcode* for this *Boolean*.

#### Example

```
Boolean bool = new Boolean("10");
int hashCode = bool.hashCode();
Logger.log("hashcode: " + hashCode);
```

The example above returns the hashcode value and writes it to the logger.

Package [java.lang](#)

Class [Boolean](#)

### Public Method toString

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

#### Description:

Returns a *String* object representing this *Booleans*'s value. If this object represents the value true, a string equal to "*true*" is returned. Otherwise, a string equal to "*false*" is returned.

#### Example

```
Boolean bool = new Boolean("10");    Logger.log(bool.toString());
```

The example above returns the boolean value converted to a string.

## Package [java.lang](#)

### Class [Boolean](#)

#### Public Method `valueOf`

```
static Boolean valueOf(String s)
```

#### Description:

Returns a *Boolean* with a value represented by the specified *String*. The *Boolean* returned represents the value true if the string argument is not null and is equal, ignoring case, to the string "true".

#### Example

```
Boolean bool = Boolean.valueOf("true");
```

The `valueOf()` method returns the *Boolean* representation by the string.

## Package [java.lang](#)

### Class `Byte`

extends [Number](#) → [Object](#)

This class wraps a value of the primitive byte type in an object. An object of type *Byte* contains a single field whose type is byte.

#### Public Constructors

- [Byte](#)

#### Public Methods

- [byteValue](#)
- [decode](#)
- [equals](#)
- [floatValue](#)
- [hashCode](#)
- [intValue](#)
- [parseByte](#)
- [shortValue](#)
- [toString](#)
- [valueOf](#)

#### Methods inherited from [java.lang.Number](#)

- [byteValue](#)
- [floatValue](#)
- [intValue](#)
- [shortValue](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)

- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Public Fields

- static final byte MAX\_VALUE
- static final byte MIN\_VALUE

Package [java.lang](#)

## Class [Byte](#)

### Public Constructor [Byte](#)

```
Byte(byte value)
```

```
Byte(String s)
```

### Throws:

- [NumberFormatException](#)

### Description:

#### constructor [Byte](#)(byte value):

This constructs a newly allocated *Byte* object that represents the specified byte value.

#### constructor [Byte](#)(String s):

This constructs a newly allocated *Byte* object that represents the byte value indicated by the *String* parameter.

Note that number literals are integers, so "byte" - literals have to be cast to byte, as shown in the example below.

### Example

```
byte[] bytes = {(byte)0xE2, (byte)0x88, (byte)0x80}; // cast literals to  
byte
```

Package [java.lang](#)

## Class [Byte](#)

### Public Method [byteValue](#)

```
byte byteValue()
```

### Overrides:

- [byteValue](#) in class [Number](#)

**Description:**

Returns the value of this *Byte* as a *byte*.

**Example**

```
Byte by = new Byte("20");
byte byteby = by.byteValue();
Logger.log("byte value: " + byteby);
```

The example above returns a byte value and writes it to the logger.

Package [java.lang](#)

Class [Byte](#)

**Public Method decode**

```
static Byte decode(String nm)
```

**Throws:**

- [NumberFormatException](#)

**Description:**

This method decodes a *String* into a *Byte*. It accepts decimal, hexadecimal, and octal number in the following formats:

- - decimal constant
- 0x - hexadecimal constant
- # - hexadecimal constant
- 0 - octal constant

The constant following an (optional) negative sign and/or "radix specifier" is parsed as by the *Byte.parseByte* method with the specified radix (10, 8 or 16). This constant must be positive or a *NumberFormatException* will result. The result is made negative if first character of the specified *String* is the negative sign. No whitespace characters are permitted in the *String*.

**Example**

```
Byte data = Byte.decode("20");
```

The example above decodes a string to a Byte number.

Package [java.lang](#)

Class [Byte](#)

**Public Method equals**

```
boolean equals(Object obj)
```

**Overrides:**

- [equals](#) in class [Object](#)

**Description:**

This method compares this object to the specified object. The result is true if and only if the argument is not null and is an *Byte* object that contains the same byte value as this object.

**Example**

```
Byte eqA = new Byte("10");
Byte eqB = new Byte("10");
if (eqA.equals(eqB))
    Logger.log("correct: A eq B");
```

The comparison between *eqA* and *eqB* delivers a true result.

Package [java.lang](#)

Class [Byte](#)

**Public Method floatValue**

```
float floatValue()
```

**Overrides:**

- [floatValue](#) in class [Number](#)

**Description:**

Returns the value of this *Byte* as a *float*.

**Example**

```
Byte by = new Byte("10");
float floatByte = by.floatValue();
Logger.log("float value: " + floatByte);
```

The example above returns a float value and writes it to the logger.

Package [java.lang](#)

Class [Byte](#)

**Public Method hashCode**

```
int hashCode()
```

**Overrides:**

- [hashCode](#) in class [Object](#)

**Description:**

Returns a *hashcode* for this *Byte*.

## Example

```
Byte by = new Byte("10");
int hashCode = by.hashCode();
Logger.log("hashcode: " + hashCode);
```

The example above returns the hashcode value and writes it to the logger.

Package [java.lang](#)

Class [Byte](#)

### Public Method intValue

```
int intValue()
```

#### Overrides:

- [intValue](#) in class [Number](#)

#### Description:

Returns an *int* value of this *Byte*.

## Example

```
Byte by = new Byte("10");
int intvalue = by.intValue();
Logger.log("intvalue: " + intvalue);
```

The example above returns the int value and writes it to the logger.

Package [java.lang](#)

Class [Byte](#)

### Public Method parseByte

```
static byte parseByte(String s)
```

#### Throws:

- [NumberFormatException](#)

```
static byte parseByte(String s, int radix)
```

#### Throws:

- [NumberFormatException](#)

**Description:****parseByte(String s) method:**

Assuming the specified *String* represents a byte, returns that byte's value. Throws an exception if the *String* cannot be parsed as a byte. The radix is assumed to be 10.

**parseByte(String s, int radix) method:**

Assuming the specified *String* represents a byte, returns that byte's value. Throws an exception if the *String* cannot be parsed as a byte.

**Example**

```
Logger.log("String to byte value: " + Byte.parseByte("23", 10));
Logger.log("Hexadecimal to byte value: " + Byte.parseByte("23", 16));
Logger.log("Octal to byte value: " + Byte.parseByte("23", 8));
```

The example above writes different byte values to the logger.

Package [java.lang](#)

Class [Byte](#)

**Public Method shortValue**

```
short shortValue()
```

**Overrides:**

- [shortValue](#) in class [Number](#)

**Description:**

Returns the value of this *Byte* as a short.

**Example**

```
Byte by = new Byte("10");
short shortByte = by.shortValue();
Logger.log("hash code: " + shortByte);
```

The example above returns the value in short data type and writes it to the logger.

Package [java.lang](#)

Class [Byte](#)

**Public Method toString**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)



```
static String toString(byte s)
```

**Description:****toString() method:**

Returns a *String* object representing this *Byte*'s value.

**toString(byte s) method:**

Returns a new *String* object representing the specified *Byte*. The radix is assumed to be 10.

**Example**

```
Logger.log(Byte.toString("20"));
```

The example above returns the byte value converted to a string.

Package [java.lang](#)

Class [Byte](#)

**Public Method valueOf**

```
static Byte valueOf(String s)
```

**Throws:**

- [NumberFormatException](#)

```
static Byte valueOf(String s, int radix)
```

**Throws:**

- [NumberFormatException](#)

**Description:****valueOf(String s) method:**

Assuming the specified *String* represents a byte, returns a new *Byte* object initialized to that value. Throws an exception if the *String* cannot be parsed as a byte. The radix is assumed to be 10.

**valueOf(String s, int radix) method:**

Assuming the specified *String* represents a byte, returns a new *Byte* object initialized to that value. Throws an exception if the *String* cannot be parsed as a byte.

**Example**

```
Byte by = Byte.valueOf("9");
```

The *valueOf()* method returns the *Byte* representation of the value 10.

## Package [java.lang](#)

### Class `Character`

extends [Object](#)

implements [Serializable](#), [Comparable](#)

The `Character` class wraps a value of the primitive type `char` in an object. An object of type `Character` contains a single field whose type is `char`.

#### Public Constructors

- [Character](#)

#### Public Methods

- [charValue](#)
- [compareTo](#)
- [digit](#)
- [equals](#)
- [forDigit](#)
- [getNumericValue](#)
- [getType](#)
- [hashCode](#)
- [isDigit](#)
- [isIdentifierIgnored](#)
- [isISOControl](#)
- [isJavaIdentifierPart](#)
- [isJavaIdentifierStart](#)
- [isJavaLetter](#)
- [isJavaLetterOrDigit](#)
- [isLetter](#)
- [isLetterOrDigit](#)
- [isLowerCase](#)
- [isSpace](#)
- [isSpaceChar](#)
- [isTitleCase](#)
- [isUnicodeIdentifierPart](#)
- [isUnicodeIdentifierStart](#)
- [isUpperCase](#)
- [isWhitespace](#)
- [toLowerCase](#)
- [toString](#)
- [toTitleCase](#)
- [toUpperCase](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final int COMBINING\_SPACING\_MARK
- static final int CONNECTOR\_PUNCTUATION
- static final int CONTROL
- static final int CURRENCY\_SYMBOL
- static final int DASH\_PUNCTUATION
- static final int DECIMAL\_DIGIT\_NUMBER
- static final int ENCLOSING\_MARK
- static final int END\_PUNCTUATION
- static final int FORMAT
- static final int LETTER\_NUMBER
- static final int LINE\_SEPARATOR
- static final int LOWERCASE\_LETTER
- static final int MATH\_SYMBOL
- static final int MAX\_RADIX
- static final char MAX\_VALUE
- static final int MIN\_RADIX
- static final char MIN\_VALUE
- static final int MODIFIER\_LETTER
- static final int MODIFIER\_SYMBOL
- static final int NON\_SPACING\_MARK
- static final int OTHER\_LETTER
- static final int OTHER\_NUMBER
- static final int OTHER\_PUNCTUATION
- static final int OTHER\_SYMBOL
- static final int PARAGRAPH\_SEPARATOR
- static final int PRIVATE\_USE
- static final int SPACE\_SEPARATOR
- static final int START\_PUNCTUATION
- static final int SURROGATE
- static final int TITLECASE\_LETTER
- static final int UNASSIGNED
- static final int UPPERCASE\_LETTER

Package [java.lang](#)

## Class [Character](#)

### Public Constructor Character

```
Character(char value)
```

#### constructor **Character(char value):**

Constructs a *Character* object and initializes it so that it represents the primitive value argument.

Package [java.lang](#)

## Class [Character](#)

### Public Method charValue

```
char charValue()
```

#### Description:

Returns the value of this *Character* as a *char*.

## Example

```
Character ch = new Character('C');
char charCh = ch.charValue();
Logger.log("char value: " + charCh);
```

The example above returns a char value and writes it to the logger.

Package [java.lang](#)

## Class [Character](#)

### Public Method [compareTo](#)

```
int compareTo(Character anotherCharacter)

int compareTo(Object anObject)
```

#### Description:

##### **compareTo(Character anotherCharacter) method:**

Compares two Characters numerically. Returns the value 0 if the argument is a Character numerically equal to this Character; a value less than 0 if the argument is a Character numerically greater than this Character; and a value greater than 0 if the argument is a Character numerically less than this Character.

##### **compareTo(Object anObject) method:**

Compares this Character to another Object. If the Object is a Character, this function behaves like `compareTo(Character)`. Otherwise, it throws a `ClassCastException` (as Characters are comparable only to other Characters). Returns the value 0 if the argument is a Character numerically equal to this Character; a value less than 0 if the argument is a Character numerically greater than this Character; and a value greater than 0 if the argument is a Character numerically less than this Character.

Package [java.lang](#)

## Class [Character](#)

### Public Method [digit](#)

```
static int digit(char ch, int radix)
```

#### Description:

Returns the numeric value of the character *ch* in the specified radix. If the radix is not in the range  $MIN\_RADIX \leq radix \leq MAX\_RADIX$  or if the value of *ch* is not a valid digit in the specified radix, -1 is returned. A character is a valid digit if at least one of the following is true:

- The method `isDigit` is true of the character and the *Unicode* decimal digit value of the character (or its single-character decomposition) is less than the specified radix. In this case the decimal digit value is returned.
- The character is one of the uppercase Latin letters 'A' through 'Z' and its code is less than  $radix + 'A' - 10$ . In this case,  $ch - 'A' + 10$  is returned.
- The character is one of the lowercase Latin letters 'a' through 'z' and its code is less than  $radix + 'a' - 10$ . In this case,  $ch - 'a' + 10$  is returned.

## Example

```
Logger.log("char value: " + Character.digit('c',10));
```

The example above returns the numeric value.

Package [java.lang](#)

Class [Character](#)

### Public Method equals

```
boolean equals(Object obj)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

This method compares this object to the specified object. The result is true if and only if the argument is not null and is a *Character* object that contains the same byte value as this object.

## Example

```
Character eqA = new Character('c');
Character eqB = new Character('c');
if (eqA.equals(eqB))
    Logger.log("correct: A eq B");
```

The comparison between *eqA* and *eqB* delivers a true result.

Package [java.lang](#)

Class [Character](#)

### Public Method forDigit

```
static char forDigit(int digit, int radix)
```

#### Description:

Determines the character representation for a specific digit in the specified radix. If the value of radix is not a valid radix, or the value of digit is not a valid digit in the specified radix, the null character ('\u0000') is returned. The radix argument is valid if it is greater than or equal to *MIN\_RADIX* and less than or equal to *MAX\_RADIX*. The digit argument is valid if  $0 \leq digit \leq radix$ . If the digit is less than 10, then *'0' + digit* is returned. Otherwise, the value *'a' + digit - 10* is returned.

## Example

```
Logger.log("character value: " + Character.forDigit(14, 16));
```

Returns the character.

Package [java.lang](#)  
Class [Character](#)

**Public Method `getNumericValue`**

```
static int getNumericValue(char ch)
```

**Description:**

Returns the *Unicode* numeric value of the character as a nonnegative integer. If the character does not have a numeric value, then -1 is returned. If the character has a numeric value that cannot be represented as a nonnegative integer (for example, a fractional value), then -2 is returned.

**Example**

```
Logger.log("numeric UNICODE value: " +  
Character.getNumericValue('j'));
```

Returns the *Unicode* numeric value.

Package [java.lang](#)  
Class [Character](#)

**Public Method `getType`**

```
static int getType(char ch)
```

**Description:**

Returns a value indicating a character category.

**Example**

```
Logger.log("get Type: " + Character.getType('j'));
```

Returns the value of type *int*.

Package [java.lang](#)  
Class [Character](#)

**Public Method `hashCode`**

```
int hashCode()
```

**Overrides:**

- [hashCode](#) in class [Object](#)

**Description:**

Returns a *hashcode* for this *Character*.

## Example

```
Character hc = new Character('c');
int hashCode = hc.hashCode();
Logger.log("hashcode: " + hashCode);
```

The example above returns the hashcode value and writes it to the logger.

Package [java.lang](#)

Class [Character](#)

### Public Method isDigit

```
static boolean isDigit(char ch)
```

#### Description:

Determines if the specified character is a digit. A character is considered to be a digit if it is not in the range `'\u2000' ≤ ch ≤ '\u2FFF'` and its *Unicode* name contains the word `"DIGIT"`.

#### Example

```
if (Character.isDigit('1'))
    Logger.log("The character is a digit!");
```

Returns true if the specified character is a digit.

Package [java.lang](#)

Class [Character](#)

### Public Method isIdentifierIgnored

```
static boolean isIdentifierIgnored(char ch)
```

#### Description:

Determines if the specified character should be regarded as an ignorable character in a Java identifier or a *Unicode* identifier. The following *Unicode* characters are ignorable in a Java identifier or a *Unicode* identifier:

- 0x0000 through 0x0008, ISO control characters that
- 0x000E through 0x001B, are not whitespace
- and 0x007F through 0x009F
- 0x200C through 0x200F join controls
- 0x200A through 0x200E bidirectional controls
- 0x206A through 0x206F format controls
- 0xFEFF zero-width no-break space

#### Example

```
if (Character.isIdentifierIgnored((char)0x0008));
    Logger.log("This character is ignorable!");
```

Returns true if the specified character is ignorable.

Package [java.lang](#)  
Class [Character](#)

**Public Method `isISOControl`**

```
static boolean isISOControl(char ch)
```

**Description:**

Determines if the specified character is an ISO control character. A character is considered to be an ISO control character if its code is in the range `\u0000` through `\u001F` or in the range `\u007F` through `\u009F`.

**Example**

```
if (Character.isISOControl((char)0x0008));  
    Logger.log("This character is a control character!");
```

Returns true if the specified character is ignorable.

Package [java.lang](#)  
Class [Character](#)

**Public Method `isJavaIdentifierPart`**

```
static boolean isJavaIdentifierPart(char ch)
```

**Description:**

Determines if the specified character may be part of a Java identifier as other than the first character. A character may be part of a Java identifier if and only if it is one of the following:

- a letter
- a currency symbol (such as "\$")
- a connecting punctuation character (such as "\_").
- a digit
- a numeric letter (such as a Roman numeral character)
- a combining mark
- a non-spacing mark
- an ignorable control character

**Example**

```
if (Character.isJavaIdentifierPart('$'));  
    Logger.log("This character is a Java identifier part character!");
```

Returns true if the specified character is part of the Java identifier part.

Package [java.lang](#)  
Class [Character](#)

**Public Method `isJavaIdentifierStart`**

```
static boolean isJavaIdentifierStart(char ch)
```



**Description:**

Determines if the specified character is permissible as the first character in a Java identifier. A character may start a Java identifier if and only if it is one of the following:

- a letter
- a currency symbol (such as "\$")
- a connecting punctuation character (such as "\_").

**Example**

```
if (Character.isJavaIdentifierStart('$'));
    Logger.log("This character is a Java identifier start
character!");
```

Returns true if the specified character is part of the Java identifier start.

Package [java.lang](#)

Class [Character](#)

**Public Method isJavaLetter**

```
static boolean isJavaLetter(char ch)
```

**Description:**

**Deprecated.** Replaced by `isJavaIdentifierStart(char)`.

Determines if the specified character is a "*Java*" letter, that is, the character is permissible as the first character in an identifier in the Java language.

A character is considered to be a Java letter if and only if it is a letter, the ASCII dollar sign character '\$', or the underscore character '\_'.

**Example**

```
if (Character.isJavaLetter('$'));
    Logger.log("This character is a Java letter!");
```

Returns true if the specified character is a Java letter.

Package [java.lang](#)

Class [Character](#)

**Public Method isJavaLetterOrDigit**

```
static boolean isJavaLetterOrDigit(char ch)
```

**Description:**

**Deprecated.** Replaced by `isJavaIdentifierPart(char)`.

Determines if the specified character is a "*Java*" letter, that is, the character is permissible as a non-initial character in an identifier in the Java language.

A character is considered to be a Java letter if and only if it is a letter, the ASCII dollar sign character '\$', or the underscore character '\_'.

## Example

```
if (Character.isJavaLetterOrDigit('c'));  
    Logger.log("This character is a Java letter or a digit!");
```

Returns true if the specified character is a Java letter.

Package [java.lang](#)

Class [Character](#)

### Public Method `isLetter`

```
static boolean isLetter(char ch)
```

#### Description:

**Deprecated.** Replaced by `isJavaIdentifierStart(char)`.

Determines if the specified character is a "Java" letter, that is, the character is permissible as the first character in an identifier in the Java language.

A character is considered to be a Java letter if and only if it is a letter, the ASCII dollar sign character '\$', or the underscore character '\_'.

## Example

```
if (Character.isLetter('c'));  
    Logger.log("This character is a letter!");
```

Returns true if the specified character is a letter.

Package [java.lang](#)

Class [Character](#)

### Public Method `isLetterOrDigit`

```
static boolean isLetterOrDigit(char ch)
```

#### Description:

**Deprecated.** Replaced by `isJavaIdentifierPart(char)`.

Determines if the specified character is a "Java" letter, that is, the character is permissible as a non-initial character in an identifier in the Java language.

A character is considered to be a Java letter if and only if it is a letter, the ASCII dollar sign character '\$', or the underscore character '\_'.

## Example

```
if (Character.isLetterOrDigit('1'));  
    Logger.log("This character is a digit of a letter!");
```

Returns true if the specified character is a digit.

Package [java.lang](#)  
Class [Character](#)

**Public Method isLowerCase**

```
static boolean isLowerCase(char ch)
```

**Description:**

Determines if the specified character is a lowercase character. A character is lowercase if it is not in the range '\u2000' through '\u2FFF', the *Unicode* attribute table does not specify a mapping to lowercase for the character, and at least one of the following is true:

- The attribute table specifies a mapping to uppercase for the character.
- The name for the character contains the words "SMALL LETTER".
- The name for the character contains the words "SMALL LIGATURE".

A character is considered to be lowercase if and only if it is specified to be lowercase by the *Unicode 2.0* standard (category "Ll" in the *Unicode* specification data file). Many other *Unicode* characters are lowercase, too.

**Example**

```
if (Character.isLowerCase('c'));  
    Logger.log("This character is a lowercase character!");
```

Returns true if the specified character is a lowercase character.

Package [java.lang](#)  
Class [Character](#)

**Public Method isSpace**

```
static boolean isSpace(char ch)
```

**Description:**

**Deprecated.** Replaced by `isWhitespace(char)`.

Determines if the specified character is *ISO-LATIN-1* whitespace. This method returns true for the following five characters only:

mode	0 ... 3	backlight mode
'\t'	\u0009	HORIZONTAL TABULATION
'\n'	\u000A	NEW LINE
'\f'	\u000C	FORM FEED
'\r'	\u000D	CARRIAGE RETURN
' '	\u0020	SPACE

## Example

```
if (Character.isSpace('\t'));
    Logger.log("This character is a Java space character!");
```

Returns true if the specified character is a Java space character.

Package [java.lang](#)

Class [Character](#)

### Public Method `isSpaceChar`

```
static boolean isSpaceChar(char ch)
```

#### Description:

Determines if the specified character is a *Unicode* space character. A character is considered to be a space character if and only if it is specified to be a space character by the *Unicode 2.0 standard* (category "Zs", "Zl", or "Zp" in the *Unicode* specification data file).

## Example

```
if (Character.isSpaceChar('\t'));
    Logger.log("This character is a Java space or Unicode
character!");
```

Returns true if the specified character is a Java space character.

Package [java.lang](#)

Class [Character](#)

### Public Method `isTitleCase`

```
static boolean isTitleCase(char ch)
```

#### Description:

Converts the character argument to titlecase. A character has a titlecase equivalent if and only if a titlecase mapping is specified for the character in the *Unicode* attribute table. Note that some *Unicode* characters in the range '\u2000' through '\u2FFF' have titlecase mappings; this method does map such characters to their titlecase equivalents even though the method `isTitleCase` does not return true for such characters. There are only four *Unicode* characters that are truly titlecase forms that are distinct from uppercase forms. As a rule, if a character has no true titlecase equivalent but does have an uppercase mapping, then the *Unicode 2.0* attribute table specifies a titlecase mapping that is the same as the uppercase mapping.

## Example

```
if (Character.isTitleCase('\u2000'));
    Logger.log("This character is a titlecase character!");
```

Returns true if the specified character is a titlecase character.

Package [java.lang](#)  
Class [Character](#)

**Public Method [isUnicodeIdentifierPart](#)**

```
static boolean isUnicodeIdentifierPart(char ch)
```

**Description:**

Determines if the specified character may be part of a *Unicode* identifier as other than the first character. A character may be part of a *Unicode* identifier if and only if it is one of the following:

- a letter
- a connecting punctuation character (such as "\_").
- a digit
- a numeric letter (such as a Roman numeral character)
- a combining mark
- a non-spacing mark
- an ignorable control character

**Example**

```
if (Character.isUnicodeIdentifierPart('_'));  
    Logger.log("This character is a Unicode identifier part  
character!");
```

Returns true if the specified character is part of the Unicode identifier part.

Package [java.lang](#)  
Class [Character](#)

**Public Method [isUnicodeIdentifierStart](#)**

```
static boolean isUnicodeIdentifierStart(char ch)
```

**Description:**

Determines if the specified character is permissible as the first character in a *Unicode* identifier. A character may start a *Unicode* identifier if and only if it is a letter.

**Example**

```
if (Character.isUnicodeIdentifierStart('c'));  
    Logger.log("This character is a Unicode identifier start!");
```

Returns true if the specified character is part of the Unicode identifier start character.

Package [java.lang](#)  
Class [Character](#)

**Public Method [isUpperCase](#)**

```
static boolean isUpperCase(char ch)
```

### Description:

Determines if the specified character is an uppercase character. A character is uppercase if it is not in the range '*Unicode*\u2000' through '*Unicode*\u2FFF', the Unicode attribute table does not specify a mapping to uppercase for the character, and at least one of the following is true:

- The attribute table specifies a mapping to lowercase for the character.
- The name for the character contains the words "*Unicode*CAPITAL LETTER".
- The name for the character contains the words "*Unicode*CAPITAL LIGATURE".

### Example

```
if (Character.isUpperCase('A'));  
    Logger.log("This character is a uppercase character!");
```

Returns true if the specified character is a uppercase character.

### Package [java.lang](#) Class [Character](#)

#### Public Method isWhitespace

```
static boolean isWhitespace(char ch)
```

### Description:

Determines if the specified character is whitespace according to Java. A character is considered to be a Java whitespace character if and only if it satisfies one of the following criteria:

- It is a Unicode space separator (category "Zs"), but is not a no-break space (\u00A0 or \uFEFF).
- It is a Unicode line separator (category "Zl").
- It is a Unicode paragraph separator (category "Zp").
- It is \u0009, HORIZONTAL TABULATION.
- It is \u000A, LINE FEED.
- It is \u000B, VERTICAL TABULATION.
- It is \u000C, FORM FEED.
- It is \u000D, CARRIAGE RETURN.
- It is \u001C, FILE SEPARATOR.
- It is \u001D, GROUP SEPARATOR.
- It is \u001E, RECORD SEPARATOR.
- It is \u001F, UNIT SEPARATOR.

### Example

```
if (Character.isWhitespace('\t'));  
    Logger.log("This character is a Java whitespace character!");
```

Returns true if the specified character is a Java whitespace character.

Package [java.lang](#)  
Class [Character](#)

### Public Method `toLowerCase`

```
static char toLowerCase(char ch)
```

#### Description:

The given character is mapped to its lowercase equivalent; if the character has no lowercase equivalent, the character itself is returned. A character has a lowercase equivalent if and only if a lowercase mapping is specified for the character in the Unicode attribute table. Note that some *Unicode* characters in the range '\u2000' to '\u2FFF' have lowercase mappings; this method does map such characters to their lowercase equivalents even though the method *isUpperCase* does not return true for such characters.

#### Example

```
char ch = 'A';  
Logger.log("chosen character: " + ch);  
Logger.log("equivalent lowercase character: " +  
Character.toLowerCase(ch));
```

Returns the equivalent lowercase character.

Package [java.lang](#)  
Class [Character](#)

### Public Method `toString`

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

#### Description:

Returns a *String* object representing this character's value. Converts this *Character* object to a string. The result is a string whose length is 1. The string's sole component is the primitive char value represented by this object.

#### Example

```
Character ch = new Character('A');  
Logger.log(ch.toString());
```

Returns the String object representation this character object.

## Package [java.lang](#) Class [Character](#)

### Public Method toTitleCase

```
static char toTitleCase(char ch)
```

#### Description:

Converts the character argument to titlecase. A character has a titlecase equivalent if and only if a titlecase mapping is specified for the character in the *Unicode* attribute table. Note that some *Unicode* characters in the range '\u2000' through '\u2FFF' have titlecase mappings; this method does map such characters to their titlecase equivalents even though the method *isTitleCase* does not return true for such characters. There are only four *Unicode* characters that are truly titlecase forms that are distinct from uppercase forms. As a rule, if a character has no true titlecase equivalent but does have an uppercase mapping, then the *Unicode 2.0* attribute table specifies a titlecase mapping that is the same as the uppercase mapping.

#### Example

```
Logger.log(Character.toTitleCase('g'));
```

Converts the character argument to titlecase.

## Package [java.lang](#) Class [Character](#)

### Public Method toUpperCase

```
static char toUpperCase(char ch)
```

#### Description:

Converts the character argument to uppercase. A character has an uppercase equivalent if and only if an uppercase mapping is specified for the character in the *Unicode* attribute table. Note that some *Unicode* characters in the range '\u2000' to '\u2000FFF' have uppercase mappings; this method does map such characters to their titlecase equivalents even though the method *isLowerCase* does not return true for such characters.

#### Example

```
Logger.log("" + Character.toUpperCase('g'));
```

Converts the specified character to an uppercase character.

## Package [java.lang](#)

### Class [ClassCastException](#)

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.



## Public Constructors

- [ClassCastException](#)

## Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class [ClassCastException](#)

#### Public Constructor ClassCastException

```
ClassCastException()
```

```
ClassCastException(String s)
```

#### Description:

##### constructor `ClassCastException()`:

Constructs an `ClassCastException` with no detail message.

##### constructor `ClassCastException(String s)`:

Constructs an `ClassCastException` with the specified detail message.

## Package [java.lang](#)

### Class `ClassNotFoundException`

extends [Exception](#) → [Throwable](#) → [Object](#)

Thrown when an application tries to load a class through its string name using the `forName`-method in class `Class` but no definition for the class could be found.

## Public Constructors

- [ClassNotFoundException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.lang](#)

## Class [ClassNotFoundException](#)

### Public Constructor [ClassNotFoundException](#)

```
ClassNotFoundException()
```

```
ClassNotFoundException(String s)
```

#### Description:

#### constructor [ClassNotFoundException\(\)](#):

Constructs an [ClassNotFoundException](#) with no detail message.

#### constructor [ClassNotFoundException\(String s\)](#):

Constructs an [ClassNotFoundException](#) with the specified detail message.

### Package [java.lang](#)

## Interface [Cloneable](#)

A class implements the [Cloneable](#) interface to indicate to the `Object.clone()` method that it is legal for that method to make a field-for-field copy of instances of that class. Attempts to clone instances that do not implement the [Cloneable](#) interface result in the exception [CloneNotSupportedException](#) being thrown. The interface [Cloneable](#) declares no methods.

### Package [java.lang](#)

## Class [CloneNotSupportedException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that the clone method in class `Object` has been called to clone an object, but that the object's class does not implement the [Cloneable](#) interface.

## Public Constructors

- [CloneNotSupportedException](#)

## Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class [CloneNotSupportedException](#)

#### Public Constructor CloneNotSupportedException

```
CloneNotSupportedException ()
```

```
CloneNotSupportedException (String s)
```

#### Description:

##### constructor `CloneNotSupportedException()`:

Constructs an `CloneNotSupportedException` with no detail message.

##### constructor `CloneNotSupportedException(String s)`:

Constructs an `CloneNotSupportedException` with the specified detail message.

## Package [java.lang](#)

### Interface `Comparable`

This interface imposes a total ordering on the objects of a class that implements it.

#### Public Methods

- [compareTo](#)

**Package [java.lang](#)****Interface [Comparable](#)****Public Method [compareTo](#)**

```
abstract int compareTo(Object o)
```

**Description:**

Compares this object with the specified object. Returns a negative integer, zero, or a positive integer when this object is less than, equal to, or greater than the specified object.

**Package [java.lang](#)****Class [Error](#)**

extends [Throwable](#) → [Object](#)

Indicates serious problems that an application should not try to catch. These errors are abnormal conditions that should never occur.

**Public Constructors**

- [Error](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [Error](#)****Public Constructor [Error](#)**

```
Error ()
```

```
Error (String s)
```

**Description:****constructor [Error](#)():**

Constructs an [Error](#) with no detail message.

**constructor Error(String s):**

Constructs an Error with the specified detail message.

**Package [java.lang](#)****Class Exception**

extends [Throwable](#) → [Object](#)

Indicates conditions that an application might want to catch.

**Public Constructors**

- [Exception](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [Exception](#)****Public Constructor Exception**

```
Exception ()
```

```
Exception (String s)
```

**Description:****constructor Exception():**

Constructs an Exception with no detail message.

**constructor Exception(String s):**

Constructs an Exception with the specified detail message.

## Package [java.lang](#)

### Class `Float`

extends [Number](#) → [Object](#)

implements [Serializable](#), [Comparable](#)

The `Float` class wraps a value of primitive type float in an object. An object of type `Float` contains a single field whose type is float.

#### Public Constructors

- [Float](#)

#### Public Methods

- [compareTo](#)
- [equals](#)
- [floatToIntBits](#)
- [floatValue](#)
- [hashCode](#)
- [intBitsToFloat](#)
- [intValue](#)
- [isInfinite](#)
- [isNaN](#)
- [toString](#)
- [valueOf](#)

#### Methods inherited from [java.lang.Number](#)

- [byteValue](#)
- [floatValue](#)
- [intValue](#)
- [shortValue](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Public Fields

- static final float MAX\_VALUE
- static final float MIN\_VALUE
- static final float NaN
- static final float NEGATIVE\_INFINITY
- static final float POSITIVE\_INFINITY

**Package [java.lang](#)****Class [Float](#)****Public Constructor [Float](#)**

```
Float(float value)
```

```
Float(String s)
```

**Throws:**

- [NumberFormatException](#)

**Description:****constructor [Float\(float value\)](#):**

Constructs a newly allocated *Float* object that represents the primitive *float* argument.

**constructor [Float\(String s\)](#):**

Constructs a newly allocated *Float* object that represents the floating-point value of type *float* represented by the string. The string is converted to a float value as if by the *valueOf* method.

**Package [java.lang](#)****Class [Float](#)****Public Method [compareTo](#)**

```
int compareTo(Float anotherFloat)
```

```
int compareTo(Object anObject)
```

**Description:****[compareTo\(Float anotherFloat\)](#) method:**

Compares two Floats numerically. There are two ways in which comparisons performed by this method differ from those performed by the Java language numerical comparison operators (<, ≤, ==, ≥ >) when applied to primitive floats:

- *Float.NaN* is considered by this method to be equal to itself and greater than all other float values (including *Float.POSITIVE\_INFINITY*).
- *0.0f* is considered by this method to be greater than *-0.0f*.

This ensures that *Float.compareTo(Object)* (which inherits its behavior from this method) obeys the general contract for *Comparable.compareTo*, and that the natural order on Floats is total.

Returns the value 0 if *anotherFloat* is numerically equal to this *Float*; a value less than 0 if this *Float* is numerically less than *anotherFloat*; and a value greater than 0 if this *Float* is numerically greater than *anotherFloat*.

**compareTo(Object anObject) method:**

Compares this Float to another Object. If the Object is a Float, this function behaves like compareTo(Float). Otherwise, it throws a ClassCastException (as Floats are comparable only to other Floats).

Returns the value 0 if anotherFloat is numerically equal to this Float; a value less than 0 if this Float is numerically less than anotherFloat; and a value greater than 0 if this Float is numerically greater than anotherFloat.

Package [java.lang](#)

**Class [Float](#)****Public Method equals**

```
boolean equals(Object obj)
```

**Overrides:**

- [equals](#) in class [Object](#)

**Description:**

Compares this object against some other object. The result is true if and only if the argument is not null and is a Float object that represents a float that has the identical bit pattern to the bit pattern of the float represented by this object. For this purpose, two float values are considered to be the same if and only if the method `floatToIntBits(float)` returns the same int value when applied to each. Note that in most cases, for two instances of class Float, f1 and f2, the value of `f1.equals(f2)` is true if and only if `f1.floatValue() == f2.floatValue()` also has the value true.

There are two exceptions:

- If f1 and f2 both represent Float.NaN, then the equals method returns true, even though Float.NaN==Float.NaN has the value false.
- If f1 represents +0.0f while f2 represents -0.0f, or vice versa, the equal test has the value false, even though `0.0f== -0.0f` has the value true.

**Example**

```
Integer data = Integer.decode("20");
```

The example above decodes a string to an Integer number.

Package [java.lang](#)

**Class [Float](#)****Public Method floatToIntBits**

```
static int floatToIntBits(float value)
```

**Description:**

Returns the bit representation of a single-float value. The result is a representation of the floating-point argument according to the IEEE 754 floating-point "single precision" bit layout.



- Bit 31 (the bit that is selected by the mask `0x80000000`) represents the sign of the floating-point number.
- Bits 30-23 (the bits that are selected by the mask `0x7f800000`) represent the exponent.
- Bits 22-0 (the bits that are selected by the mask `0x007fffff`) represent the significand (sometimes called the mantissa) of the floating-point number.
- If the argument is positive infinity, the result is `0x7f800000`.
- If the argument is negative infinity, the result is `0xff800000`.
- If the argument is NaN, the result is `0x7fc00000`.

In all cases, the result is an integer that, when given to the `intBitsToFloat(int)` method, will produce a floating-point value equal to the argument to `floatToIntBits`.

### Example

```
int data = Float.floatToIntBits(80000000f);
```

The example above returns the bit representation.

Package [java.lang](#)

Class **Float**

**Public Method floatValue**

```
float floatValue()
```

Overrides:

- [floatValue](#) in class [Number](#)

Description:

Returns the value of this *Float* as a *float*.

### Example

```
Float f1 = new Float(20);  
float floatf1 = f1.floatValue();  
Logger.log("float value: " + floatf1);
```

The example above returns a float value and writes it to the logger.

Package [java.lang](#)

Class **Float**

**Public Method hashCode**

```
int hashCode()
```

Overrides:

- [hashCode](#) in class [Object](#)

**Description:**

Returns a *hashCode* for this *Float* object. The result is the integer bit representation, exactly as produced by the method *floatToIntBits(float)*, of the primitive float value represented by this *Float* object.

**Example**

```
Integer in = new Integer("10");
int floatfl = fl.hashCode();
Logger.log("hash code: " + floatfl);
```

The example above returns the hash code and writes it to the logger.

Package [java.lang](#)

**Class [Float](#)****Public Method intBitsToFloat**

```
static float intBitsToFloat(int bits)
```

**Description:**

Returns the single-float corresponding to a given bit representation. The argument is considered to be a representation of a floating-point value according to the IEEE 754 floating-point "single precision" bit layout. If the argument is *0x7f800000*, the result is positive infinity. If the argument is *0xff800000*, the result is negative infinity. If the argument is any value in the range *0x7f800001* through *0x7fffffff* or in the range *0xff800001* through *0xffffffff*, the result is NaN. All IEEE 754 NaN values of type float are, in effect, lumped together by the Java programming language into a single float value called *NaN*. Distinct values of NaN are only accessible by use of the *Float.floatToRawIntBits* method.

**Example**

```
Integer in = new Integer("10");
int floatfl = fl.hashCode();
Logger.log("hash code: " + floatfl);
```

The example above returns the hash code and writes it to the logger.

Package [java.lang](#)

**Class [Float](#)****Public Method intValue**

```
int intValue()
```

**Overrides:**

- [intValue](#) in class [Number](#)

**Description:**

Returns the integer value of this *Float* (by casting to an int).

## Example

```
Float flo = new Float(10);
int value = flo.intValue();
Logger.log("integer value: " + value);
```

The example above returns the *float* value converted to *int* type.

Package [java.lang](#)

## Class [Float](#)

### Public Method isInfinite

```
boolean isInfinite()
static boolean isInfinite(float v)
```

#### Description:

##### **isInfinite()** method:

Returns true if this *Float* value is infinitely large in magnitude.

##### **isInfinite(float v)** method:

Returns true if the specified number is infinitely large in magnitude.

## Example

```
Float flo = new Float(10);
if (flo.isInfinite())
    Logger.log("is infinite");
```

The example above returns true if the value is infinitely large in magnitude.

Package [java.lang](#)

## Class [Float](#)

### Public Method isNaN

```
boolean isNaN()
static boolean isNaN(float v)
```

#### Description:

##### **isNaN()** method:

Returns true if this *Float* value is Not-a-Number (NaN).

##### **isNaN(float v)** method:

Returns true if the specified number is the special Not-a-Number (NaN) value.

## Example

```
if (Float.isNaN(20.23465f))
    Logger.log("the specified number is NaN");
```

The example above returns true if the value is not a number.

Package [java.lang](#)

## Class [Float](#)

### Public Method [toString](#)

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

```
static String toString(float f)
```

#### Description:

##### [toString\(\)](#) method:

Returns a *String* representation of this *Float* object. The primitive float value represented by this object is converted to a *String* exactly as if by the method *toString* of one argument.

##### [toString\(float f\)](#) method:

Returns a *String* representation for the specified *float* value. The argument is converted to a readable string format as follows. All characters and characters in strings mentioned below are ASCII characters.

- If the argument is *NaN*, the result is the string *"NaN"*.
- Otherwise, the result is a string that represents the sign and magnitude (absolute value) of the argument. If the sign is negative, the first character of the result is '-' ('-'); if the sign is positive, no sign character appears in the result. As for the magnitude *m*:
  - If *m* is infinity, it is represented by the characters "Infinity"; thus, positive infinity produces the result "Infinity" and negative infinity produces the result "-Infinity".
  - If *m* is zero, it is represented by the characters "0.0"; thus, negative zero produces the result "-0.0" and positive zero produces the result "0.0".
  - If *m* is greater than or equal to  $10^{-3}$  but less than  $10^7$ , then it is represented as the integer part of *m*, in decimal form with no leading zeroes, followed by '.' (.), followed by one or more decimal digits representing the fractional part of *m*.
  - If *m* is less than  $10^{-3}$  or greater or equal to  $10^7$ , then it is represented in so-called "computerized scientific notation". Let *n* be the unique integer such that  $10^n \leq m < 10^{n+1}$ ; then let *a* be the mathematically exact quotient of *m* and  $10^n$  so that  $1 < a < 10$ . The magnitude is then represented as the integer part of *a*, as a single decimal digit, followed by '.' (.), followed by decimal digits representing the fractional part of *a*, followed by the letter 'E' (E), followed by a representation of *n* as a decimal integer, as produced by the method *Integer.toString(int)* of one argument.

## Example

```
Logger.log(Float.toString(10.4362627f));
```

The example above returns the float value converted to a string in science notation.

Package [java.lang](#)

## Class **Float**

### Public Method **valueOf**

```
static Float valueOf(String s)
```

#### Throws:

- [NumberFormatException](#)

#### Description:

Returns the floating point value represented by the specified *String*. The string *s* is interpreted as the representation of a floating-point value and a *Float* object representing that value is created and returned. If *s* is *null*, then a *NullPointerException* is thrown. Leading and trailing whitespace characters in *s* are ignored. The rest of the string *s* should constitute a float value. If it does not have the form of a float value, then a *NumberFormatException* is thrown.

## Example

```
Logger.log("integer value: " + Float.valueOf("3.40283e30f"));
```

The example above returns the floating point value represented by the specified *String*.

Package [java.lang](#)

## Class **IllegalArgumentException**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that a method has been passed an illegal or inappropriate argument.

### Public Constructors

- [IllegalArgumentException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)

- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [IllegalArgumentException](#)

### Public Constructor [IllegalArgumentException](#)

```
IllegalArgumentException ()
```

```
IllegalArgumentException (String s)
```

#### Description:

##### constructor [IllegalArgumentException\(\)](#):

Constructs an [IllegalArgumentException](#) with no detail message.

##### constructor [IllegalArgumentException\(String s\)](#):

Constructs an [IllegalArgumentException](#) with the specified detail message.

Package [java.lang](#)

## Class [IllegalMonitorStateException](#)

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.

### Public Constructors

- [IllegalMonitorStateException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [IllegalMonitorStateException](#)

### Public Constructor IllegalMonitorStateException

```
IllegalMonitorStateException ()
```

```
IllegalMonitorStateException (String s)
```

#### Description:

##### constructor `IllegalMonitorStateException()`:

Constructs an `IllegalMonitorStateException` with no detail message.

##### constructor `IllegalMonitorStateException(String s)`:

Constructs an `IllegalMonitorStateException` with the specified detail message.

Package [java.lang](#)

## Class `IllegalStateException`

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Signals that a method has been invoked at an illegal or inappropriate time, i.e. the Java Environment or the Java Program is not in an appropriate state for the requested operation.

### Public Constructors

- [IllegalStateException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [IllegalStateException](#)

### Public Constructor IllegalStateException

```
IllegalStateException ()
```

```
IllegalStateException(String s)
```

**Description:****constructor `IllegalStateException()`:**

Constructs an `IllegalStateException` with no detail message.

**constructor `IllegalStateException(String s)`:**

Constructs an `IllegalStateException` with the specified detail message.

Package [java.lang](#)

**Class `IncompatibleClassChangeError`**

extends [LinkageError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown when the definition of some class, on which the currently executing method depends, has changed.

**Public Constructors**

- [IncompatibleClassChangeError](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

**Class [IncompatibleClassChangeError](#)****Public Constructor `IncompatibleClassChangeError`**

```
IncompatibleClassChangeError()
```

```
IncompatibleClassChangeError(String s)
```



**Description:****constructor `IncompatibleClassChangeError()`:**

Constructs an `IncompatibleClassChangeError` with no detail message.

**constructor `IncompatibleClassChangeError(String s)`:**

Constructs an `IncompatibleClassChangeError` with the specified detail message.

Package [java.lang](#)

**Class `IndexOutOfBoundsException`**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

**Public Constructors**

- [IndexOutOfBoundsException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

**Class [IndexOutOfBoundsException](#)****Public Constructor `IndexOutOfBoundsException`**

```
IndexOutOfBoundsException ()
```

```
IndexOutOfBoundsException (String s)
```

**Description:****constructor `IndexOutOfBoundsException()`:**

Constructs an `IndexOutOfBoundsException` with no detail message.

**constructor `IndexOutOfBoundsException(String s)`:**

Constructs an `IndexOutOfBoundsException` with the specified detail message.

**Package [java.lang](#)****Class `Integer`**

extends [Number](#) → [Object](#)

implements [Serializable](#), [Comparable](#)

This class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

**Public Constructors**

- [Integer](#)

**Public Methods**

- [compareTo](#)
- [decode](#)
- [equals](#)
- [floatValue](#)
- [hashCode](#)
- [intValue](#)
- [parseInt](#)
- [toBinaryString](#)
- [toHexString](#)
- [toOctalString](#)
- [toString](#)
- [valueOf](#)

**Methods inherited from [java.lang.Number](#)**

- [byteValue](#)
- [floatValue](#)
- [intValue](#)
- [shortValue](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Public Fields**

- static final `int` `MAX_VALUE`
- static final `int` `MIN_VALUE`

Package [java.lang](#)

## Class [Integer](#)

### Public Constructor Integer

```
Integer(int value)
```

```
Integer(String s)
```

#### Throws:

- [NumberFormatException](#)

#### Description:

##### **constructor Integer(int value):**

This constructs a newly allocated Integer object that represents the specified int value.

##### **constructor Integer(String s):**

This constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

Package [java.lang](#)

## Class [Integer](#)

### Public Method compareTo

```
int compareTo(Integer anotherInteger)
```

```
int compareTo(Object anObject)
```

#### Description:

##### **compareTo(Integer anotherInteger) method:**

Compares two Integers numerically.

Returns the value 0 if the argument Integer is equal to this Integer; a value less than 0 if this Integer is numerically less than the Integer argument; and a value greater than 0 if this Integer is numerically greater than the Integer argument (signed comparison).

##### **compareTo(Object anObject) method:**

Compares this Integer to another Object. If the Object is a Integer, this function behaves like compareTo(Integer). Otherwise, it throws a ClassCastException (as Integers are comparable only to other Integers).

Returns the value 0 if the argument Integer is equal to this Integer; a value less than 0 if this Integer is numerically less than the Integer argument; and a value greater than 0 if this Integer is numerically greater than the Integer argument (signed comparison).

## Package [java.lang](#)

### Class [Integer](#)

#### Public Method decode

```
static Integer decode(String nm)
```

#### Throws:

- [NumberFormatException](#)

#### Description:

This method decodes a *String* into an *Integer*. It accepts decimal, hexadecimal, and octal number in the following formats:

- - decimal constant
- 0x - hexadecimal constant
- # - hexadecimal constant
- 0 - octal constant

The constant following an (optional) negative sign and/or "radix specifier" is parsed as by the *Integer.parseInt* method with the specified radix (10, 8 or 16). This constant must be positive or a *NumberFormatException* will result. The result is made negative if first character of the specified *String* is the negative sign. No whitespace characters are permitted in the *String*.

#### Example

```
Integer data = Integer.decode("20");
```

The example above decodes a string to an Integer number.

## Package [java.lang](#)

### Class [Integer](#)

#### Public Method equals

```
boolean equals(Object obj)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

This method compares this object to the specified object. The result is true if and only if the argument is not null and is an *Integer* object that contains the same int value as this object.

#### Example

```
Integer eqA = new Integer("10");  
Integer eqB = new Integer("10");  
if (eqA.equals(eqB))  
    Logger.log("correct: A eq B");
```

The comparison between  $eqA$  and  $eqB$  delivers a true result.

Package [java.lang](#)

Class [Integer](#)

### Public Method floatValue

```
float floatValue()
```

#### Overrides:

- [floatValue](#) in class [Number](#)

#### Description:

Returns the value of this *Integer* as a *float*.

#### Example

```
Integer in = new Integer("10");
float floatInt = in.floatValue();
Logger.log("float value: " + floatInt);
```

The example above returns a float value and writes it to the logger.

Package [java.lang](#)

Class [Integer](#)

### Public Method hashCode

```
int hashCode()
```

#### Overrides:

- [hashCode](#) in class [Object](#)

#### Description:

Returns a *hashcode* for this *Integer*.

#### Example

```
Integer hc = new Integer("10");
int hashCode = hc.hashCode();
Logger.log("hashcode: " + hashCode);
```

The example above returns the hashcode value and writes it to the logger.

Package [java.lang](#)

Class [Integer](#)

### Public Method intValue

```
int intValue()
```

**Overrides:**

- [intValue](#) in class [Number](#)

**Description:**

Returns an *int* of this *Integer*.

**Example**

```
Integer iv = new Integer("10");
int intvalue = iv.intValue();
Logger.log("intvalue: " + intvalue);
```

The example above returns the int value and writes it to the logger.

Package [java.lang](#)

**Class [Integer](#)****Public Method [parseInt](#)**

```
static int parseInt(String s)
```

**Throws:**

- [NumberFormatException](#)

```
static int parseInt(String s, int radix)
```

**Throws:**

- [NumberFormatException](#)

**Description:****[parseInt\(String s\)](#) method:**

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002d') to indicate a negative value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the *parseInt(String, int)* method.

**[parseInt\(String s, int radix\)](#) method:**

Parses the string argument as a signed integer in the radix specified by the second argument. The characters in the string must all be digits of the specified radix (as determined by whether *Character.digit(char, int)* returns a nonnegative value), except that the first character may be an ASCII minus sign '-' ('\u002d') to indicate a negative value. The resulting integer value is returned.

An exception of type *NumberFormatException* is thrown if any of the following situations occurs:

- The first argument is null or is a string of length zero.

- The radix is either smaller than `Character.MIN_RADIX` or larger than `Character.MAX_RADIX`.
- Any character of the string is not a digit of the specified radix, except that the first character may be a minus sign `'-'` (`'\u002d'`) provided that the string is longer than length 1.
- The integer value represented by the string is not a value of type `int`.

### Example

```
Logger.log("String to integer value: " + Integer.parseInt("23", 10));
Logger.log("Hexadecimal to integer value: " + Integer.parseInt("23",
16));
Logger.log("Octal to integer value: " + Integer.parseInt("23", 8));
```

The example above writes different integer values to the logger.

Package [java.lang](#)

Class [Integer](#)

### Public Method `toBinaryString`

```
static String toBinaryString(int i)
```

#### Description:

Creates a string representation of the integer argument as an unsigned integer in base 2. The unsigned integer value is the argument plus  $2^{32}$  if the argument is negative; otherwise it is equal to the argument. This value is converted to a string of ASCII digits in binary (base 2) with no extra leading `0s`. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character. The characters `'0'` (`'\u0030'`) and `'1'` (`'\u0031'`) are used as binary digits.

#### Example

```
String toSt = Integer.toBinaryString(10);
```

The `toBinaryString()` method returns the binary `String` representation of the decimal value 10.

Package [java.lang](#)

Class [Integer](#)

### Public Method `toHexString`

```
static String toHexString(int i)
```

#### Description:

Creates a string representation of the integer argument as an unsigned integer in base 16. The unsigned integer value is the argument plus  $2^{32}$  if the argument is negative; otherwise, it is equal to the argument. The `int` value is converted to a string of ASCII digits in hexadecimal (base 16) with no extra leading `0s` for the string representation. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character.

## Example

```
String toSt = Integer.toHexString(10);
```

The `toHexString()` method returns the hexadecimal *String* representation of the decimal value 10.

Package [java.lang](#)

Class [Integer](#)

### Public Method toOctalString

```
static String toOctalString(int i)
```

#### Description:

Creates a string representation of the integer argument as an unsigned integer in base 8. The unsigned integer value is the argument plus  $2^{32}$  if the argument is negative; otherwise, it is equal to the argument. This value is converted to a string of ASCII digits in octal (base 8) with no extra leading `0s`. If the unsigned magnitude is zero, it is represented by a single zero character `'0'` (`'\u0030'`); otherwise, the first character of the representation of the unsigned magnitude will not be the zero character.

## Example

```
String toSt = Integer.toOctalString(10);
```

The `toOctalString()` method returns the octal *String* representation of the decimal value 10.

Package [java.lang](#)

Class [Integer](#)

### Public Method toString

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

```
static String toString(int i)
```

```
static String toString(int i, int radix)
```

#### Description:

#### `toString()` method:

Returns a *String* object representing this *Integer*'s value. The value is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString(int)` method.



**toString(int i) method:**

Returns a new *String* object representing the specified integer. The argument is converted to signed decimal representation and returned as a string, exactly as if the argument and radix 10 were given as arguments to the *toString(int, int)* method.

**toString(int i, int radix) method:**

Creates a string representation of the first argument in the radix specified by the second argument. If the radix is smaller than *Character.MIN\_RADIX* or larger than *Character.MAX\_RADIX*, then the radix 10 is used instead. If the first argument is negative, the first element of the result is the ASCII minus character '-' ('\u002d'). If the first argument is not negative, no sign character appears in the result. The remaining characters of the result represent the magnitude of the first argument. If the magnitude is zero, it is represented by a single zero character '0' ('\u0030'); otherwise, the first character of the representation of the magnitude will not be the zero character.

**Example**

```
String toSt = Integer.toString(10);
```

The *toString()* method returns the decimal *String* representation of the decimal value 10.

**Package [java.lang](#)****Class [Integer](#)****Public Method [valueOf](#)**

```
static Integer valueOf(String s)
```

**Throws:**

- [NumberFormatException](#)

```
static Integer valueOf(String s, int radix)
```

**Throws:**

- [NumberFormatException](#)

**Description:****valueOf(String s) method:**

Returns a new *Integer* object initialized to the value of the specified *String*. The argument is interpreted as representing a signed decimal integer, exactly as if the argument were given to the *parseInt(java.lang.String)* method. The result is an *Integer* object that represents the integer value specified by the string.

**valueOf(String s, int radix) method:**

Returns a new *Integer* object initialized to the value of the specified *String*. The first argument is interpreted as representing a signed integer in the radix specified by the second argument, exactly as if the arguments were given to the *parseInt(java.lang.String, int)* method. The result is an *Integer* object that represents the integer value specified by the string.

## Example

```
Integer in = Integer.valueOf("9");
```

The `valueOf()` method returns the *Integer* representation of the decimal value 10.

## Package [java.lang](#)

### Class **InternalError**

extends [VirtualMachineError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown to indicate some unexpected internal error has occurred in the Java Virtual Machine.

#### Public Constructors

- [InternalError](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class **InternalError**

#### **Public Constructor InternalError**

```
InternalError()
```

```
InternalError(String s)
```

#### Description:

##### **constructor InternalError():**

Constructs an `InternalError` with no detail message.

##### **constructor InternalError(String s):**

Constructs an `InternalError` with the specified detail message.

## Package [java.lang](#)

### Class **InterruptedException**

extends [Exception](#) → [Throwable](#) → [Object](#)

Thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the interrupt method in class Thread.

#### Public Constructors

- [InterruptedException](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class **InterruptedException**

#### Public Constructor InterruptedException

```
InterruptedException()
```

```
InterruptedException(String s)
```

#### Description:

##### **constructor InterruptedException():**

Constructs an InterruptedException with no detail message.

##### **constructor InterruptedException(String s):**

Constructs an InterruptedException with the specified detail message.

## Package [java.lang](#)

### Class **LinkageError**

extends [Error](#) → [Throwable](#) → [Object](#)

Indicates that a class has some dependency on another class and the latter has incompatibly changed after the compilation of the former class.

### Public Constructors

- [LinkageError](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.lang](#)

## Class [LinkageError](#)

### Public Constructor LinkageError

```
LinkageError()
```

```
LinkageError(String s)
```

#### Description:

#### constructor `LinkageError()`:

Constructs an `LinkageError` with no detail message.

#### constructor `LinkageError(String s)`:

Constructs an `LinkageError` with the specified detail message.

### Package [java.lang](#)

## Class `Math`

extends [Object](#)

Contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

### Public Constructors

- [Math](#)

## Public Methods

- [abs](#)
- [acos](#)
- [asin](#)
- [atan](#)
- [atan2](#)
- [ceil](#)
- [cos](#)
- [exp](#)
- [floor](#)
- [log](#)
- [max](#)
- [min](#)
- [pow](#)
- [rint](#)
- [round](#)
- [sin](#)
- [sqrt](#)
- [tan](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final float E
- static final float PI

## Package [java.lang](#)

### Class [Math](#)

#### Public Constructor Math

```
Math()
```

## Package [java.lang](#)

### Class [Math](#)

#### Public Method abs

```
static float abs(float a)
```

```
static int abs(int a)
```

**Description:****abs(float a) method:**

This method returns the absolute value of a float value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

**Note**

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

**abs(int a) method:**

This method returns the absolute value of an *int* value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

**Note**

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

**Example**

```
Draw.drawText("data: " + Math.abs(-3.2315523f));
```

The example above draws the calculated value to the screen.

**Package [java.lang](#)****Class [Math](#)****Public Method acos**

```
static float acos(float a)
```

**Description:**

Returns the arc cosine of an angle, in the range of 0.0 through pi. If the argument is NaN or its absolute value is greater than 1, then the result is NaN.

**Example**

```
Draw.drawText("acos of cos PI: " + Math.acos(Math.cos(Math.PI)));
```

The example above draws the calculated value of the constant PI to the screen.

**Package [java.lang](#)****Class [Math](#)****Public Method asin**

```
static float asin(float a)
```

**Description:**

Returns the arc sine of an angle, in the range of  $-\pi/2$  through  $\pi/2$ .

**Note**

- If the argument is NaN or its absolute value is greater than 1, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

**Example**

```
Draw.drawText("asin of sin PI/2: " +  
Math.asin(Math.sin(Math.PI/2)));
```

The example above draws the calculated value of  $\pi / 2$  to the screen.

Package [java.lang](#)

Class [Math](#)

**Public Method atan**

```
static float atan(float a)
```

**Description:**

Returns the arc tangent of an angle, in the range of  $-\pi/2$  through  $\pi/2$ .

**Note**

- If the argument is NaN, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

**Example**

```
Draw.drawText("atan: " + Math.atan(0.02344f));
```

The example above draws the calculated value to the screen.

Package [java.lang](#)

Class [Math](#)

**Public Method atan2**

```
static float atan2(float a, float b)
```

**Description:**

Converts rectangular coordinates (a, b) to polar (r, theta) coordinates. This method computes the phase theta by computing an arc tangent of a/b in the range of  $-\pi$  to  $\pi$ .

**Note**

- If either argument is NaN, then the result is NaN.

- If the first argument is positive zero and the second argument is positive, or the first argument is positive and finite and the second argument is positive infinity, then the result is positive zero.
- If the first argument is negative zero and the second argument is positive, or the first argument is negative and finite and the second argument is positive infinity, then the result is negative zero.
- If the first argument is positive zero and the second argument is negative, or the first argument is positive and finite and the second argument is negative infinity, then the result is the float value closest to  $\pi$ .
- If the first argument is negative zero and the second argument is negative, or the first argument is negative and finite and the second argument is negative infinity, then the result is the float value closest to  $-\pi$ .
- If the first argument is positive and the second argument is positive zero or negative zero, or the first argument is positive infinity and the second argument is finite, then the result is the float value closest to  $\pi/2$ .
- If the first argument is negative and the second argument is positive zero or negative zero, or the first argument is negative infinity and the second argument is finite, then the result is the float value closest to  $-\pi/2$ .
- If both arguments are positive infinity, then the result is the float value closest to  $\pi/4$ .
- If the first argument is positive infinity and the second argument is negative infinity, then the result is the float value closest to  $3\pi/4$ .
- If the first argument is negative infinity and the second argument is positive infinity, then the result is the float value closest to  $-\pi/4$ .
- If both arguments are negative infinity, then the result is the float value closest to  $-3\pi/4$ .

### Example

```
Draw.writeText("atan2: " + Math.atan2(0.02344f), 0.45632f);
```

The example above draws the calculated value to the screen.

### Package [java.lang](#)

### Class [Math](#)

### Public Method `ceil`

```
static float ceil(float a)
```

### Description:

Returns the smallest (closest to negative infinity) float value that is greater than or equal to the argument and is equal to a mathematical integer.

### Note

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.
- If the argument value is less than zero but greater than -1.0, then the result is negative zero.

### Example

```
float data = Math.ceil(3.2315523f);
```

The example above returns a value of 4.



**Package [java.lang](#)****Class [Math](#)****Public Method cos**

```
static float cos(float a)
```

**Description:**

Returns the trigonometric cosine of an angle. If the argument is NaN or an infinity, then the result is NaN.

**Example**

```
Draw.writeText("cos of PI: " + Math.cos(Math.PI));
```

The example above draws a calculated value of -1 to the screen.

**Package [java.lang](#)****Class [Math](#)****Public Method exp**

```
static float exp(float a)
```

**Description:**

Returns Euler's number e raised to the power of a float value.

**Note**

- If the argument is NaN, the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is negative infinity, then the result is positive zero.

**Example**

```
Draw.writeText("data: " + Math.exp(3.2315523f));
```

The example above draws the calculated value to the screen.

**Package [java.lang](#)****Class [Math](#)****Public Method floor**

```
static float floor(float a)
```

**Description:**

Returns the largest (closest to positive infinity) float value that is less than or equal to the argument and is equal to a mathematical integer.

## Note

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

## Example

```
Draw.drawText("data: " + Math.floor(3.2315523f));
```

The example above draws a calculated value of 3 to the screen.

Package [java.lang](#)

Class [Math](#)

## Public Method [log](#)

```
static float log(float a)
```

### Description:

This method returns the natural logarithm (base e) of a float value.

## Note

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is negative infinity.

## Example

```
Draw.drawText("data: " + Math.log(3.2315523f));
```

The example above draws the calculated value to the screen.

Package [java.lang](#)

Class [Math](#)

## Public Method [max](#)

```
static float max(float a, float b)
```

```
static int max(int a, int b)
```

### Description:

#### **max(float a, float b) method:**

This method returns the greater value of the two float values. That is, the result is the argument closer to positive infinity.

## Note

- If the arguments have the same value, the result is that same value.
- If either value is NaN, then the result is NaN.
- Unlike the the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero.
- If one argument is positive zero and the other negative zero, the result is positive zero.

### **max(int a, int b) method:**

Returns the greater value of two the int values. That is, the result is the argument closer to the value of Integer.MAX\_VALUE. If the arguments have the same value, the result is that same value.

### Example

```
Draw.drawText("data: " + Math.max(3.2315523f, 9.23f));
```

The example above draws the greater value to the screen.

Package [java.lang](#)

Class [Math](#)

### Public Method min

```
static float min(float a, float b)
```

```
static int min(int a, int b)
```

### Description:

#### **min(float a, float b) method:**

This method returns the smaller value of the two float parameter values. That is, the result is the value closer to negative infinity.

## Note

- If the arguments have the same value, the result is that same value.
- If either value is NaN, then the result is NaN.
- Unlike the numerical comparison operators, this method considers negative zero to be strictly smaller than positive zero.
- If one argument is positive zero and the other is negative zero, the result is negative zero.

#### **min(int a, int b) method:**

This method returns the smaller value of the two int parameter values. That is, the result the argument closer to the value of Integer.MIN\_VALUE. If the arguments have the same value, the result is that same value.

### Example

```
Draw.drawText("data: " + Math.min(3.2315523f, 9.23f));
```

The example above draws the smaller value to the screen.

**Package [java.lang](#)****Class [Math](#)****Public Method [pow](#)**

```
static float pow(float a, float b)
```

**Description:**

This method calculates and returns the value of  $a^b$ . In the foregoing descriptions, a floating-point value is considered to be an integer if and only if it is finite and a fixed point of the method `ceil` or, equivalently, a fixed point of the method `floor`. A value is a fixed point of a one-argument method if and only if the result of applying the method to the value is equal to the value.

**Note**

- If the second argument is positive or negative zero, then the result is 1.0.
- If the second argument is 1.0, then the result is the same as the first argument.
- If the second argument is NaN, then the result is NaN.
- If the first argument is NaN and the second argument is nonzero, then the result is NaN.
- If the absolute value of the first argument is greater than 1 and the second argument is positive infinity, then the result is positive infinity.
- If the absolute value of the first argument is less than 1 and the second argument is negative infinity, then the result is positive infinity.
- If the absolute value of the first argument is greater than 1 and the second argument is negative infinity, then the result is positive zero.
- If the absolute value of the first argument is less than 1 and the second argument is positive infinity, then the result is positive zero.
- If the absolute value of the first argument equals 1 and the second argument is infinite, then the result is NaN.
- If the first argument is positive zero and the second argument is greater than zero, then the result is positive zero.
- If the first argument is positive infinity and the second argument is less than zero, then the result is positive zero.
- If the first argument is positive zero and the second argument is less than zero, then the result is positive infinity.
- If the first argument is positive infinity and the second argument is greater than zero, then the result is positive infinity.
- If the first argument is negative zero and the second argument is greater than zero but not a finite odd integer, then the result is positive zero.
- If the first argument is negative infinity and the second argument is less than zero but not a finite odd integer, then the result is positive zero.
- If the first argument is negative zero and the second argument is a positive finite odd integer, then the result is negative zero.
- If the first argument is negative infinity and the second argument is a negative finite odd integer, then the result is negative zero.
- If the first argument is negative zero and the second argument is less than zero but not a finite odd integer, then the result is positive infinity.
- If the first argument is negative infinity and the second argument is greater than zero but not a finite odd integer, then the result is positive infinity.
- If the first argument is negative zero and the second argument is a negative finite odd integer, then the result is negative infinity.
- If the first argument is negative infinity and the second argument is a positive finite odd integer, then the result is negative infinity.
- If the first argument is finite and less than zero and if the second argument is a finite even integer, the result is equal to the result of raising the absolute value of the first argument to the power of the second argument.

- If the first argument is finite and less than zero and if the second argument is a finite odd integer, the result is equal to the negative of the result of raising the absolute value of the first argument to the power of the second argument.
- If the first argument is finite and less than zero and if the second argument is finite and not an integer, then the result is NaN.
- If both arguments are integers, then the result is exactly equal to the mathematical result of raising the first argument to the power of the second argument if that result can in fact be represented exactly as a float value.

### Example

```
Draw.drawText("data: " + Math.pow(3.2315523f), 9.23f);
```

The example above draws the calculated value to the screen.

Package [java.lang](#)

Class [Math](#)

### Public Method rint

```
static float rint(float a)
```

#### Description:

Returns the float value that is closest in value to *a* and is equal to a mathematical integer. If two float values that are mathematical integers are equally close to the value of the argument, the result is the integer value that is even.

#### Note

- If the argument value is already equal to a mathematical integer, then the result is the same as the argument.
- If the argument is NaN or an infinity or positive zero or negative zero, then the result is the same as the argument.

### Example

```
Draw.drawText("data: " + Math.rint(9.23f));
```

The example above draws the calculated value to the screen.

Package [java.lang](#)

Class [Math](#)

### Public Method round

```
static int round(float a)
```

#### Description:

Rounds a float number to the nearest integer number (round up or down) and returns this value as int. The result is rounded to an integer by adding 1/2, taking the floor of the result, and casting the result to type int.

## Note

- If the argument is NaN, the result is 0.
- If the argument is negative infinity or any value less than or equal to the value of `Integer.MIN_VALUE`, the result is equal to the value of `Integer.MIN_VALUE`.
- If the argument is positive infinity or any value greater than or equal to the value of `Integer.MAX_VALUE`, the result is equal to the value of `Integer.MAX_VALUE`.

## Example

```
Draw.writeText("data: " + Math.round(9.23f));
```

The example above draws the calculated value to the screen.

Package [java.lang](#)

Class [Math](#)

### Public Method sin

```
static float sin(float a)
```

#### Description:

Returns the trigonometric sine of an angle.

#### Note

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

## Example

```
Draw.writeText("sin: " + Math.sin(Math.PI / 2f));
```

The example above draws a calculated value of 1 to the screen.

Package [java.lang](#)

Class [Math](#)

### Public Method sqrt

```
static float sqrt(float a)
```

#### Description:

This method returns a rounded and positive square root of a float value. Otherwise, the result is the float value closest to the true mathematical square root of the argument value.

#### Note

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

## Example

```
Draw.writeText("data: " + Math.sqrt(4f));
```

The example above draws a calculated value of 2 to the screen.

Package [java.lang](#)

## Class [Math](#)

### Public Method [tan](#)

```
static float tan(float a)
```

#### Description:

Returns the trigonometric tan of an angle.

#### Note

- If the argument is NaN or an infinity, then the result is NaN.
- If the argument is zero, then the result is a zero with the same sign as the argument.

## Example

```
Draw.writeText("data: " + Math.tan(0.2315523f));
```

The example above draws the calculated value to the screen.

Package [java.lang](#)

## Class [NegativeArraySizeException](#)

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown if an application tries to create an array with negative size.

### Public Constructors

- [NegativeArraySizeException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)

- [notifyAll](#)

Package [java.lang](#)

## Class [NegativeArraySizeException](#)

### Public Constructor NegativeArraySizeException

```
NegativeArraySizeException ()
```

```
NegativeArraySizeException (String s)
```

#### Description:

##### constructor NegativeArraySizeException():

Constructs an NegativeArraySizeException with no detail message.

##### constructor NegativeArraySizeException(String s):

Constructs an NegativeArraySizeException with the specified detail message.

Package [java.lang](#)

## Class NoClassDefFoundError

extends [LinkageError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown if the Java Virtual Machine tries to load the definition of a class that could not be found.

### Public Constructors

- [NoClassDefFoundError](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [NoClassDefFoundError](#)

### Public Constructor NoClassDefFoundError

```
NoClassDefFoundError ()
```



```
NoClassDefFoundError(String s)
```

**Description:****constructor NoClassDefFoundError():**

Constructs an NoClassDefFoundError with no detail message.

**constructor NoClassDefFoundError(String s):**

Constructs an NoClassDefFoundError with the specified detail message.

**Package [java.lang](#)****Class NoSuchFieldError**

extends [IncompatibleClassChangeError](#) → [LinkageError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown if an application tries to access or modify a specified field of an object, and that object no longer has that field.

**Public Constructors**

- [NoSuchFieldError](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [NoSuchFieldError](#)****Public Constructor [NoSuchFieldError](#)**

```
NoSuchFieldError()
```

```
NoSuchFieldError(String s)
```

**Description:****constructor NoSuchFieldError():**

Constructs an NoSuchFieldError with no detail message.

**constructor NoSuchFieldError(String s):**

Constructs an NoSuchFieldError with the specified detail message.

**Package [java.lang](#)****Class NoSuchMethodError**

extends [IncompatibleClassChangeError](#) → [LinkageError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown if an application tries to call a specified method of a class (either static or instance), and that class no longer has a definition of that method.

**Public Constructors**

- [NoSuchMethodError](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [NoSuchMethodError](#)****Public Constructor NoSuchMethodError**

```
NoSuchMethodError ()
```

```
NoSuchMethodError (String s)
```

**Description:****constructor NoSuchMethodError():**

Constructs an NoSuchMethodError with no detail message.

**constructor NoSuchMethodError(String s):**

Constructs an NoSuchMethodError with the specified detail message.

**Package [java.lang](#)****Class [NullPointerException](#)**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown when an application attempts to use null in a case where an object is required.

**Public Constructors**

- [NullPointerException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [NullPointerException](#)****Public Constructor [NullPointerException](#)**

```
NullPointerException()
```

```
NullPointerException(String s)
```

**Description:****constructor [NullPointerException\(\)](#):**

Constructs an NullPointerException with no detail message.

**constructor [NullPointerException\(String s\)](#):**

Constructs an NullPointerException with the specified detail message.

## Package [java.lang](#)

### Abstract Class **Number**

extends [Object](#)

The abstract class *Number* is the superclass of classes *Byte*, *Float*, *Integer*, and *Short*. Subclasses of *Number* must provide methods to convert the represented numeric value to *float*, and *int*.

#### Public Constructors

- [Number](#)

#### Public Methods

- [byteValue](#)
- [floatValue](#)
- [intValue](#)
- [shortValue](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class [Number](#)

#### Public Constructor **Number**

```
Number ()
```

## Package [java.lang](#)

### Class [Number](#)

#### Public Method **byteValue**

```
byte byteValue ()
```

#### Description:

Returns the value of the specified value as a *byte*. This may involve rounding.

## Package [java.lang](#)

### Class [Number](#)

#### Public Method **floatValue**

```
abstract float floatValue ()
```

**Description:**

Returns the value of the specified value as a *float*. This may involve rounding.

Package [java.lang](#)

Class [Number](#)

**Public Method intValue**

```
abstract int intValue()
```

**Description:**

Returns the value of the specified value as a *int*. This may involve rounding.

Package [java.lang](#)

Class [Number](#)

**Public Method shortValue**

```
short shortValue()
```

**Description:**

Returns the value of the specified value as a *short*. This may involve rounding.

Package [java.lang](#)

**Class NumberFormatException**

extends [IllegalArgumentException](#) → [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

**Public Constructors**

- [NumberFormatException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [NumberFormatException](#)

### Public Constructor [NumberFormatException](#)

```
NumberFormatException()
```

```
NumberFormatException(String s)
```

#### Description:

##### constructor [NumberFormatException\(\)](#):

Constructs an [NumberFormatException](#) with no detail message.

##### constructor [NumberFormatException\(String s\)](#):

Constructs an [NumberFormatException](#) with the specified detail message.

Package [java.lang](#)

## Class Object

In Java all classes are ultimately inherited from a single base class (single rooted hierarchy). This base class is *Object*. Every Java class has *Object* as it's superclass. All objects, including arrays, implement the methods of this class, so all objects in Java are ultimately of the same type.

### Public Constructors

- [Object](#)

### Public Methods

- [equals](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)
- [toString](#)
- [wait](#)

Package [java.lang](#)

## Class [Object](#)

### Public Constructor [Object](#)

```
Object()
```

#### Description:

##### constructor [Object\(\)](#):

This Constructor is run before any other.

**Package [java.lang](#)****Class [Object](#)****Protected Method clone**

```
Object clone()
```

**Throws:**

- [CloneNotSupportedException](#)

**Description:**

This method creates and returns a copy of this object.

**Package [java.lang](#)****Class [Object](#)****Public Method equals**

```
boolean equals(Object obj)
```

**Description:**

Indicates whether some other object is "equal to" this one. For any non-null reference values x and y, this method returns true if and only if x and y refer to the same object, i.e.  $x == y$  is true.

It is generally recommended to override the hashCode method whenever the equals method is overridden, to ensure that equal objects have equal hash codes.

**Package [java.lang](#)****Class [Object](#)****Public Method hashCode**

```
int hashCode()
```

**Description:**

Returns a hash code value for this object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable.

**Package [java.lang](#)****Class [Object](#)****Public Method notify**

```
final void notify()
```

**Description:**

Wakes up a single thread that is waiting for this object's monitor. If many threads are waiting on this object, one of them is chosen to be awakened. A thread waits on an object's monitor by calling one of the wait methods.

This method should only be called by a thread that is the owner of this object's monitor. A thread becomes the owner of the object's monitor in one of three ways:

- By executing a synchronized instance method of that object.
- By executing the body of a synchronized statement that synchronizes on the object.
- For objects of type Class, by executing a synchronized static method of that class.

Only one thread at a time can own an object's monitor.

Package [java.lang](#)

Class [Object](#)

**Public Method `notifyAll`**

```
final void notifyAll()
```

**Description:**

Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods.

This method should only be called by a thread that is the owner of this object's monitor. A thread becomes the owner of the object's monitor in one of three ways:

- By executing a synchronized instance method of that object.
- By executing the body of a synchronized statement that synchronizes on the object.
- For objects of type Class, by executing a synchronized static method of that class.

Only one thread at a time can own an object's monitor.

Package [java.lang](#)

Class [Object](#)

**Public Method `toString`**

```
String toString()
```

**Description:**

Returns a string representation of the object. In general, the `toString` method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

Package [java.lang](#)

Class [Object](#)

**Public Method `wait`**

```
final void wait()
```

**Throws:**

- [InterruptedException](#)

```
final void wait(int timeout)
```



**Throws:**

- [InterruptedException](#)

```
final void wait(int timeout, int nanos)
```

**Throws:**

- [InterruptedException](#)

**Parameters**

- **timeout** – timeout in milliseconds.
- **nanos** – additional time in nanoseconds.

**Description:**

Causes the current thread to wait until another thread invokes the `notify()` or the `notifyAll()` method for this object or a certain amount of time has elapsed.

The current thread must own this object's monitor.

**wait() method:**

Behaves as if `wait(0)` were called.

**wait(int timeout) method:**

Wait until notified or timeout in ms has been reached. If timeout is zero, however, then real time is not taken into consideration and the thread simply waits until notified.

**wait(int timeout, int nanos) method:**

Like the above method except that it allows for finer granularity.

**Package [java.lang](#)****Class `OutOfMemoryError`**

extends [VirtualMachineError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

**Public Constructors**

- [OutOfMemoryError](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)

- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.lang](#)

## Class [OutOfMemoryError](#)

### Public Constructor OutOfMemoryError

```
OutOfMemoryError()
```

```
OutOfMemoryError(String s)
```

#### Description:

##### constructor [OutOfMemoryError\(\)](#):

Constructs an [OutOfMemoryError](#) with no detail message.

##### constructor [OutOfMemoryError\(String s\)](#):

Constructs an [OutOfMemoryError](#) with the specified detail message.

### Package [java.lang](#)

## Interface [Runnable](#)

This interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a `run` - method.

#### Public Methods

- [run](#)

### Package [java.lang](#)

## Interface [Runnable](#)

### Public Method `run`

```
abstract void run()
```

#### Description:

When an object implementing interface [Runnable](#) is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread.

## Package [java.lang](#)

### Class **RuntimeException**

extends [Exception](#) → [Throwable](#) → [Object](#)

Superclass for those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

#### Public Constructors

- [RuntimeException](#)

#### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class **RuntimeException**

#### Public Constructor RuntimeException

```
RuntimeException()
```

```
RuntimeException(String s)
```

#### Description:

##### constructor RuntimeException():

Constructs an RuntimeException with no detail message.

##### constructor RuntimeException(String s):

Constructs an RuntimeException with the specified detail message.

## Package [java.lang](#)

### Class **SecurityException**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown by the security manager to indicate a security violation.

## Public Constructors

- [SecurityException](#)

## Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Package [java.lang](#)

### Class [SecurityException](#)

#### Public Constructor SecurityException

```
SecurityException()
```

```
SecurityException(String s)
```

#### Description:

##### constructor `SecurityException()`:

Constructs an `SecurityException` with no detail message.

##### constructor `SecurityException(String s)`:

Constructs an `SecurityException` with the specified detail message.

## Package [java.lang](#)

### Class Short

extends [Number](#) → [Object](#)

implements [Serializable](#), [Comparable](#)

The `Short` class is the standard wrapper for short values.

#### Public Constructors

- [Short](#)

## Public Methods

- [compareTo](#)
- [decode](#)
- [equals](#)
- [floatValue](#)
- [hashCode](#)
- [intValue](#)
- [parseShort](#)
- [shortValue](#)
- [toString](#)
- [valueOf](#)

## Methods inherited from [java.lang.Number](#)

- [byteValue](#)
- [floatValue](#)
- [intValue](#)
- [shortValue](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

## Public Fields

- static final short MAX\_VALUE
- static final short MIN\_VALUE

## Package [java.lang](#)

## Class [Short](#)

### Public Constructor [Short](#)

```
Short(short value)
```

```
Short(String s)
```

### Throws:

- [NumberFormatException](#)

### Description:

#### constructor [Short](#)(short value):

This constructs a newly allocated *Short* object that represents the specified short value.

**constructor Short(String s):**

This constructs a newly allocated *Short* object that represents the byte value indicated by the *String* parameter. The radix is assumed to be 10.

Package [java.lang](#)

**Class [Short](#)****Public Method compareTo**

```
int compareTo(Object anObject)
int compareTo(Short anotherShort)
```

**Description:****compareTo(Short anotherShort) method:**

Compares two Shorts numerically.

Returns the value 0 if the argument Short is equal to this Short; a value less than 0 if this Short is numerically less than the Short argument; and a value greater than 0 if this Short is numerically greater than the Short argument (signed comparison).

**compareTo(Object anObject) method:**

Compares this Short to another Object. If the Object is a Short, this function behaves like compareTo(Short). Otherwise, it throws a ClassCastException (as Shorts are comparable only to other Shorts).

Returns the value 0 if the argument is a Short numerically equal to this Short; a value less than 0 if the argument is a Short numerically greater than this Short; and a value greater than 0 if the argument is a Short numerically less than this Short.

Package [java.lang](#)

**Class [Short](#)****Public Method decode**

```
static Short decode(String nm)
```

**Throws:**

- [NumberFormatException](#)

**Description:**

This method decodes a String into a *Short*. It accepts decimal, hexadecimal, and octal number in the following formats:

- - decimal constant
- 0x - hexadecimal constant
- # - hexadecimal constant
- 0 - octal constant

The constant following an (optional) negative sign and/or "radix specifier" is parsed as by the `Short.parseShort` method with the specified radix (10, 8 or 16). This constant must be positive or a `NumberFormatException` will result. The result is made negative if first character of the specified `String` is the negative sign. No whitespace characters are permitted in the `String`.

### Example

```
Short data = Short.decode("20");
```

The example above decodes a string to a Short number.

Package [java.lang](#)

Class [Short](#)

### Public Method equals

```
boolean equals(Object obj)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

This method compares this object to the specified object. The result is true if and only if the argument is not null and is an `Short` object that contains the same short value as this object.

### Example

```
Short eqA = new Short("10");
Short eqB = new Short("10");
if (eqA.equals(eqB))
    Logger.log("correct: A eq B");
```

The comparison between `eqA` and `eqB` delivers a true result.

Package [java.lang](#)

Class [Short](#)

### Public Method floatValue

```
float floatValue()
```

#### Overrides:

- [floatValue](#) in class [Number](#)

#### Description:

Returns the value of this `Short` as a `float`.

## Example

```
Short sh = new Short("10");
float floatShort = sh.floatValue();
Logger.log("short value: " + floatShort);
```

The example above returns a float value and writes it to the logger.

Package [java.lang](#)

Class [Short](#)

### Public Method hashCode

```
int hashCode()
```

#### Overrides:

- [hashCode](#) in class [Object](#)

#### Description:

Returns a *hashCode* for this *Short*.

## Example

```
Short sh = new Short("10");
int hashCode = sh.hashCode();
Logger.log("hashcode: " + hashCode);
```

The example above returns the hashcode value and writes it to the logger.

Package [java.lang](#)

Class [Short](#)

### Public Method intValue

```
int intValue()
```

#### Overrides:

- [intValue](#) in class [Number](#)

#### Description:

Returns an *int* value of this *Short*.

## Example

```
Short sh = new Short("10");
int intvalue = sh.intValue();
Logger.log("intvalue: " + intvalue);
```

The example above returns the int value and writes it to the logger.



**Package [java.lang](#)****Class [Short](#)****Public Method [parseShort](#)**

```
static short parseShort(String s)
```

**Throws:**

- [NumberFormatException](#)

```
static short parseShort(String s, int radix)
```

**Throws:**

- [NumberFormatException](#)

**Description:****[parseByte\(String s\)](#) method:**

Assuming the specified *String* represents a short, returns that short's value. Throws an exception if the *String* cannot be parsed as a short. The radix is assumed to be 10.

**[parseByte\(String s, int radix\)](#) method:**

Assuming the specified *String* represents a short, returns that short's value. Throws an exception if the *String* cannot be parsed as a byte.

**Example**

```
Logger.log("String to short value: " + Short.parseShort("23", 10));  
Logger.log("Hexadecimal to short value: " + Short.parseShort("23",  
16));  
Logger.log("Octal to short value: " + Short.parseShort("23", 8));
```

The example above writes different short values to the logger.

**Package [java.lang](#)****Class [Short](#)****Public Method [shortValue](#)**

```
short shortValue()
```

**Overrides:**

- [shortValue](#) in class [Number](#)

**Description:****[shortValue\(\)](#) method:**

Returns the value of this Short as a short.

**Package [java.lang](#)****Class [Short](#)****Public Method toString**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

```
static String toString(short s)
```

**Description:****toString() method:**

Returns a *String* object representing this *Short*'s value.

**toString(short s) method:**

Returns a new *String* object representing the specified *Short*. The radix is assumed to be 10.

**Example**

```
Logger.log(Short.toString((short)20));
```

The example above returns the short value converted to a string.

**Package [java.lang](#)****Class [Short](#)****Public Method valueOf**

```
static Short valueOf(String s)
```

**Throws:**

- [NumberFormatException](#)

```
static Short valueOf(String s, int radix)
```

**Throws:**

- [NumberFormatException](#)

**Description:****valueOf(String s) method:**

Assuming the specified *String* represents a short, returns a new *Short* object initialized to that value. Throws an exception if the *String* cannot be parsed as a short. The radix is assumed to be 10.

**valueOf(String s, int radix) method:**

Assuming the specified *String* represents a short, returns a new *Short* object initialized to that value. Throws an exception if the *String* cannot be parsed as a short.

**Example**

```
Short sh = Short.valueOf("9");
```

The *valueOf()* method returns the *Short* representation of the value 9.

**Package [java.lang](#)****Class [StackOverflowError](#)**

extends [VirtualMachineError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown when a stack overflow occurs because an application recurses too deeply.

**Public Constructors**

- [StackOverflowError](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

**Package [java.lang](#)****Class [StackOverflowError](#)****Public Constructor StackOverflowError**

```
StackOverflowError()
```

```
StackOverflowError(String s)
```

**Description:****constructor StackOverflowError():**

Constructs an `StackOverflowError` with no detail message.

## constructor `StackOverflowError(String s)`:

Constructs an `StackOverflowError` with the specified detail message.

## Package [java.lang](#)

## Class `String`

extends [Object](#)

implements [Serializable](#), [Comparable](#)

The `String` class represents character sequences. All string literals (e.g. "abc") are implemented as instances of this class. The `String` class is immutable, i.e. once it is created a `String` object cannot be changed. If there is a necessity to make a lot of modifications to `Strings` of characters, then you should use the `StringBuffer` class.

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase.

The Java language provides special support for the string concatenation operator ( + ), and for conversion of other objects to strings. String concatenation is implemented through the `StringBuffer` class and its `append` method. String conversions are implemented through the method `toString`, defined by `Object` and inherited by all classes in Java.

Note that `Strings` are UTF8 encoded internally and `char` variables have only one byte. This has to be taken into account when concatenating strings with `chars` that exceed 127. Use the method [Draw.writeASCIIText\(String\)](#) to explicitly interpret the bytes of a string as ASCII.

## Examples

```
byte[] bytes = {(byte)0xE2, (byte)0x88, (byte)0x80}; // UTF8 code for
"V"
String str = new String(bytes, 0, 3); // initializes String to "V"
char c = 'u'; // c will have the value 0xBC although the Unicode symbol
is 0x3BC.
```

## Public Constructors

- [String](#)

## Public Methods

- [charAt](#)
- [compareTo](#)
- [endsWith](#)
- [equals](#)
- [equalsIgnoreCase](#)
- [getBytes](#)
- [getChars](#)
- [hashCode](#)
- [indexOf](#)
- [lastIndexOf](#)
- [length](#)
- [regionMatches](#)
- [replace](#)

- [startsWith](#)
- [substring](#)
- [toCharArray](#)
- [toLowerCase](#)
- [toString](#)
- [toUpperCase](#)
- [trim](#)
- [valueOf](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.lang](#)

### Class **String**

#### **Public Constructor String**

```
String()
```

```
String(byte[] src, int offs, int cnt)
```

#### Throws:

- [StringIndexOutOfBoundsException](#)

```
String(char[] src)
```

```
String(char[] src, int offs, int cnt)
```

#### Throws:

- [StringIndexOutOfBoundsException](#)

```
String(String other)
```

```
String(StringBuffer sb)
```

#### Description:

##### constructor **String**():

Create a String object that represents an empty character sequence.

##### constructor **String(byte[] src, int offs, int cnt)**:

Constructs a new String by decoding the specified subarray of bytes using UTF8 encoding.

**constructor String(char[] src):**

Allocates a new String so that it represents the sequence of characters currently contained in the character array argument.

Note that characters in the Java Virtual Machine are only represented with one byte. Only the lowest byte of a Unicode symbol will be stored in a character variable (see example below).

**constructor String(char[] src, int offs, int cnt):**

Allocates a new String that contains characters from a subarray of the character array argument.

Note that characters in the Java Virtual Machine are only represented with one byte. Only the lowest byte of a Unicode symbol will be stored in a character variable (see example below).

**constructor String(String other):**

Creates a String object that is a copy of the argument string.

**constructor String(StringBuffer sb):**

Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

**Examples**

```
byte[] bytes = {(byte)0xE2, (byte)0x88, (byte)0x80}; // UTF8 code for
"V"
String str = new String(bytes, 0, 3); // initializes String to "V"
char c = 'µ'; // c will have the value 0xBC although the Unicode symbol
is 0x3BC.
```

**Package [java.lang](#)****Class [String](#)****Public Method charAt**

```
char charAt(int index)
```

**Throws:**

- [StringIndexOutOfBoundsException](#)

**Description:**

Returns the character at the specified index, beginning with index 0.

Note that characters in the Java Virtual Machine are only represented with one byte. Only the lowest byte of a Unicode symbol will be stored in a character variable.

**Example**

```
char c = 'µ'; // c will have the value 0xBC although the Unicode symbol
is 0x3BC.
```

Package [java.lang](#)  
Class [String](#)

### Public Method compareTo

```
int compareTo(Object anObject)
int compareTo(String other)
```

#### Description:

Compares this string with string other lexicographically based on the Unicode value of each character.

This method returns the value 0 if the argument string is equal to this string, a value less than 0 if this string is lexicographically less than the string argument and a value greater than 0 if this string is lexicographically greater than the string argument.

#### Example

```
int i = "abc".compareTo("abd"); // returns -1
int i = "abc".compareTo("abc"); // returns 0
int i = "abc".compareTo("abb"); // returns 1
```

Package [java.lang](#)  
Class [String](#)

### Public Method endsWith

```
boolean endsWith(String suffix)
```

#### Description:

Tests if this string ends with the specified string suffix. Note that the result will be true if the argument is the empty string or is equal to this string

#### Example

```
"blablabla this is the end".endsWith("this is the end"); // returns true
```

Package [java.lang](#)  
Class [String](#)

### Public Method equals

```
boolean equals(Object anObject)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

Compares this string to the object anObject. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object.

## Example

```
boolean result = "aaa".equals("aaa"); // returns true
```

Package [java.lang](#)

Class [String](#)

### Public Method equalsIgnoreCase

```
boolean equalsIgnoreCase(String other)
```

#### Description:

Compares this String to another String, ignoring case. Returns true if they are of the same length and corresponding characters are equal except for their case.

#### Example

```
boolean result = "aaa".equalsIgnoreCase("AAA"); // returns true
```

Package [java.lang](#)

Class [String](#)

### Public Method getBytes

```
byte[] getBytes()
```

#### Throws:

- [StringIndexOutOfBoundsException](#)

```
void getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin)
```

#### Throws:

- [StringIndexOutOfBoundsException](#)

#### Description:

##### **getBytes()** method:

Encodes this String into a sequence of bytes using UTF8 encoding. The result will be stored in a new byte array.

##### **getBytes(srcBegin, srcEnd, dst, dstBegin)** method:

Copies characters from this string into the destination byte array. The first character (starting with 0) is at position srcBegin, the last at position srcEnd-1. The characters, converted to bytes, are copied into the subarray of dst starting at index dstBegin. The destination array has to be big enough to hold the converted character bytes. Note that in case of UTF8 encoding only the first byte will be used.



## Examples

```
byte[] arr = "bäuV".getBytes(); // returns [0x62 (0xC3 0xA4) (0xC3
0xBC) (0xE2 0x88 0x80)]
"0123V".getBytes(2, 5, arr, 0); // chars 2,3 and V will be converted to
bytes and inserted into // arr at position 0 resulting in [0x32,
0x33, 0xE2]
```

Package [java.lang](#)

## Class [String](#)

### Public Method getChars

```
char[] getChars ()
```

#### Throws:

- [StringIndexOutOfBoundsException](#)

```
void getChars (int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

#### Throws:

- [StringIndexOutOfBoundsException](#)

#### Description:

#### **getChars() method:**

Returns the characters of this String into a character array.

#### **getChars(srcBegin, srcEnd, dst, dstBegin) method:**

Copies characters from this string into the destination character array. The first character (starting with 0) is at position srcBegin, the last at position srcEnd-1. The characters are copied into the subarray of dst starting at index dstBegin. The destination array has to be big enough to hold the returned characters.

## Examples

```
byte[] arr = "bäuV".getChars(); // returns [0x62 (0xC3 0xA4) (0xC3
0xBC) (0xE2 0x88 0x80)]
"0123V".getChars(2, 5, arr, 0); // chars 2,3 and V will be converted to
bytes and inserted into // arr at position 0 resulting in [0x32,
0x33, 0xE2]
```

**Package [java.lang](#)****Class [String](#)****Public Method hashCode**

```
int hashCode()
```

**Overrides:**

- [hashCode](#) in class [Object](#)

**Description:**

Returns a hash code for this string. The hash code for a String object is computed as  $-s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + \dots + s[n-1]$  where,  $s[i]$  is the  $i$ -th UTF8 byte of the string. The hash value of the empty string is zero.

**Package [java.lang](#)****Class [String](#)****Public Method indexOf**

```
int indexOf(int ch)
```

```
int indexOf(int ch, int sIdx)
```

```
int indexOf(String str)
```

```
int indexOf(String str, int sIdx)
```

**Parameters**

- **ch** – character to find.
- **sIdx** – index starting forward search.
- **str** – string to find.

**Description:****indexOf() method:**

Returns the index within this string of the first occurrence of the specified character.

**indexOf(int ch, int sIdx) method:**

Returns the index within this string of the first occurrence of the specified character, searching from the specified index.

**indexOf(String str) method:**

Returns the index within this string of the first occurrence of the specified substring.

**indexOf(String str, int sIdx) method:**

Returns the index within this string of the first occurrence of the specified substring, searching from the specified index.

Package [java.lang](#)

Class [String](#)

**Public Method lastIndexOf**

```
int lastIndexOf(int ch)

int lastIndexOf(int ch, int eIdx)

int lastIndexOf(String str)

int lastIndexOf(String str, int eIdx)
```

**Parameters**

- **ch** – character to find.
- **eIdx** – index starting backward search.
- **str** – string to find.

**Description:****lastIndexOf() method:**

Returns the index within this string of the first occurrence of the specified character.

**lastIndexOf(int ch, int eIdx) method:**

Returns the index within this string of the first occurrence of the specified character, searching backward starting at the specified index.

**lastIndexOf(String str) method:**

Returns the index within this string of the first occurrence of the specified substring.

**lastIndexOf(String str, int eIdx) method:**

Returns the index within this string of the first occurrence of the specified substring, searching backward starting at the specified index.

Package [java.lang](#)

Class [String](#)

**Public Method length**

```
int length()
```

**Description:**

Returns the length of this string. The length is equal to the number of Unicode characters in the string.

```
int l = "bäüV".length(); // returns 4
```

Package [java.lang](#)

**Class [String](#)****Public Method [regionMatches](#)**

```
boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)
```

```
boolean regionMatches(int toffset, String other, int ooffset, int len)
```

**Parameters:**

- **[ignoreCase]** – if true, ignore case when comparing characters.
- **toffset** – starting index of the subregion in this string.
- **other** – the string argument.
- **ooffset** – starting index of substring in the string argument.
- **len** – number of characters to compare.

**Description:**

Tests if two string regions are equal. A substring of this String object is compared to a substring of the other String object. The substring of this String object begins at index toffset and has length len. The substring the other String object begins at index ooffset and has length len. Note that indices begin with 0.

Returns false if an index is negative or offset+len is greater than the corresponding string.

**regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len) method:**

Extra parameter ignoreCase.

**regionMatches(int toffset, String other, int ooffset, int len) method:**

Same as above with ignoreCase = false.

**Example**

```
boolean result = "123456789".regionMatches(false, 1, "23xx", 0, 2); //  
returns true
```

Package [java.lang](#)  
Class [String](#)

**Public Method replace**

```
String replace(char oldChar, char newChar)
```

**Description:**

Replace all occurrences of oldChar in this string with newChar.

Package [java.lang](#)  
Class [String](#)

**Public Method startsWith**

```
boolean startsWith(String prefix)
```

```
boolean startsWith(String prefix, int toffset)
```

**Description:**

**startsWith(String prefix) method:**

Returns true if this string starts with the specified prefix.

**startsWith(String prefix, int toffset) method:**

Returns true this string starts with the specified prefix starting at index toffset.

Package [java.lang](#)  
Class [String](#)

**Public Method substring**

```
String substring(int sIdx)
```

```
String substring(int sIdx, int eIdx)
```

**Throws:**

- [StringIndexOutOfBoundsException](#)

**Description:**

**substring(int sIdx) method:**

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

**substring(int sIdx, int eIdx) method:**

Returns a new string that is a substring of this string. The substring begins with the character at the specified index sIdx and extends to the index eIdx - 1.

**Package [java.lang](#)****Class [String](#)****Public Method toCharArray**

```
char[] toCharArray()
```

**Description:**

Converts this string to a new character array.

Note that characters in the Java Virtual Machine are only represented with one byte. Only the lowest byte of a Unicode symbol will be stored in a character variable.

**Example**

```
char c = 'µ'; // c will have the value 0xBC although the Unicode symbol  
is 0x3BC.
```

**Package [java.lang](#)****Class [String](#)****Public Method toLowerCase**

```
String toLowerCase()
```

**Description:**

Converts characters A-Z in this String to lower case.

**Package [java.lang](#)****Class [String](#)****Public Method toString**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

This method returns the string itself.

**Package [java.lang](#)****Class [String](#)****Public Method toUpperCase**

```
String toUpperCase()
```

**Description:**

Converts characters a-z in this String to upper case.

Package [java.lang](#)

Class [String](#)

**Public Method trim**

```
String trim()
```

**Description:**

Returns a copy of this string with leading and trailing whitespace characters removed.

Package [java.lang](#)

Class [String](#)

**Public Method valueOf**

```
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(char[] data)
static String valueOf(char[] data, int offset, int count)
static String valueOf(float f)
static String valueOf(int i)
static String valueOf(Object obj)
```

**Description:****valueOf(boolean b) method:**

Returns the string representation of the boolean argument. If the argument is true, "true" is returned else, "false".

**valueOf(char c) method:**

Returns the string representation of the char argument.

**valueOf(char[] data) method:**

Returns the string representation of the char array argument.

**valueOf(char[] data, int offset, int count) method:**

Returns the string representation of a specific subarray of the char array argument.

**valueOf(float f) method:**

Returns the string representation of the float argument.

**valueOf(int i) method:**

Returns the string representation of the int argument.

**valueOf(Object obj) method:**

Returns the string representation of the Object argument. If the argument is null, "null" is returned, else the value of obj.toString().

**Example**

```
String str = str.valueOf(false); // returns "false"  
String str = str.valueOf(0.2f); // returns 2.000000E-01
```

**Package [java.lang](#)****Class StringBuffer**

extends [Object](#)

A string buffer is like a String, but can be modified. If there is a necessity to make a lot of modifications to Strings of characters use this class, otherwise many copies will be created.

Note that string buffers are char arrays and chars occupy one byte in memory. Unicode symbols will be encoded using UTF8 and might need more than one entry in the string buffer.

**Public Constructors**

- [StringBuffer](#)

**Public Methods**

- [append](#)
- [capacity](#)
- [charAt](#)
- [delete](#)
- [ensureCapacity](#)
- [getChars](#)
- [insert](#)
- [length](#)
- [reverse](#)
- [setCharAt](#)
- [setLength](#)
- [toString](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)



- [notifyAll](#)

Package [java.lang](#)

## Class [StringBuffer](#)

### Public Constructor StringBuffer

```
StringBuffer()
```

```
StringBuffer(int length)
```

#### Throws:

- [NegativeArraySizeException](#)

```
StringBuffer(String str)
```

#### Description:

##### constructor [StringBuffer\(\)](#):

Constructs a string buffer with no characters in it and an initial capacity of 16 characters.

##### constructor [StringBuffer\(int length\)](#):

Constructs a string buffer with no characters in it and the specified initial capacity.

##### constructor [StringBuffer\(String str\)](#):

Constructs a string buffer initialized to the contents of the specified string.

Note that Unicode symbols whose UTF8 code need more than one byte will occupy more than one entry in the string buffer.

#### Example

```
StringBuffer strBuffer = new StringBuffer("aV"); // buffer: [(0xC3  
0xA4) (0xE2 0x88 0x80)]
```

Package [java.lang](#)

## Class [StringBuffer](#)

### Public Method append

```
StringBuffer append(boolean b)
```

```
StringBuffer append(byte[] str)
```

```
StringBuffer append(byte[] str, int offset, int len)
```

```
StringBuffer append(char c)
```

```
StringBuffer append(char[] str)
```

```
StringBuffer append(char[] str, int offset, int len)
StringBuffer append(float f)
StringBuffer append(int i)
StringBuffer append(Object obj)
StringBuffer append(String str)
```

**Description:****append(boolean b) method:**

Appends the string representation of the boolean argument to the sequence.

**append(byte[] str) method:**

Appends the string representation of the byte array argument to the sequence.

**append(byte[] str, int offset, int len) method:**

Appends the string representation of the specified subarray of the byte array argument to the sequence.

**append(char c) method:**

Appends the string representation of the char argument to the sequence.

**append(char[] str) method:**

Appends the string representation of the char array argument to the sequence.

**append(char[] str, int offset, int len) method:**

Appends the string representation of the specified subarray of the char array argument to the sequence.

**append(float f) method:**

Appends the string representation of the float argument to the sequence.

**append(int i) method:**

Appends the string representation of the int argument to the sequence.

**append(Object obj) method:**

Appends the string representation of the Object argument to the sequence.

**append(String str) method:**

Appends the string parameter to the sequence.

Package [java.lang](#)  
Class [StringBuffer](#)

**Public Method capacity**

```
int capacity()
```

**Description:**

Returns the current number of possible character entries in the String buffer. If more characters are inserted an allocation of memory will occur.

Package [java.lang](#)  
Class [StringBuffer](#)

**Public Method charAt**

```
char charAt(int index)
```

**Description:**

Returns the character at the specified index, beginning with index 0.

Note that characters in the Java Virtual Machine are only represented with one byte. Only the lowest byte of a Unicode symbol will be stored in a charater variable.

**Example**

```
char c = 'µ'; // c will have the value 0xBC although the Unicode symbol  
is 0x3BC.
```

Package [java.lang](#)  
Class [StringBuffer](#)

**Public Method delete**

```
StringBuffer delete(int start, int end)
```

**Throws:**

- [StringIndexOutOfBoundsException](#)

**Description:**

Removes the specified substring from this StringBuffer. The substring begins at the index start and extends to the index end - 1. An exception is thrown when unused entries are deleted.

Package [java.lang](#)  
Class [StringBuffer](#)

**Public Method ensureCapacity**

```
void ensureCapacity(int minimumCapacity)
```

**Description:**

Ensures that the capacity of the buffer is at least equal to the specified minimum. If the current capacity of this string buffer is less than the argument, then a new internal buffer is allocated with greater capacity.

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method `getChars`**

```
void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

**Parameters**

- **srcBegin** – Index of first character to copy.
- **srcEnd** – Last character to copy is at srcEnd-1.
- **dst** – Destination character array.
- **dstBegin** – Destination array start index (characters will be copied to this location).

**Description:**

Copies the characters from this sequence into the destination character array dst. The first character to be copied is at index srcBegin. The last character to be copied is at index srcEnd - 1. The total number of characters to be copied is srcEnd - srcBegin. The characters are copied into the subarray of dst starting at index dstBegin and ending at index: dstbegin + (srcEnd-srcBegin) - 1.

Note that unicode symbols might occupy up to three entries in the string buffer. Be careful with the indices when using non ASCII symbols!

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method `insert`**

```
StringBuffer insert(int offset, boolean b)
```

```
StringBuffer insert(int offset, char c)
```

```
StringBuffer insert(int offset, char[] str)
```

**Throws:**

- [StringIndexOutOfBoundsException](#)

```
StringBuffer insert(int offset, float f)
```

```
StringBuffer insert(int offset, int i)
```

```
StringBuffer insert(int offset, Object obj)
```

```
StringBuffer insert(int offset, String str)
```

**Description:**

Inserts the string representation of a certain data type into this string buffer. The offset argument must be greater than or equal to 0, and less than or equal to the length of this sequence.

**insert(int offset, boolean b) method:**

This method inserts the string representation of the boolean argument into this sequence.

**insert(int offset, char c) method:**

This method inserts the string representation of the char argument into this sequence. Note that chars occupy only one byte, so take care when using Unicode symbols.

**insert(int offset, char[] str) method:**

This method inserts the string representation of the char array argument into this sequence. Note that chars occupy only one byte, so take care when using Unicode symbols.

**insert(int offset, float f) method:**

This method inserts the string representation of the float argument into this sequence.

**insert(int offset, int i) method:**

This method inserts the string representation of the second int argument into this sequence.

**insert(int offset, Object obj) method:**

This method inserts the string representation of the Object argument into this character sequence.

**insert(int offset, String str) method:**

This method inserts the string str into this character sequence.

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method length**

```
int length ()
```

**Description:**

Returns the character count of the sequence of characters currently represented by this string buffer object.

Note that unicode symbols might occupy up to three entries in the string buffer.

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method reverse**

```
StringBuffer reverse ()
```

**Description:**

Replace this character sequence with the reversed sequence.

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method `setCharAt`**

```
void setCharAt(int index, char ch)
```

**Description:**

The character at the specified index is set to ch.

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method `setLength`**

```
void setLength(int newLength)
```

**Description:**

Sets the length of the character sequence. If the newLength argument is greater than the current length, null characters are appended. If the newLength argument is less than the current length of the string buffer, the string buffer is truncated.

Package [java.lang](#)

Class [StringBuffer](#)

**Public Method `toString`**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

Returns a string representing the data in this sequence.

Package [java.lang](#)

**Class `StringIndexOutOfBoundsException`**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown by String methods to indicate that an index is either negative or greater than the size of the string.

**Public Constructors**

- [StringIndexOutOfBoundsException](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.lang](#)

## Class [StringIndexOutOfBoundsException](#)

### Public Constructor [StringIndexOutOfBoundsException](#)

```
StringIndexOutOfBoundsException()  
  
StringIndexOutOfBoundsException(int index)  
  
StringIndexOutOfBoundsException(String s)
```

#### Description:

##### constructor [StringIndexOutOfBoundsException\(\)](#):

Constructs an [StringIndexOutOfBoundsException](#) with no detail message.

##### constructor [StringIndexOutOfBoundsException\(int index\)](#):

Constructs an [StringIndexOutOfBoundsException](#) with an argument indicating the illegal index.

##### constructor [StringIndexOutOfBoundsException\(String s\)](#):

Constructs an [StringIndexOutOfBoundsException](#) with the specified detail message.

### Package [java.lang](#)

## Class System

extends [Object](#)

Contains utility methods.

This class cannot be instantiated.

### Public Methods

- [arraycopy](#)

- [currentTimeMillis](#)
- [freeMemory](#)
- [gc](#)
- [getProperty](#)
- [identityHashCode](#)
- [totalMemory](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

### Package [java.lang](#)

### Class [System](#)

#### Public Method [arraycopy](#)

```
static void arraycopy(Object src, int src_position, Object dst, int dst_position, int length)
```

#### Parameters

- **src** – source array.
- **src\_position** – starting position in the source array.
- **dst** – destination array.
- **dst\_position** – starting position in the destination data.
- **length** – number of array elements to be copied.

#### Description:

Copies length components from the specified source array, beginning at the specified position, to the specified position in the destination array.

If src or dst are null a system NullPointerException will occur. If src or dst are not arrays a system ArrayStoreException will occur. If src\_position, dst\_position or length are negative or the array size is exceeded a system IndexOutOfBoundsException-Excetion will occur.

Note that system exceptions are VM internal and cannot be caught.

### Package [java.lang](#)

### Class [System](#)

#### Public Method [currentTimeMillis](#)

```
static int currentTimeMillis()
```

#### Description:

Returns time in milliseconds since the start of the Java Application.



Note that in debug mode this time will not increase when the program is halted on a breakpoint. Use [IO.getTime\(\)](#) to get the value of the Real Time Clock.

Package [java.lang](#)  
Class [System](#)

**Public Method [freeMemory](#)**

```
static int freeMemory()
```

**Description:**

Returns the amount of free memory on the heap. Call [totalMemory](#) to get the amount of free memory.

Package [java.lang](#)  
Class [System](#)

**Public Method [gc](#)**

```
static void gc()
```

**Description:**

Force the garbage collector to run. The Java Virtual Machine will try to free any memory currently occupied by unused objects.

Package [java.lang](#)  
Class [System](#)

**Public Method [getProperty](#)**

```
static String getProperty(String key)  
static String getProperty(String key, String def)
```

**Parameters**

- **key** – property key.
- **def** – default value, returned if key wasn't found.

**Description:**

Returns specified system property.

Current properties are:

Key	Value
file.separator	"\"
path.separator	";"

line.separator	"\r\n"
----------------	--------

Package [java.lang](#)

Class [System](#)

### Public Method `identityHashCode`

```
static int identityHashCode(Object o)
```

#### Description:

Returns the hash code of the specified Object as would be returned from its ultimate Object parent, regardless of whether the passed-in object overrode the hashCode method. The hashcode for the null reference is zero.

Package [java.lang](#)

Class [System](#)

### Public Method `totalMemory`

```
static int totalMemory()
```

#### Description:

Returns the amount of total memory on the heap. Call [freeMemory](#) to get the amount of free memory.

Package [java.lang](#)

## Class Thread

extends [Object](#)

implements [Runnable](#)

Threads allow the execution of code in a seemingly parallel fashion. There are two ways to create a new thread of execution:

- Derive a subclass from Thread and overwrite the [run](#)-method. The tread can be started by calling the [start](#)-method.
- Declare a class that implements the Runnable interface. This also requires an implementation of the run-method. The thread can be started by passing an object of this class to a new Thread object and calling it's [start](#)-method.

### Examples

Method 1:

```
class MyThread extends Thread { // derive from
Thread
    public void run() { // implement run() -
method
MyThread obj = new MyThread(); // create new object
obj.start(); // start thread,
run() will be executed
```

## Method 2:

```
class class MyThread implements Runnable {           // derive from Thread
    public void run() {                               // implement run() -
method
MyThread obj = new MyThread();                       // create new object
new Thread(obj).start();                             // start thread,
run() will be executed
```

The total number of possible threads (including the always present main thread) is currently 9.

## Note

- The iLCD panel is a global resource and changing panel settings and attributes in different threads likely leads to non-deterministic behavior. It is generally not recommended to change iLCD panel properties outside the main thread.

## Public Constructors

- [Thread](#)

## Public Methods

- [currentThread](#)
- [currentThreadId](#)
- [getMaxThreadsCount](#)
- [interrupt](#)
- [interrupted](#)
- [isAlive](#)
- [isDaemon](#)
- [isInterrupted](#)
- [join](#)
- [resume](#)
- [run](#)
- [setDaemon](#)
- [sleep](#)
- [start](#)
- [stop](#)
- [suspend](#)
- [yield](#)

## Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

Class [Thread](#)

### **Public Constructor Thread**

```
Thread ()
```

```
Thread (Runnable target)
```

#### **Description:**

##### **constructor Thread():**

Creates a new Thread object.

##### **constructor Thread(Runnable target):**

Creates a new Thread object. The run()-method of the specified Runnable object will be invoked when the thread is started.

Package [java.lang](#)

Class [Thread](#)

### **Public Method currentThread**

```
static Thread currentThread ()
```

#### **Description:**

Returns a reference to the currently executing thread object.

Package [java.lang](#)

Class [Thread](#)

### **Public Method currentThreadId**

```
static int currentThreadId ()
```

#### **Description:**

Returns the ID of the to the currently executing thread object.

Package [java.lang](#)

Class [Thread](#)

### **Public Method getMaxThreadsCount**

```
static int getMaxThreadsCount ()
```

#### **Description:**

Returns the max number of possible threads including the main thread (always started).

Package [java.lang](#)  
Class [Thread](#)

**Public Method interrupt**

```
void interrupt()
```

**Description:**

Interrupts this thread.

Package [java.lang](#)  
Class [Thread](#)

**Public Method interrupted**

```
static boolean interrupted()
```

**Description:**

Checks whether the current thread has been interrupted clearing the current status, i.e. calling this method twice returns false for the second call.

Package [java.lang](#)  
Class [Thread](#)

**Public Method isAlive**

```
boolean isAlive()
```

**Description:**

Checks if this thread has been started and hasn't died yet.

Package [java.lang](#)  
Class [Thread](#)

**Public Method isDaemon**

```
final boolean isDaemon()
```

**Description:**

Check if this thread is a daemon thread. A daemon thread is a thread that provides background services to other user threads. It will be automatically terminated when all user threads died.

Package [java.lang](#)  
Class [Thread](#)

**Public Method isInterrupted**

```
boolean isInterrupted()
```

**Description:**

Checks whether the current thread has been interrupted without clearing the current status.

Package [java.lang](#)

Class [Thread](#)

**Public Method [join](#)**

```
final void join()
```

**Throws:**

- [InterruptedException](#)

```
final void join(int millis)
```

**Throws:**

- [InterruptedException](#)

**Description:**

Call this method to wait for this thread to die. If the parameter is specified wait at most millis milliseconds for the thread to die. A timeout of 0 means to wait forever.

Package [java.lang](#)

Class [Thread](#)

**Public Method [resume](#)**

```
final void resume()
```

**Description:**

Resumes a suspended thread. If the thread is alive but suspended, it is resumed and permitted to make progress in its execution.

Package [java.lang](#)

Class [Thread](#)

**Public Method [run](#)**

```
void run()
```

**Description:**

This method should either be overwritten by a derived class or this thread is started with a Runnable object which implements the run-method. If both is not the case, i.e. this thread is started without an alternative run()-method this method will simply do nothing and return.

Package [java.lang](#)  
Class [Thread](#)

**Public Method [setDaemon](#)**

```
final void setDaemon(boolean set)
```

**Description:**

Marks this thread as daemon or user thread. A daemon thread is a thread that provides background services to other user threads. It will be automatically terminated when all user threads died. This method must be invoked before the thread is started.

Package [java.lang](#)  
Class [Thread](#)

**Public Method [sleep](#)**

```
static void sleep(int millis)
```

**Throws:**

- [InterruptedException](#)

**Description:**

**[sleep\(int millis\)](#) method:**

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.

Package [java.lang](#)  
Class [Thread](#)

**Public Method [start](#)**

```
void start()
```

**Description:**

Begin the execution of this thread. The threads [run](#)-method will be called. The result is that two threads are running concurrently, the thread who called the start-method and the thread that has been started.

Package [java.lang](#)  
Class [Thread](#)

**Public Method [stop](#)**

```
final void stop()
```

**Description:**

Forces this thread to stop executing. Stopping a thread causes it to unlock all the monitors that it has locked. Note that any objects protected by these monitors can now be viewed by other threads.

It is recommended to replace uses of stop by code that uses a flag variable to indicate that the target thread should stop running. The target thread should check this variable regularly, and return from its run method in an orderly fashion.

Package [java.lang](#)

## Class [Thread](#)

### Public Method suspend

```
final void suspend()
```

#### Description:

Suspends this thread, i.e. when it is alive it will make no further progress until it is resumed.

Note that using suspend might lead to deadlocks because any resource protected by a monitor locked by this thread cannot be accessed by another thread until this thread is resumed. So if the thread that should resume this thread tries to lock such a monitor a deadlock situation will result.

Package [java.lang](#)

## Class [Thread](#)

### Public Method yield

```
static void yield()
```

#### Description:

Causes the currently executing thread object to temporarily pause and allow other threads to execute.

Package [java.lang](#)

## Class [Throwable](#)

extends [Object](#)

Superclass of all errors and exceptions. Only objects deriving from this class can be thrown with the throw-statement or caught with the catch-clause.

### Public Constructors

- [Throwable](#)

### Public Methods

- [fillInStackTrace](#)
- [getLocalizedMessage](#)
- [getMessage](#)
- [getStackTrace](#)
- [toString](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)



- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [Throwable](#)

### Public Constructor Throwable

```
Throwable ()
```

```
Throwable (String message)
```

#### Description:

##### constructor Throwable():

Constructs an Throwable with no detail message.

##### constructor Throwable(String s):

Constructs an Throwable with the specified detail message.

Package [java.lang](#)

## Class [Throwable](#)

### Public Method fillInStackTrace

```
void fillInStackTrace ()
```

#### Description:

Stores within this Throwable object information about the current state of the stack frames for the current thread. This method is useful when an application is re-throwing an error or exception.

Package [java.lang](#)

## Class [Throwable](#)

### Public Method getLocalizedMessage

```
String getLocalizedMessage ()
```

#### Description:

Subclasses may override this method in order to produce a locale-specific message. For subclasses that do not override this method, the default implementation returns the same result as [getMessage](#).

Package [java.lang](#)

## Class [Throwable](#)

### Public Method getMessage

```
String getMessage ()
```

**Description:**

Returns the error message string of this throwable object.

Package [java.lang](#)

Class [Throwable](#)

**Public Method `getStackTrace`**

```
void getStackTrace(byte[] Buffer)
```

**Description:**

Copies stack\_trace buffer into provided destination buffer. The number of copied bytes is limited by the size of given byte array Buffer or the size of stack\_trace buffer.

Package [java.lang](#)

Class [Throwable](#)

**Public Method `toString`**

```
String toString()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

Returns a short description of this throwable object.

Package [java.lang](#)

**Class `UnsupportedOperationException`**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown to indicate that the requested operation is not supported.

**Public Constructors**

- [UnsupportedOperationException](#)

**Methods inherited from [java.lang.Throwable](#)**

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)

- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [UnsupportedOperationException](#)

### Public Constructor UnsupportedOperationException

```
UnsupportedOperationException()
```

```
UnsupportedOperationException(String s)
```

#### Description:

##### constructor `UnsupportedOperationException()`:

Constructs an `UnsupportedOperationException` with no detail message.

##### constructor `UnsupportedOperationException(String s)`:

Constructs an `UnsupportedOperationException` with the specified detail message.

Package [java.lang](#)

## Class `VerifyError`

extends [LinkageError](#) → [Error](#) → [Throwable](#) → [Object](#)

Thrown when the "verifier" detects that a class file, though well formed, contains some sort of internal inconsistency or security problem.

### Public Constructors

- [VerifyError](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [VerifyError](#)

### Public Constructor VerifyError

```
VerifyError()
```

```
VerifyError(String s)
```

#### Description:

##### constructor VerifyError():

Constructs an VerifyError with no detail message.

##### constructor VerifyError(String s):

Constructs an VerifyError with the specified detail message.

Package [java.lang](#)

## Abstract Class VirtualMachineError

extends [Error](#) → [Throwable](#) → [Object](#)

Thrown to indicate that the Java Virtual Machine is broken or has run out of resources necessary for it to continue operating.

### Public Constructors

- [VirtualMachineError](#)

### Methods inherited from [java.lang.Throwable](#)

- [getMessage](#)
- [getLocalizedMessage](#)
- [toString](#)
- [fillInStackTrace](#)
- [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.lang](#)

## Class [VirtualMachineError](#)

### Public Constructor VirtualMachineError

```
VirtualMachineError()
```

**VirtualMachineError**(String s)

### Description:

#### constructor **VirtualMachineError()**:

Constructs an VirtualMachineError with no detail message.

#### constructor **VirtualMachineError(String s)**:

Constructs an VirtualMachineError with the specified detail message.

## Package java.util

Contains collections (data structures representing a group of objects) such as the sets, vectors, maps, hashes. It also contains iterators and utility classes (e.g. string tokenizer, random number generator).

## Interfaces

- [Collection](#)
- [Comparator](#)
- [Enumeration](#)
- [EventListener](#)
- [Iterator](#)
- [List](#)
- [ListIterator](#)
- [Map](#)
- [Map.Entry](#)
- [Set](#)
- [SortedMap](#)
- [SortedSet](#)

## Classes

- [AbstractCollection](#)
- [AbstractList](#)
- [AbstractMap](#)
- [AbstractSequentialList](#)
- [AbstractSet](#)
- [ArrayList](#)
- [Arrays](#)
- [BitSet](#)
- [Collections](#)
- [ConcurrentModificationException](#)
- [Date](#)
- [Dictionary](#)
- [EmptyStackException](#)
- [EventObject](#)
- [HashMap](#)
- [HashSet](#)
- [Hashtable](#)
- [LinkedList](#)
- [Locale](#)
- [MissingResourceException](#)
- [NoSuchElementException](#)

- [NotImplemented](#)
- [Random](#)
- [Stack](#)
- [StringTokenizer](#)
- [TooManyListenersException](#)
- [TreeMap](#)
- [TreeSet](#)
- [Vector](#)

## Package [java.util](#)

### Abstract Class [AbstractCollection](#)

extends [Object](#)

implements [Collection](#)

Basic implementation of the [Collection](#) interface.

To implement an unmodifiable collection, the programmer needs only to extend this class and provide implementations for the iterator and size methods.

To implement a modifiable collection, the programmer must additionally override this class's add method and the iterator returned by the iterator method must additionally implement its remove method.

#### Public Constructors

- [AbstractCollection](#)

#### Public Methods

- [add](#)
- [addAll](#)
- [clear](#)
- [contains](#)
- [containsAll](#)
- [isEmpty](#)
- [iterator](#)
- [remove](#)
- [removeAll](#)
- [retainAll](#)
- [size](#)
- [toArray](#)
- [toString](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Constructor AbstractCollection

```
AbstractCollection()
```

#### Description:

The only constructor of this class.

Note that generally two constructors should be provided when implementing the collection interface. One with no argument (empty collection) and one one with a single argument of type [Collection](#), which creates a new collection with the same elements. (see Interface [Collection](#))

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method add

```
boolean add(Object o)
```

#### Throws:

- [UnsupportedOperationException](#)

#### Description:

Implements interface method [Collection.add](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method addAll

```
boolean addAll(Collection c)
```

#### Description:

Implements interface method [Collection.addAll](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method clear

```
void clear()
```

#### Description:

Implements interface method [Collection.clear](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method contains

```
boolean contains(Object o)
```

#### Description:

Implements interface method [Collection.contains](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method containsAll

```
boolean containsAll(Collection c)
```

#### Description:

Implements interface method [Collection.containsAll](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method isEmpty

```
boolean isEmpty()
```

#### Description:

Implements interface method [Collection.isEmpty](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method iterator

```
abstract Iterator iterator()
```

#### Description:

Implements interface method [Collection.iterator](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method remove

```
boolean remove(Object o)
```

#### Description:

Implements interface method [Collection.remove](#).



Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method removeAll

```
boolean removeAll(Collection c)
```

#### Description:

Implements interface method [Collection.removeAll](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method retainAll

```
boolean retainAll(Collection c)
```

#### Description:

Implements interface method [Collection.retainAll](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method size

```
abstract int size()
```

#### Description:

Implements interface method [Collection.size](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method toArray

```
Object[] toArray()
```

#### Description:

Implements interface method [Collection.toArray](#).

Package [java.util](#)

## Class [AbstractCollection](#)

### Public Method toString

```
String toString()
```

#### Overrides:

- [toString](#) in class [Object](#)

## Description:

Returns a string representation of this collection. The string representation consists of a list of the collection's elements in the order they are returned by its iterator, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Elements are converted to strings as by `String.valueOf(Object)`.

This implementation creates an empty string buffer, appends a left square bracket, and iterates over the collection appending the string representation of each element in turn. After appending each element except the last, the string ", " is appended. Finally a right bracket is appended. A string is obtained from the string buffer, and returned.

## Package [java.util](#)

### Abstract Class `AbstractList`

extends [AbstractCollection](#) → [Object](#)

implements [List](#), [Collection](#)

Basic implementation of the [List](#) interface for random access (such as array). For sequential access (such as a linked list) use the class `AbstractSequentialList`.

To implement an unmodifiable list, the programmer needs only to extend this class and provide implementations for the `get(int index)` and `size()` methods.

To implement a modifiable list, the programmer must additionally override the `set(int index, Object element)`. If the list is variable-size the programmer must additionally override the `add(int index, Object element)` and `remove(int index)` methods.

Unlike the other abstract collection implementations, the programmer does not have to provide an iterator implementation; the iterator and list iterator are implemented by this class, on top the "random access" methods: `get(int index)`, `set(int index, Object element)`, `set(int index, Object element)`, `add(int index, Object element)` and `remove(int index)`.

### Public Constructors

- [AbstractList](#)

### Public Methods

- [add](#)
- [addAll](#)
- [clear](#)
- [equals](#)
- [get](#)
- [hashCode](#)
- [indexOf](#)
- [iterator](#)
- [lastIndexOf](#)
- [listIterator](#)
- [remove](#)
- [set](#)
- [subList](#)

### Methods inherited from [java.util.AbstractCollection](#)

- [iterator](#)

- [size](#)
- [add](#)
- [addAll](#)
- [clear](#)
- [contains](#)
- [containsAll](#)
- [isEmpty](#)
- [remove](#)
- [removeAll](#)
- [retainAll](#)
- [toArray](#)
- [toString](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.util](#)

### Class [AbstractList](#)

#### Public Constructor [AbstractList](#)

```
AbstractList()
```

#### Description:

The only constructor of this class.

Note that generally two constructors should be provided when implementing the collection interface. One with no argument (empty collection) and one one with a single argument of type [Collection](#), which creates a new collection with the same elements. (see Interface [Collection](#))

#### Package [java.util](#)

### Class [AbstractList](#)

#### Public Method [add](#)

```
void add(int index, Object o)
```

#### Throws:

- [UnsupportedOperationException](#)

```
boolean add(Object o)
```

#### Overrides:

- [add](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.add](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method addAll**

```
boolean addAll(int index, Collection c)
```

**Description:**

Implements interface method [List.addAll](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method clear**

```
void clear()
```

**Overrides:**

- [clear](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.clear](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method equals**

```
boolean equals(Object o)
```

**Overrides:**

- [equals](#) in class [Object](#)

**Description:**

Implements interface method [List.equals](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method get**

```
abstract Object get(int index)
```

**Description:**

Implements interface method [List.get](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method hashCode**

```
int hashCode ()
```

**Overrides:**

- [hashCode](#) in class [Object](#)

**Description:**

Implements interface method [List.hashCode](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method indexOf**

```
int indexOf (Object o)
```

**Description:**

Implements interface method [List.indexOf](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method iterator**

```
Iterator iterator ()
```

**Overrides:**

- [iterator](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.iterator](#).

Package [java.util](#)

Class [AbstractList](#)

**Public Method lastIndexOf**

```
int lastIndexOf (Object o)
```

**Description:**

Implements interface method [List.lastIndexOf](#).

Package [java.util](#)

**Class [AbstractList](#)****Public Method listIterator**

```
ListIterator listIterator()
```

```
ListIterator listIterator(int index)
```

**Throws:**

- [IndexOutOfBoundsException](#)

**Description:**

Implements interface method [List.listIterator](#).

Package [java.util](#)

**Class [AbstractList](#)****Public Method remove**

```
Object remove(int index)
```

**Throws:**

- [UnsupportedOperationException](#)

**Description:**

Implements interface method [List.remove](#).

Package [java.util](#)

**Class [AbstractList](#)****Public Method set**

```
Object set(int index, Object o)
```

**Throws:**

- [UnsupportedOperationException](#)

**Description:**

Implements interface method [List.set](#).

**Package [java.util](#)****Class [AbstractList](#)****Public Method [subList](#)**

```
List subList(int fromIndex, int toIndex)
```

**Throws:**

- [IllegalArgumentException](#)
- [IndexOutOfBoundsException](#)

**Description:**

Implements interface method [List.subList](#).

**Package [java.util](#)****Abstract Class [AbstractMap](#)**

extends [Object](#)

implements [Map](#)

Basic implementation of the Map interface.

To implement an unmodifiable map, the programmer needs only to extend this class and provide an implementation for the `entrySet` method, which returns a set-view of the map's mappings. Typically, the returned set will, in turn, be implemented atop `AbstractSet`. This set should not support the add or remove methods, and its iterator should not support the remove method.

To implement a modifiable map, the programmer must additionally override this class's `put` method (which otherwise throws an `UnsupportedOperationException`), and the iterator returned by `entrySet().iterator()` must additionally implement its remove method.

**Public Methods**

- [clear](#)
- [clone](#)
- [containsKey](#)
- [containsValue](#)
- [entrySet](#)
- [equals](#)
- [get](#)
- [hashCode](#)
- [isEmpty](#)
- [keySet](#)
- [put](#)
- [putAll](#)
- [remove](#)
- [size](#)
- [toString](#)
- [values](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)

- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method clear

```
void clear ()
```

#### Description:

Implements interface method [Map.clear](#).

This implementation calls `entrySet().clear()`.

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method clone

```
Object clone ()
```

#### Overrides:

- [clone](#) in class [Object](#)

#### Throws:

- [Error](#)

#### Description:

Returns new cloned object.

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method containsKey

```
boolean containsKey (Object key)
```

#### Description:

Implements interface method [Map.containsKey](#).

This implementation iterates over `entrySet()` searching for an entry with the specified key. If such an entry is found, `true` is returned. If the iteration terminates without finding such an entry, `false` is returned. Note that this implementation requires linear time in the size of the map; many implementations will override this method.



Package [java.util](#)

## Class [AbstractMap](#)

### Public Method containsValue

```
boolean containsValue(Object value)
```

#### Description:

Implements interface method [Map.containsValue](#).

This implementation iterates over `entrySet()` searching for an entry with the specified value. If such an entry is found, `true` is returned. If the iteration terminates without finding such an entry, `false` is returned. Note that this implementation requires linear time in the size of the map.

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method entrySet

```
abstract Set entrySet()
```

#### Description:

Implements interface method [Map.entrySet](#).

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method equals

```
boolean equals(Object o)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

Implements interface method [Map.equals](#).

This implementation first checks if the specified object is this map; if so it returns `true`. Then, it checks if the specified object is a map whose size is identical to the size of this set; if not, it returns `false`. If so, it iterates over this map's `entrySet` collection, and checks that the specified map contains each mapping that this map contains. If the specified map fails to contain such a mapping, `false` is returned. If the iteration completes, `true` is returned.

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method get

```
Object get(Object key)
```

**Description:**

Implements interface method [Map.get](#).

This implementation iterates over `entrySet()` searching for an entry with the specified key. If such an entry is found, the entry's value is returned. If the iteration terminates without finding such an entry, null is returned. Note that this implementation requires linear time in the size of the map; many implementations will override this method.

Package [java.util](#)

**Class [AbstractMap](#)****Public Method [hashCode](#)**

```
int hashCode ()
```

**Overrides:**

- [hashCode](#) in class [Object](#)

**Description:**

Implements interface method [Map.hashCode](#).

This implementation iterates over `entrySet()`, calling `hashCode` on each element (entry) in the Collection, and adding up the results.

Package [java.util](#)

**Class [AbstractMap](#)****Public Method [isEmpty](#)**

```
boolean isEmpty ()
```

**Description:**

Implements interface method [Map.isEmpty](#).

Package [java.util](#)

**Class [AbstractMap](#)****Public Method [keySet](#)**

```
Set keySet ()
```

**Description:**

Implements interface method [Map.keySet](#).

This implementation returns a Set that subclasses `AbstractSet`. The subclass's iterator method returns a "wrapper object" over this map's `entrySet()` iterator. The size method delegates to this map's size method and the contains method delegates to this map's `containsKey` method.

The Set is created the first time this method is called, and returned in response to all subsequent calls. No synchronization is performed, so there is a slight chance that multiple calls to this method will not all return the same Set.

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method put

```
Object put(Object key, Object value)
```

#### Throws:

- [UnsupportedOperationException](#)

#### Description:

Implements interface method [Map.put](#).

This implementation always throws an [UnsupportedOperationException](#).

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method putAll

```
void putAll(Map t)
```

#### Description:

Implements interface method [Map.putAll](#).

This implementation iterates over the specified map's `entrySet()` collection, and calls this map's `put` operation once for each entry returned by the iteration.

Package [java.util](#)

## Class [AbstractMap](#)

### Public Method remove

```
Object remove(Object key)
```

#### Description:

Implements interface method [Map.remove](#).

This implementation iterates over `entrySet()` searching for an entry with the specified key. If such an entry is found, its value is obtained with its `getValue` operation, the entry is removed from the Collection (and the backing map) with the iterator's `remove` operation, and the saved value is returned. If the iteration terminates without finding such an entry, `null` is returned. Note that this implementation requires linear time in the size of the map; many implementations will override this method.

Package [java.util](#)  
Class [AbstractMap](#)

**Public Method size**

```
int size ()
```

**Description:**

Implements interface method [Map.size](#).

This implementation returns `entrySet().size()`.

Package [java.util](#)  
Class [AbstractMap](#)

**Public Method toString**

```
String toString ()
```

**Overrides:**

- [toString](#) in class [Object](#)

**Description:**

Returns a string representation of this map. The string representation consists of a list of key-value mappings in the order returned by the map's `entrySet` view's iterator, enclosed in braces ("`{}`"). Adjacent mappings are separated by the characters ", " (comma and space). Each key-value mapping is rendered as the key followed by an equals sign ("`=`") followed by the associated value. Keys and values are converted to strings as by `String.valueOf(Object)`.

This implementation creates an empty string buffer, appends a left brace, and iterates over the map's `entrySet` view, appending the string representation of each `map.entry` in turn. After appending each entry except the last, the string ", " is appended. Finally a right brace is appended. A string is obtained from the stringbuffer, and returned.

Package [java.util](#)  
Class [AbstractMap](#)

**Public Method values**

```
Collection values ()
```

**Description:**

Implements interface method [Map.values](#).

This implementation returns a collection that subclasses abstract collection. The subclass's iterator method returns a "wrapper object" over this map's `entrySet()` iterator. The size method delegates to this map's size method and the contains method delegates to this map's `containsValue` method.

The collection is created the first time this method is called, and returned in response to all subsequent calls. No synchronization is performed, so there is a slight chance that multiple calls to this method will not all return the same Collection.

## Package [java.util](#)

### Abstract Class [AbstractSequentialList](#)

extends [AbstractList](#) → [AbstractCollection](#) → [Object](#)

implements [List](#), [Collection](#) p> Basic implementation of the [List](#) interface for sequential access lists (such as linked lists).

To implement a list the programmer needs only to extend this class and provide implementations for the listIterator and size methods. For an unmodifiable list, the programmer need only implement the list iterator's hasNext, next, hasPrevious, previous and index methods.

For a modifiable list the programmer should additionally implement the list iterator's set method. For a variable-size list the programmer should additionally implement the list iterator's remove and add methods.

#### Public Constructors

- [AbstractSequentialList](#)

#### Public Methods

- [add](#)
- [addAll](#)
- [get](#)
- [iterator](#)
- [listIterator](#)
- [remove](#)
- [set](#)

#### Methods inherited from [java.util.AbstractList](#)

- [get](#)
- [add](#)
- [addAll](#)
- [clear](#)
- [equals](#)
- [hashCode](#)
- [indexOf](#)
- [iterator](#)
- [lastIndexOf](#)
- [listIterator](#)
- [remove](#)
- [set](#)
- [subList](#)

#### Methods inherited from [java.util.AbstractCollection](#)

- [iterator](#)
- [size](#)
- [add](#)
- [addAll](#)
- [clear](#)
- [contains](#)
- [containsAll](#)
- [isEmpty](#)
- [remove](#)

- [removeAll](#)
- [retainAll](#)
- [toArray](#)
- [toString](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.util](#)

### Class [AbstractSequentialList](#)

#### Public Constructor AbstractSequentialList

```
AbstractSequentialList()
```

#### Package [java.util](#)

### Class [AbstractSequentialList](#)

#### Public Method add

```
void add(int index, Object o)
```

#### Overrides:

- [add](#) in class [AbstractList](#)

#### Description:

Inserts the specified element at the specified position in this list. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

This implementation first gets a list iterator pointing to the indexed element (with `listIterator(index)`). Then, it inserts the specified element with `ListIterator.add`.

#### Package [java.util](#)

### Class [AbstractSequentialList](#)

#### Public Method addAll

```
boolean addAll(int index, Collection c)
```

#### Overrides:

- [addAll](#) in class [AbstractList](#)

**Description:**

Inserts all of the elements in the specified collection into this list at the specified position. Shifts the elements currently at that position to the right (increases their indices). The new elements will appear in the list in the order that they are returned by the specified collection's iterator.

The behavior of this operation is unspecified if the specified collection is modified while the operation is in progress. (Note that this will occur if the specified collection is this list, and it's nonempty.)

This implementation gets an iterator over the specified collection and a list iterator over this list pointing to the indexed element (with `listIterator(index)`). Then, it iterates over the specified collection, inserting the elements obtained from the iterator into this list, one at a time, using `ListIterator.add` followed by `ListIterator.next` (to skip over the added element).

Package [java.util](#)

**Class [AbstractSequentialList](#)****Public Method `get`**

```
Object get(int index)
```

**Overrides:**

- [get](#) in class [AbstractList](#)

**Throws:**

- [IndexOutOfBoundsException](#)

**Description:**

Returns the element at the specified position in this list.

This implementation first gets a list iterator pointing to the indexed element (with `listIterator(index)`). Then, it gets the element using `ListIterator.next` and returns it.

Package [java.util](#)

**Class [AbstractSequentialList](#)****Public Method `iterator`**

```
Iterator iterator()
```

**Overrides:**

- [iterator](#) in class [AbstractList](#)

**Description:**

Returns an iterator over the elements in this list (in proper sequence).

This implementation merely returns a list iterator over the list.

Package [java.util](#)

## Class [AbstractSequentialList](#)

### Public Method listIterator

```
abstract ListIterator listIterator(int index)
```

#### Overrides:

- [listIterator](#) in class [AbstractList](#)

#### Description:

Returns a list iterator over the elements in this list (in proper sequence).

Package [java.util](#)

## Class [AbstractSequentialList](#)

### Public Method remove

```
Object remove(int index)
```

#### Overrides:

- [remove](#) in class [AbstractList](#)

#### Throws:

- [IndexOutOfBoundsException](#)

#### Description:

Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices).

This implementation first gets a list iterator pointing to the indexed element (with `listIterator(index)`). Then, it removes the element with `ListIterator.remove`.

Package [java.util](#)

## Class [AbstractSequentialList](#)

### Public Method set

```
Object set(int index, Object o)
```

#### Overrides:

- [set](#) in class [AbstractList](#)

#### Throws:

- [IndexOutOfBoundsException](#)



**Description:**

Replaces the element at the specified position in this list with the specified element.

This implementation first gets a list iterator pointing to the indexed element (with `listIterator(index)`). Then, it gets the current element using `ListIterator.next` and replaces it with `ListIterator.set`.

**Package [java.util](#)****Abstract Class [AbstractSet](#)**

extends [AbstractCollection](#) → [Object](#)

implements [Set](#), [Collection](#)

Basic implementation of the [Set](#) interface.

The process of implementing a set by extending this class is identical to that of implementing a [Collection](#) by extending [AbstractCollection](#), except that all of the methods and constructors in subclasses of this class must obey the additional constraints imposed by the [Set](#) interface (for instance, the `add` method must not permit addition of multiple instances of an object to a set).

Note that this class does not override any of the implementations from the [AbstractCollection](#) class. It merely adds implementations for `equals` and `hashCode`.

**Public Constructors**

- [AbstractSet](#)

**Public Methods**

- [equals](#)
- [hashCode](#)

**Methods inherited from [java.util.AbstractCollection](#)**

- [iterator](#)
- [size](#)
- [add](#)
- [addAll](#)
- [clear](#)
- [contains](#)
- [containsAll](#)
- [isEmpty](#)
- [remove](#)
- [removeAll](#)
- [retainAll](#)
- [toArray](#)
- [toString](#)

**Methods inherited from [java.lang.Object](#)**

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

Package [java.util](#)

## Class [AbstractSet](#)

### Public Constructor AbstractSet

```
AbstractSet ()
```

#### Description:

The only constructor of this class.

Package [java.util](#)

## Class [AbstractSet](#)

### Public Method equals

```
boolean equals (Object o)
```

#### Overrides:

- [equals](#) in class [Object](#)

#### Description:

Compares the specified object with this set for equality. Returns true if the given object is also a set, the two sets have the same size, and every member of the given set is contained in this set.

This implementation first checks if the specified object is this set; if so it returns true. Then, it checks if the specified object is a set whose size is identical to the size of this set; if not, it returns false. If so, it returns `containsAll((Collection) o)`.

Package [java.util](#)

## Class [AbstractSet](#)

### Public Method hashCode

```
int hashCode ()
```

#### Overrides:

- [hashCode](#) in class [Object](#)

#### Description:

Returns the hash code value for this set. The hash code of a set is defined to be the sum of the hash codes of the elements in the set. This ensures that `s1.equals(s2)` implies that `s1.hashCode()==s2.hashCode()` for any two sets `s1` and `s2`, as required by the general contract of `Object.hashCode`.

This implementation enumerates over the set, calling the `hashCode` method on each element in the collection, and adding up the results.

## Package [java.util](#)

### Class `ArrayList`

extends [AbstractList](#) → [AbstractCollection](#) → [Object](#)  
implements [List](#), [Cloneable](#), [Serializable](#), [Collection](#)

Resizable-array implementation of the *List* interface. Implements all optional list operations, and permits all elements, including *null*. In addition to implementing the *List* interface, this class provides methods to manipulate the size of the array that is used internally to store the list. This class is roughly equivalent to *Vector*, except that it is unsynchronized.

Each *ArrayList* instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added an *ArrayList*, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

An application can increase the capacity of an *ArrayList* instance before adding a large number of elements using the *ensureCapacity* operation. This may reduce the amount of incremental reallocation.

#### Public Constructors

- [ArrayList](#)

#### Public Methods

- [add](#)
- [addAll](#)
- [clear](#)
- [clone](#)
- [contains](#)
- [ensureCapacity](#)
- [get](#)
- [indexOf](#)
- [isEmpty](#)
- [lastIndexOf](#)
- [remove](#)
- [set](#)
- [size](#)
- [toArray](#)
- [trimToSize](#)

#### Methods inherited from [java.util.AbstractList](#)

- [get](#)
- [add](#)
- [addAll](#)
- [clear](#)
- [equals](#)
- [hashCode](#)
- [indexOf](#)
- [iterator](#)
- [lastIndexOf](#)
- [listIterator](#)
- [remove](#)
- [set](#)

- [subList](#)

#### Methods inherited from [java.util.AbstractCollection](#)

- [iterator](#)
- [size](#)
- [add](#)
- [addAll](#)
- [clear](#)
- [contains](#)
- [containsAll](#)
- [isEmpty](#)
- [remove](#)
- [removeAll](#)
- [retainAll](#)
- [toArray](#)
- [toString](#)

#### Methods inherited from [java.lang.Object](#)

- [equals](#)
- [toString](#)
- [wait](#)
- [hashCode](#)
- [notify](#)
- [notifyAll](#)

#### Package [java.util](#)

### Class [ArrayList](#)

#### Public Constructor ArrayList

```
ArrayList ()
```

```
ArrayList (Collection c)
```

```
ArrayList (int initialCapacity)
```

#### Throws:

- [IllegalArgumentException](#)

#### Description:

##### **constructor ArrayList():**

Constructs an empty list.

##### **constructor ArrayList(Collection c):**

Constructs a list containing the elements of the specified collection, in order they are returned by the collection's iterator. The *ArrayList* instance has an initial capacity of 110% the size of the specified collection.

**constructor ArrayList(int initialCapacity):**

Constructs an empty list with specified initial capacity.

Package [java.util](#)

Class [ArrayList](#)

**Public Method add**

```
void add(int index, Object element)
```

**Overrides:**

- [add](#) in class [AbstractList](#)

**Throws:**

- [IndexOutOfBoundsException](#)

```
boolean add(Object element)
```

**Overrides:**

- [add](#) in class [AbstractList](#)

**Description:****add(int index, Object element) method:**

Inserts the specified element at the specified position in this list. Shifts the element currently at that position if any and any subsequent elements to the right.

**add(Object element) method:**

Inserts the specified element at the specified position in this list. Shifts the element currently at that position if any and any subsequent elements to the right.

**Example**

```
String element = "This is the second element!";
ArrayList aList = new ArrayList(10);
aList.add(element);
```

The example above adds a string element to the ArrayList.

Package [java.util](#)

Class [ArrayList](#)

**Public Method addAll**

```
boolean addAll(Collection c)
```

**Overrides:**

- [addAll](#) in class [AbstractCollection](#)

```
boolean addAll(int index, Collection c)
```

**Overrides:**

- [addAll](#) in class [AbstractList](#)

**Throws:**

- [IndexOutOfBoundsException](#)

**Description:****addAll(Collection c) method:**

Appends all of the elements in the specified Collection to the end of this list, in order that they are returned by the specified Collection's Iterator. The behavior of this operation is undefined if the specified Collection is modified while the operation is in progress. This implies that the behavior of this call is undefined if the specified Collection is this list, and this list is nonempty.

**addAll(int index, Collection) method:**

Inserts all of the elements in the specified Collection into this list, starting at the specified position. Shifts the element currently at that position if any and any subsequent elements to the right. The new element will appear in the list in the order that they are returned by the specified Collection's iterator.

**Example**

```
ArrayList aList = new ArrayList(5);
aList.add("A The first element!");
aList.add("A The second element!");
aList.add("A The third element!");

for (int i = 0; i < 3; i++)
{
    Logger.log("" + aList.get(i));
}

ArrayList bList = new ArrayList(2);
bList.add("B the second element!");
bList.add("B the third element!");

aList.addAll(1, bList);

for (int i = 0; i < 5; i++)
{
    Logger.log("" + aList.get(i));
}
```

The example moves the second and third element of the *aList* to the right and inserts the elements of the *bList*.

Package [java.util](#)

Class [ArrayList](#)

**Public Method `clear`**

```
void clear()
```

**Overrides:**

- [clear](#) in class [AbstractList](#)

**Description:**

Removes all of the elements from this list. The list will be empty after this call returns.

**Example**

```
ArrayList aList = new ArrayList(10);  
aList.add("This is the first element!");  
aList.clear();
```

The `clear()` method removes the added element from the list.

Package [java.util](#)

Class [ArrayList](#)

**Public Method `clone`**

```
Object clone()
```

**Overrides:**

- [clone](#) in class [Object](#)

**Throws:**

- [Error](#)

**Description:**

Returns a shallow copy of this *ArrayList* instance. The elements themselves are not copied.

**Example**

```
ArrayList aList = new ArrayList(2);  
aList.add("A The first element!");  
aList.add("A The second element!");  
  
ArrayList bList = (ArrayList)aList.clone();  
for (int i = 0; i < 2; i++)
```

```
{
    Logger.log("List A element: " + aList.get(i) + " List B element: "
+ aList.get(i));
}
```

The example above copies the *ArrayList* instance.

Package [java.util](#)  
Class [ArrayList](#)

**Public Method contains**

```
boolean contains(Object elem)
```

**Overrides:**

- [contains](#) in class [AbstractCollection](#)

**Description:**

Returns *true* if this list contains the specified element.

**Example**

```
String text = "contains";
String text2 = "contains";
ArrayList aList = new ArrayList(2);
aList.add(text);
if (aList.contains(text2))
{
    Logger.log("This list contains this element!");
}
```

A message is written to the logger if the string "*contains*" is contained into the *aList*.

Package [java.util](#)  
Class [ArrayList](#)

**Public Method ensureCapacity**

```
void ensureCapacity(int minCapacity)
```

**Throws:**

[IllegalArgumentException](#)

**Description:**

Increases the capacity of this *ArrayList* instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.



## Example

```
ArrayList aList = new ArrayList(1);
aList.ensureCapacity(2);
```

The example above ensures the capacity of this instance.

## Package [java.util](#) Class [ArrayList](#)

### Public Method [get](#)

```
Object get(int index)
```

#### Overrides:

↳ [get](#) in class [AbstractList](#)

#### Throws:

↳ [IndexOutOfBoundsException](#)

#### Description:

Returns the element at the specified position in this list.

## Example

```
Logger.log("" + aList.get(1);
```

The example above writes the second element to the logger.

## Package [java.util](#) Class [ArrayList](#)

### Public Method [indexOf](#)

```
int indexOf(Object elem)
```

#### Overrides:

↳ [indexOf](#) in class [AbstractList](#)

#### Description:

Searchs for the first occurence of the given argument, testing for equality using the *equals* method.

## Example

```
Logger.log("The index of the element: " + aList.indexOf(element));
```

The example above adds a string element to the ArrayList.

Package [java.util](#)  
Class [ArrayList](#)

**Public Method isEmpty**

```
boolean isEmpty()
```

**Overrides:**

⇒ [isEmpty](#) in class [AbstractCollection](#)

**Description:**

Tests if this list has no elements.

**Example**

```
ArrayList bList = new ArrayList(5);
if (bList.isEmpty())
{
    Logger.log("This ArrayList is empty!");
}
```

The `bList.isEmpty()` method returns true if the list is empty.

Package [java.util](#)  
Class [ArrayList](#)

**Public Method lastIndexOf**

```
int lastIndexOf(Object elem)
```

**Overrides:**

⇒ [lastIndexOf](#) in class [AbstractList](#)

**Description:**

Returns the index of the last occurrence of the specified object in this list.

Package [java.util](#)  
Class [ArrayList](#)

**Public Method remove**

```
Object remove(int index)
```

**Overrides:**

⇒ [remove](#) in class [AbstractList](#)

**Description:**

Removes the element at the specified position in this list. Shifts any subsequent elements to the left.

## Example

```
aList.remove(3);
```

The example above removes the specified element from the *ArrayList*.

### Package [java.util](#) Class [ArrayList](#)

#### **Public Method set**

```
Object set(int index, Object element)
```

#### **Overrides:**

⇒ [set](#) in class [AbstractList](#)

#### **Throws:**

⇒ [IndexOutOfBoundsException](#)

#### **Description:**

Replaces the element at the specified position in this list with the specified element.

## Example

```
aList.set(3, element);
```

The example above replaces 4 element with the specified *element*.

### Package [java.util](#) Class [ArrayList](#)

#### **Public Method size**

```
int size()
```

#### **Overrides:**

⇒ [size](#) in class [AbstractCollection](#)

#### **Description:**

Returns the number of elements in this list.

## Example

```
ArrayList aList = new ArrayList(5);  
Logger.log("number of currently stored elements in the ArrayList: " +  
aList.size());
```

The example above returns the number of elements in the *ArrayList*.

## Package [java.util](#)

### Class [ArrayList](#)

#### Public Method [toArray](#)

```
Object[] toArray()
```

#### Overrides:

↳ [toArray](#) in class [AbstractCollection](#)

#### Description:

Returns an array containing all of the elements in this list in the correct order.

## Package [java.util](#)

### Class [ArrayList](#)

#### Public Method [trimToSize](#)

```
void trimToSize()
```

#### Description:

Trims the capacity of this *ArrayList* instance to be the list's current size. An application can use this operation to minimize the storage of an *ArrayList* instance.

## Package [java.util](#)

### Class [Arrays](#)

extends [Object](#)

This class contains various methods for manipulating arrays (such as sorting and searching). It also contains a static factory that allows arrays to be viewed as lists.

The documentation for the sorting and searching methods contained in this class includes briefs description of the implementations. Such descriptions should be regarded as implementation notes, rather than parts of the specification. Implementors should feel free to substitute other algorithms, so long as the specification itself is adhered to.

#### Public Methods

- ↳ [asList](#)
- ↳ [binarySearch](#)
- ↳ [equals](#)
- ↳ [fill](#)
- ↳ [sort](#)

#### Methods inherited from [java.lang.Object](#)

- ↳ [equals](#)
- ↳ [toString](#)
- ↳ [wait](#)
- ↳ [hashCode](#)
- ↳ [notify](#)
- ↳ [notifyAll](#)

Package [java.util](#)  
Class [Arrays](#)

### **Public Method `asList`**

```
static List asList(Object[] a)
```

#### **Description:**

Returns a fixed-size list backed by the specified array. Changes to the returned list "write through" to the array. This method acts as bridge between array-based and collection-based APIs, in combination with *Collection.toArray*. The returned list is serializable.

#### **Example**

```
String str[] = {"Hello", "World"};  
List stringList = Arrays.asList(str);
```

The example above returns the list.

Package [java.util](#)  
Class [Arrays](#)

### **Public Method `binarySearch`**

```
static int binarySearch(byte[] a, byte key)  
  
static int binarySearch(char[] a, char key)  
  
static int binarySearch(float[] a, float key)  
  
static int binarySearch(int[] a, int key)  
  
static int binarySearch(Object[] a, Object key)  
  
static int binarySearch(Object[] a, Object key, Comparator c)  
  
static int binarySearch(short[] a, short key)
```

#### **Description:**

##### **`binarySearch(byte[] a, byte key)` method:**

Searches the specified array of bytes for the specified value using the binary search algorithm. The array must be sorted prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

##### **`binarySearch(char[] a, char key)` method:**

Searches the specified array of chars for the specified value using the binary search algorithm. The array must be sorted prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

**binarySearch(float[] a, float key) method:**

Searches the specified array of floats for the specified value using the binary search algorithm. The array must be sorted prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

**binarySearch(int[] a, int key) method:**

Searches the specified array of ints for the specified value using the binary search algorithm. The array must be sorted prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

**binarySearch(Object[] a, Object key) method:**

Searches the specified array of Objects for the specified value using the binary search algorithm. The array must be sorted into ascending order according to the natural ordering of its elements prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements equal to the specified object, there is no guarantee which one will be found.

**binarySearch(Object[] a, Object key, Comparator c) method:**

Searches the specified array of Objects for the specified value using the binary search algorithm. The array must be sorted into ascending order according to the specified comparator prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements equal to the specified object, there is no guarantee which one will be found.

**binarySearch(short[] a, short key) method:**

Searches the specified array of shorts for the specified value using the binary search algorithm. The array must be sorted prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found.

**Example**

```
int intArr[] = {30,20,5,12,55};
Arrays.sort(intArr);
Logger.log("The index of element 12 is : " +
Arrays.binarySearch(intArr,12));
```

The example above returns the index value of element and writes it to the logger.

**Package [java.util](#)****Class [Arrays](#)****Public Method equals**

```
static boolean equals(boolean[] a, boolean[] a2)
static boolean equals(byte[] a, byte[] a2)
static boolean equals(char[] a, char[] a2)
static boolean equals(float[] a, float[] a2)
```

```
static boolean equals(int[] a, int[] a2)

static boolean equals(Object[] a, Object[] a2)

static boolean equals(short[] a, short[] a2)
```

### Description:

#### **equals(boolean[] a, boolean[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

#### **equals(byte[] a, byte[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

#### **equals(char[] a, char[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

#### **equals(float[] a, float[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

#### **equals(int[] a, int[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

#### **equals(Object[] a, Object[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

#### **equals(short[] a, short[] a2) method:**

Returns *true* if the two specified arrays of equals are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of

elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are *null*.

### Example

```
boolean bool1[] = {true, false};
boolean bool2[] = {ture, false};
if (Arrays.equals(bool1, bool2))
    Logger.log("bool1 equals bool2");
```

The *Arrays.equals* method returns the *boolean* value *true* if the two arrays equals to one another.

### Package [java.util](#)

## Class [Arrays](#)

### Public Method fill

```
static void fill(boolean[] a, boolean val)
static void fill(boolean[] a, int fromIndex, int toIndex, boolean val)
static void fill(byte[] a, byte val)
static void fill(byte[] a, int fromIndex, int toIndex, byte val)
static void fill(char[] a, char val)
static void fill(char[] a, int fromIndex, int toIndex, char val)
static void fill(float[] a, float val)
static void fill(float[] a, int fromIndex, int toIndex, float val)
static void fill(int[] a, int val)
static void fill(int[] a, int fromIndex, int toIndex, int val)
static void fill(Object[] a, int fromIndex, int toIndex, Object val)
static void fill(Object[] a, Object val)
static void fill(short[] a, int fromIndex, int toIndex, short val)
static void fill(short[] a, short val)
```

### Description:

#### **fill(boolean[] a, boolean val) method:**

Assigns the specified boolean value to each element of the specified array of booleans.



**fill(boolean[] a, int fromIndex, int toIndex, boolean val) method:**

Assigns the specified boolean value to each element of the specified range of the specified array of booleans. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex==toIndex*, the range to be filled is empty.

**fill(byte[] a, byte val) method:**

Assigns the specified byte value to each element of the specified array of bytes.

**fill(byte[] a, int fromIndex, int toIndex, byte val) method:**

Assigns the specified byte value to each element of the specified range of the specified array of bytes. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex==toIndex*, the range to be filled is empty.

**fill(char[] a, char val) method:**

Assigns the specified char value to each element of the specified array of chars.

**fill(char[] a, int fromIndex, int toIndex, char val) method:**

Assigns the specified char value to each element of the specified range of the specified array of chars. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex==toIndex*, the range to be filled is empty.

**fill(float[] a, float val) method:**

Assigns the specified float value to each element of the specified array of floats.

**fill(float[] a, int fromIndex, int toIndex, float val) method:**

Assigns the specified float value to each element of the specified range of the specified array of floats. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex==toIndex*, the range to be filled is empty.

**fill(int[] a, int val) method:**

Assigns the specified int value to each element of the specified array of ints.

**fill(int[] a, int fromIndex, int toIndex, int val) method:**

Assigns the specified int value to each element of the specified range of the specified array of ints. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex==toIndex*, the range to be filled is empty.

**fill(Object[] a, Object val) method:**

Assigns the specified Object reference to each element of the specified array.

Note that there will be only one object on the heap.

**fill(Object[] a, int fromIndex, int toIndex, Object val) method:**

Assigns the specified Object reference to each element of the specified range in the array. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be filled is empty.

Note that there will be only one object on the heap.

**fill(short[] a, short val) method:**

Assigns the specified short value to each element of the specified array of shorts.

**fill(short[] a, int fromIndex, int toIndex, short val) method:**

Assigns the specified short value to each element of the specified range of the specified array of shorts. The range to be filled extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be filled is empty.

**Example**

```
int intArray[] = new int[10];
Arrays.fill(intArray,10);
for (int i = 0; i < 10; i++)
    Logger.log(i + " element has a value of: " + intArray[i]);
```

The example above assigns a value of 10 to every element in the *intArray*.

**Package [java.util](#)****Class [Arrays](#)****Public Method [sort](#)**

```
static void sort(byte[] a)
static void sort(byte[] a, int fromIndex, int toIndex)
```

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)

```
static void sort(char[] a)
static void sort(char[] a, int fromIndex, int toIndex)
```

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)

```
static void sort(float[] a)
static void sort(float[] a, int fromIndex, int toIndex)
```

**Throws:**

☞ [ArrayIndexOutOfBoundsException](#)

```
static void sort(int[] a)
static void sort(int[] a, int fromIndex, int toIndex)
```

**Throws:**

☞ [ArrayIndexOutOfBoundsException](#)

```
static void sort(Object[] a)
static void sort(Object[] a, Comparator c)
static void sort(Object[] a, int fromIndex, int toIndex)
static void sort(Object[] a, int fromIndex, int toIndex, Comparator c)
static void sort(short[] a)
static void sort(short[] a, int fromIndex, int toIndex)
```

**Throws:**

☞ [ArrayIndexOutOfBoundsException](#)

**Description:****sort(byte[] a) method:**

Sorts the specified array of bytes into ascending numerical order.

**sort(byte[] a, int fromIndex, int toIndex) method:**

Sorts the specified range of the specified array of bytes into ascending numerical order. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty.

**sort(char[] a) method:**

Sorts the specified array of bytes into ascending numerical order.

**sort(char[] a, int fromIndex, int toIndex) method:**

Sorts the specified range of the specified array of bytes into ascending numerical order. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty.

**sort(float[] a) method:**

Sorts the specified array of bytes into ascending numerical order.

**sort(float[] a, int fromIndex, int toIndex) method:**

Sorts the specified range of the specified array of bytes into ascending numerical order. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty.

**sort(int[] a) method:**

Sorts the specified array of bytes into ascending numerical order.

**sort(int[] a, int fromIndex, int toIndex) method:**

Sorts the specified range of the specified array of bytes into ascending numerical order. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty.

**sort(short[] a) method:**

Sorts the specified array of bytes into ascending numerical order.

**sort(short[] a, int fromIndex, int toIndex) method:**

Sorts the specified range of the specified array of bytes into ascending numerical order. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty.

**sort(Object[] a) method:**

Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. All elements in the array must implement the *Comparable* interface. Furthermore, all elements in the array must be mutually comparable that is, *e1.compareTo(e2)* must not throw a *ClassCastException* for any elements *e1* and *e2* in the array.

**sort(Object[] a, Comparator c) method:**

Sorts the specified array of objects according to the order induced by the specified comparator. All elements in the array must be mutually comparable by the specified comparator that is, *c.compare(e1, e2)* must not throw a *ClassCastException* for any elements *e1* and *e2* in the array.

**sort(float[] a, int fromIndex, int toIndex) method:**

Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty. All elements in this range must implement the *Comparable* interface. Furthermore, all elements in this range must be mutually comparable that is, *e1.compareTo(e2)* must not throw a *ClassCastException* for any elements *e1* and *e2* in the array.

**sort(Object[] a, int fromIndex, int toIndex, Comparator c) method:**

Sorts the specified range of the specified array of objects according to the order induced by the specified comparator. The range to be sorted extends from index *fromIndex*, inclusive, to index *toIndex*, exclusive. If *fromIndex*==*toIndex*, the range to be sorted is empty. All elements in the

range must be mutually comparable by the specified comparator that is, *c.compare(e1, e2)* must not throw a *ClassCastException* for any elements *e1* and *e2* in the range.

### Example

```
int intArr[] = {30,20,5,12,55};
Arrays.sort(intArr);
```

The example above orders the contained elements of the *intArr* variable.

## Package [java.util](#)

### Class **BitSet**

extends [Object](#)

implements [Cloneable](#), [Serializable](#)

This class implements a vector of bits that grows as needed. Each component of the bit set has a *boolean* value. The bits of a *BitSet* are indexed by nonnegative integers. Individual indexed bits can be examined, set, or cleared. One *BitSet* may be used to modify the contents of another *BitSet* through logical AND, logical inclusive OR, and logical exclusive OR operations.

#### Public Constructors

[BitSet](#)

#### Public Methods

[and](#)  
[andNot](#)  
[clear](#)  
[clone](#)  
[equals](#)  
[get](#)  
[hashCode](#)  
[length](#)  
[or](#)  
[set](#)  
[size](#)  
[toString](#)  
[xor](#)

#### Methods inherited from [java.lang.Object](#)

[equals](#)  
[toString](#)  
[wait](#)  
[hashCode](#)  
[notify](#)  
[notifyAll](#)

## Package [java.util](#)

### Class **BitSet**

#### Public Constructor **BitSet**

```
BitSet()
```

```
BitSet(int nr)
```

**Throws:**

☞ [NegativeArraySizeException](#)

**Description:****constructor BitSet():**

Constructs a new bit set. All bits are assigned internally with the *boolean* value *false*.

**constructor BitSet(int nr):**

Constructs a new bit set whose initial size is large enough to explicitly represent bits with indices in the range *0* to *nbits-1*. All bits are assigned internally with the *boolean* value *false*.

Package [java.util](#)

**Class [BitSet](#)****Public Method and**

```
void and(BitSet bitset)
```

**Description:**

Performs a logical AND operation of this bit set with the argument bit set. This bit set is modified so that each bit in it has the *boolean* value *true* if it both initially had the *boolean* value *true* and the corresponding bit in the bit set argument also had the *boolean* value *true*.

**Example**

```
BitSet bitset1 = new BitSet(6);
BitSet bitset2 = new BitSet(6);
bitset1.set(0);
bitset1.set(2);

bitset2.set(2);
bitset2.set(4);

Logger.log("Bitset1:" + bitset1);
Logger.log("Bitset2:" + bitset2);
bitset1.and(bitset2);

Logger.log("Bitset1 after AND operation: " + bitset1);
```

The example above returns bit set after the AND operation.

Package [java.util](#)

**Class [BitSet](#)****Public Method andNot**

```
void andNot(BitSet bitset)
```

**Description:**

Clears all of the bits in the *BitSet* whose corresponding bit is set in the specified *BitSet*.

**Example**

```
BitSet bitset1 = new BitSet(6);
BitSet bitset2 = new BitSet(6);
bitset1.set(0);
bitset1.set(2);

bitset2.set(2);
bitset2.set(4);

Logger.log("Bitset1:" + bitset1);
Logger.log("Bitset2:" + bitset2);
bitset1.andNot(bitset2);

Logger.log("Bitset1 after ANDNOT operation: " + bitset1);
```

The example above returns bit set after the ANDNOT operation.

Package [java.util](#)

Class [BitSet](#)

**Public Method clear**

```
void clear(int bit)
```

**Throws:**

☞ [IndexOutOfBoundsException](#)

**Description:**

Clears the specified bit index to the *boolean* value *false*.

**Example**

```
BitSet bitset = new BitSet(6);
bitset.set(20);
bitset.set(2);
bitset.set(24);
Logger.log("Bitset:" + bitset);
bitset.clear(1);
Logger.log("Bitset:" + bitset);
```

The example above clears the second element in the *bitset*.

## Package [java.util](#)

### Class [BitSet](#)

#### Public Method clone

```
Object clone()
```

#### Overrides:

↳ [clone](#) in class [Object](#)

#### Description:

Cloning this *BitSet* produces a new *BitSet* that is equal to it. The clone of the bit set is another bit set that has exactly the same bits set to *true* as this bit set and the same current size.

#### Example

```
BitSet bitset1 = new BitSet();
BitSet bitset2 = new BitSet();

bitset1.set(0);
bitset1.set(1);
bitset1.set(2);
bitset1.set(3);
bitset1.set(4);

Logger.log("Bitset2: " + bitset2);
bitset2 = (BitSet) bitset1.clone();
Logger.log("Bitset2: " + bitset2);
```

The example above clones the bitset1 to the bitset2.

## Package [java.util](#)

### Class [BitSet](#)

#### Public Method equals

```
boolean equals(Object obj)
```

#### Overrides:

↳ [equals](#) in class [Object](#)

#### Description:

Compares this object against the specified object. The result is *true* if and only if the argument is not *null* and is a *BitSet* object that has exactly the same set of bits set to *true* as this bit set. That is, for every nonnegative *int* index *k*,

$$((\text{BitSet}) \text{obj}).\text{get}(k) == \text{this}.\text{get}(k)$$

must be true. The current sizes of the two bit sets are not compared.



## Example

```
BitSet bitset1 = new BitSet();
BitSet bitset2 = new BitSet();

bitset1.set(0);
bitset1.set(1);
bitset1.set(2);
bitset1.set(3);
bitset1.set(4);

Logger.log("Bitset2: " + bitset2);
bitset2 = (BitSet) bitset1.clone();
Logger.log("Bitset2: " + bitset2);
if (bitset2.equals(bitset1))
    Logger.log("bitset2 equals bitset1");
```

The example above clones the `bitset1` to the `bitset2`. Afterwards the equality between `bitset1` and `bitset2` is verified.

Package [java.util](#)

Class [BitSet](#)

### Public Method `get`

```
boolean get(int bit)
```

#### Throws:

☞ [IndexOutOfBoundsException](#)

#### Description:

Returns the value of the bit with the specified index. The value is *true* if the bit with the index *bitIndex* is currently set in this *BitSet*. Otherwise, the result is *false*.

#### Example

```
if (bitset.get(1))
    Logger.log("The second element is set to a correct value!");
```

The `bitset.get` method returns true only if a correct value is assigned to the second element.

Package [java.util](#)

Class [BitSet](#)

### Public Method `hashCode`

```
int hashCode()
```

#### Overrides:

☞ [hashCode](#) in class [Object](#)

**Description:**

Returns a hash code value for this bit set. The hash code depends only on which bits have been set with this *BitSet*.

**Example**

```
BitSet bitset = new BitSet(6);
bitset.set(0);
Logger.log("hash code of the bitset: " + bitset.hashCode());
```

The example above returns the hash code of the *bitset*.

Package [java.util](#)

Class [BitSet](#)

**Public Method length**

```
int length()
```

**Description:**

Returns the length of this *BitSet*. The index of the highest set bit in the *BitSet* plus one. Returns zero if the *BitSet* contains no set bits.

**Example**

```
BitSet bitset = new BitSet(6);
bitset.set(0);
Logger.log("length of the bitset: " + bitset.length());
```

The example above returns the length of the *bitset*.

Package [java.util](#)

Class [BitSet](#)

**Public Method or**

```
void or(BitSet bitset)
```

**Description:**

Performs a logical OR operation of this bit set with the bit set argument. This bit set is modified so that a bit is set to the *boolean* value *true* if it either already had the *boolean* value *true* or the corresponding bit in the bit set argument has the *boolean* value *true*.

**Example**

```
BitSet bitset1 = new BitSet(6);
BitSet bitset2 = new BitSet(6);
bitset1.set(0);
bitset1.set(2);
```

```
bitset2.set(2);
bitset2.set(4);

Logger.log("Bitset1:" + bitset1);
Logger.log("Bitset2:" + bitset2);
bitset1.or(bitset2);

Logger.log("Bitset1 after OR operation: " + bitset1);
```

The example above returns bit set after the OR operation.

Package [java.util](#)

Class [BitSet](#)

### Public Method set

```
void set(int bit)
```

#### Throws:

☞ [IndexOutOfBoundsException](#)

#### Description:

Sets the bit specified by the index to *true*.

#### Example

```
BitSet bitset = new BitSet(6);
bitset.set(0);
```

The *bitset.set* method sets the first element of the *bitset*.

Package [java.util](#)

Class [BitSet](#)

### Public Method size

```
int size()
```

#### Description:

Returns the number of bits of space actually in use by this *BitSet* to represent bit values. The maximum element in the set is the size - 1st element.

#### Example

```
BitSet bitset = new BitSet(6);
bitset.set(0);
Logger.log("size of the bitset: " + bitset.size());
```

The example above returns the size of the *bitset*.

Package [java.util](#)  
Class [BitSet](#)

### Public Method `toString`

```
String toString()
```

#### Overrides:

↳ [toString](#) in class [Object](#)

#### Description:

Returns a string representing of this bit set. For every index for which this *BitSet* contains a bit in the set state, the decimal representation of that index is included in the result.

#### Example

```
BitSet bitset = new BitSet(6);  
bitset.set(0);  
Logger.log("bitset to string: " + bitset.toString());
```

The example above returns the string representation of this *bitset*.

Package [java.util](#)  
Class [BitSet](#)

### Public Method `xor`

```
void xor(BitSet bitset)
```

#### Description:

Performs a logical XOR of this bit set with the bit set argument. This bit set is modified so that a bit in it has the value *true* if one of the following statements holds:

- ↳ The bit initially has the value *true*, and the corresponding bit in the argument has the value *false*.
- ↳ The bit initially has the value *false*, and the corresponding bit in the argument has the value *true*.

#### Example

```
BitSet bitset1 = new BitSet(6);  
BitSet bitset2 = new BitSet(6);  
bitset1.set(0);  
bitset1.set(2);  
  
bitset2.set(2);  
bitset2.set(4);  
  
Logger.log("Bitset1:" + bitset1);  
Logger.log("Bitset2:" + bitset2);  
bitset1.xor(bitset2);
```

```
Logger.log("Bitset1 after XOR operation: " + bitset1);
```

The example above returns bit set after the XOR operation.

## Package [java.util](#)

### Interface **Collection**

The *Collection* interface is the root interface in the *collection hierarchy*. A collection represents a group of objects, known as its elements. It is not allowed to directly implement this interface. This interface is typically used to pass collection around and manipulate them where maximum generality is desired.

All general-purpose *Collection* implementation classes which typically implement *Collection* indirectly through one of its subinterfaces should provide two "standard" constructors:

- ☐ a void (no arguments) constructor, which creates an empty collection, and
- ☐ a constructor with a single argument of type *Collection*, which creates a new collection with the same elements as its argument

In effect, the latter constructor allows the user to copy any collection, producing an equivalent collection of the desired implementation type.

#### Public Methods

- ☐ [add](#)
- ☐ [addAll](#)
- ☐ [clear](#)
- ☐ [contains](#)
- ☐ [containsAll](#)
- ☐ [equals](#)
- ☐ [hashCode](#)
- ☐ [isEmpty](#)
- ☐ [iterator](#)
- ☐ [remove](#)
- ☐ [removeAll](#)
- ☐ [retainAll](#)
- ☐ [size](#)
- ☐ [toArray](#)

## Package [java.util](#)

### Interface **Collection**

#### **Public Method add**

```
abstract boolean add(Object o)
```

#### **Description:**

Ensures that this collection contains the specified element (optional operation).

Returns *true* if this collection changed as a result of the call. Returns *false* if this collection does not permit duplicates and already contains the specified element.

Collections that support this operation may place limitations on what elements may be added to this collection. In particular, some collections will refuse to add *null* elements, and others will impose

restrictions on the type of elements that may be added. Collection classes should clearly specify in their documentation any restrictions on what elements may be added.

If a collection refuses to add a particular element for any reason other than that it already contains the element, it must throw an exception rather than returning false. This preserves the invariant that a collection always contains the specified element after this call returns.

Package [java.util](#)

## Interface [Collection](#)

### Public Method addAll

```
abstract boolean addAll(Collection c)
```

#### Description:

Adds all of the elements in the specified collection to this collection (optional operation).

The behavior of this operation is undefined if the specified collection is modified while the operation is in progress. This implies that the behavior of this call is undefined if the specified collection is this collection, and this collection is nonempty.

Package [java.util](#)

## Interface [Collection](#)

### Public Method clear

```
abstract void clear()
```

#### Description:

Removes all of the elements from this collection (optional operation). This collection will be empty after this method returns unless it throws an exception.

Package [java.util](#)

## Interface [Collection](#)

### Public Method contains

```
abstract boolean contains(Object o)
```

#### Description:

Returns *true* if this collection contains the specified element. More formally, returns *true* if and only if this collection contains at least one element *e* such that  $(o==null ? e==null : o.equals(e))$ .

Package [java.util](#)

## Interface [Collection](#)

### Public Method containsAll

```
abstract boolean containsAll(Collection c)
```

**Description:**

Returns *true* if this collection contains all of the elements in the specified collection.

Package [java.util](#)

**Interface [Collection](#)****Public Method [equals](#)**

```
abstract boolean equals(Object o)
```

**Overrides:**

☞ [equals](#) in class [Object](#)

**Description:**

Compares the specified object with this collection for equality.

While the *Collection* interface adds no stipulations to the general contract for the *Object.equals*, programmers who implement the *Collection* interface "directly" (in other words, create a class that is a *Collection* but is not a *Set* or a *List*) must exercise care if they choose to override the *Object.equals*. It is not necessary to do so, and the simplest course of action is to rely on *Object*'s implementation, but the implementer may wish to implement a "value comparison" in place of the default "reference comparison." (The *List* and *Set* interfaces mandate such value comparisons.)

The general contract for the *Object.equals* method states that equals must be symmetric (in other words, *a.equals(b)* if and only if *b.equals(a)*). The contracts for *List.equals* and *Set.equals* state that lists are only equal to other lists, and sets to other sets. Thus, a custom equals method for a collection class that implements neither the *List* nor *Set* interface must return false when this collection is compared to any list or set. (By the same logic, it is not possible to write a class that correctly implements both the *Set* and *List* interfaces.)

Package [java.util](#)

**Interface [Collection](#)****Public Method [hashCode](#)**

```
abstract int hashCode()
```

**Overrides:**

☞ [hashCode](#) in class [Object](#)

**Description:**

Returns the hash code value for this collection. While the *Collection* interface adds no stipulations to the general contract for the *Object.hashCode* method, programmers should take note that any class that overrides the *Object.equals* method must also override the *Object.hashCode* method in order to satisfy the general contract for the *Object.hashCode* method. In particular, *c1.equals(c2)* implies that *c1.hashCode()==c2.hashCode()*.

Package [java.util](#)  
Interface [Collection](#)

**Public Method isEmpty**

```
abstract boolean isEmpty()
```

**Description:**

Returns *true* if this collection contains no elements.

Package [java.util](#)  
Interface [Collection](#)

**Public Method iterator**

```
abstract Iterator iterator()
```

**Description:**

Returns an iterator over the elements in this collection. There are no guarantees concerning the order in which the elements are returned unless this collection is an instance of some class that provides a guarantee.

Package [java.util](#)  
Interface [Collection](#)

**Public Method remove**

```
abstract boolean remove(Object o)
```

**Description:**

Removes a single instance of the specified element from this collection, if it is present (optional operation).

More formally, removes an element *e* such that  $(o == null ? e == null : o.equals(e))$ , if this collection contains one or more such elements. Returns *true* if this collection contained the specified element (or equivalently, if this collection changed as a result of the call).

Package [java.util](#)  
Interface [Collection](#)

**Public Method removeAll**

```
abstract boolean removeAll(Collection c)
```

**Description:**

Removes all this collection's elements that are also contained in the specified collection (optional operation).

After this call returns, this collection will contain no elements in common with the specified collection.



Package [java.util](#)

## Interface [Collection](#)

### Public Method retainAll

```
abstract boolean retainAll(Collection c)
```

#### Description:

Retains only the elements in this collection that are contained in the specified collection (optional operation).

In other words, removes from this collection all of its elements that are not contained in the specified collection.

Package [java.util](#)

## Interface [Collection](#)

### Public Method size

```
abstract int size()
```

#### Description:

Returns the number of elements in this collection. If this collection contains more than *Integer.MAX\_VALUE* elements, returns *Integer.MAX\_VALUE*.

Package [java.util](#)

## Interface [Collection](#)

### Public Method toArray

```
abstract Object[] toArray()
```

#### Description:

Returns an array containing all of the elements in this collection. If the collection makes any guarantees as to what order its elements are returned by its iterator, this method must return the elements in the same order.

The returned array will be "safe" in that no references to it are maintained by this collection. The caller is thus free to modify the returned array.

This method acts as bridge between array-based and collection-based API-s.

Package [java.util](#)

## Class Collections

extends [Object](#)

Contains static methods that operate on or return collections.

The methods throw a `NullPointerException` if collections or provided objects are null.

## Public Methods

- ☐ [binarySearch](#)
- ☐ [copy](#)
- ☐ [enumeration](#)
- ☐ [fill](#)
- ☐ [max](#)
- ☐ [min](#)
- ☐ [nCopies](#)
- ☐ [reverse](#)
- ☐ [reverseOrder](#)
- ☐ [shuffle](#)
- ☐ [singleton](#)
- ☐ [singletonList](#)
- ☐ [singletonMap](#)
- ☐ [sort](#)
- ☐ [synchronizedCollection](#)
- ☐ [synchronizedList](#)
- ☐ [synchronizedMap](#)
- ☐ [synchronizedSet](#)
- ☐ [synchronizedSortedMap](#)
- ☐ [synchronizedSortedSet](#)
- ☐ [unmodifiableCollection](#)
- ☐ [unmodifiableList](#)
- ☐ [unmodifiableMap](#)
- ☐ [unmodifiableSet](#)
- ☐ [unmodifiableSortedMap](#)
- ☐ [unmodifiableSortedSet](#)

## Methods inherited from [java.lang.Object](#)

- ☐ [equals](#)
- ☐ [toString](#)
- ☐ [wait](#)
- ☐ [hashCode](#)
- ☐ [notify](#)
- ☐ [notifyAll](#)

## Public Fields

- ☐ static final List EMPTY\_LIST
- ☐ static final Map EMPTY\_MAP
- ☐ static final Set EMPTY\_SET

The empty list, map, set. All are immutable.

## Package [java.util](#)

## Class [Collections](#)

### Public Method [binarySearch](#)

```
static int binarySearch(List l, Object key)
static int binarySearch(List l, Object key, Comparator c)
```

### Throws:

- ☐ [NullPointerException](#)

**Description:****binarySearch(List l, Object key) method:**

Searches the specified list for the specified key object using the binary search algorithm. The list must be sorted into ascending order according to the natural ordering of its elements (as by the `sort(List)` method, above) prior to making this call. If it is not sorted, the results are undefined. If the list contains multiple elements equal to the specified object, there is no guarantee which one will be found.

This method runs in  $\log(n)$  time for a "random access" list (which provides near-constant-time positional access). It may run in  $n \log(n)$  time if it is called on a "sequential access" list (which provides linear-time positional access).

If the specified list implements the `AbstractSequentialList` interface, this method will do a sequential search instead of a binary search; this offers linear performance instead of  $n \log(n)$  performance if this method is called on a `LinkedList` object.

Returns index of the search key, if it is contained in the list; otherwise,  $-(\text{insertion point}) - 1$ . The insertion point is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or `list.size()`, if all elements in the list are less than the specified key. Note that this guarantees that the return value will be  $\geq 0$  if and only if the key is found.

**binarySearch(List l, Object key, Comparator c) method:**

Like above but with additional `Comparator` which determines the list ordering. A null value indicates that the elements' natural ordering should be used.

Package [java.util](#)

Class [Collections](#)

**Public Method copy**

```
static void copy(List dest, List source)
```

**Description:**

Copies all of the elements from one list into another. After the operation, the index of each copied element in the destination list will be identical to its index in the source list. The destination list must be at least as long as the source list. If it is longer, the remaining elements in the destination list are unaffected.

This method runs in linear time.

Package [java.util](#)

Class [Collections](#)

**Public Method enumeration**

```
static Enumeration enumeration(Collection c)
```

**Description:**

Returns an enumeration over the specified collection.s

Package [java.util](#)  
Class [Collections](#)

**Public Method fill**

```
static void fill(List l, Object val)
```

**Description:**

Replaces all of the elements of the specified list with the specified element.

This method runs in linear time.

Package [java.util](#)  
Class [Collections](#)

**Public Method max**

```
static Object max(Collection c)
```

```
static Object max(Collection c, Comparator order)
```

**Description:**

**max(Collection c) method:**

Like below but using the natural ordering of the elements.

**max(Collection c, Comparator order) method:**

Returns the maximum element of the given collection, according to the order induced by the specified comparator. All elements in the collection must be mutually comparable by the specified comparator (that is, `comp.compare(e1, e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the collection).

This method iterates over the entire collection, hence it requires time proportional to the size of the collection.

Package [java.util](#)  
Class [Collections](#)

**Public Method min**

```
static Object min(Collection c)
```

```
static Object min(Collection c, Comparator order)
```

**Description:**

**max(Collection c) method:**

Like below but using the natural ordering of the elements.

**max(Collection c, Comparator order) method:**

Returns the minimum element of the given collection, according to the order induced by the specified comparator. All elements in the collection must be mutually comparable by the specified comparator (that is, `comp.compare(e1, e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the collection).

This method iterates over the entire collection, hence it requires time proportional to the size of the collection.

Package [java.util](#)

**Class [Collections](#)****Public Method nCopies**

```
static List nCopies(int n, Object o)
```

**Throws:**

☞ [IllegalArgumentException](#)

**Description:**

Returns an immutable list consisting of `n` copies of the specified object. The newly allocated data object is tiny (it contains a single reference to the data object). This method is useful in combination with the `List.addAll` method to grow lists. The returned list is serializable.

Package [java.util](#)

**Class [Collections](#)****Public Method reverse**

```
static void reverse(List l)
```

**Description:**

Reverses the order of the elements in the specified list. This method runs in linear time.

Package [java.util](#)

**Class [Collections](#)****Public Method reverseOrder**

```
static Comparator reverseOrder()
```

Returns a comparator that imposes the reverse of the natural ordering on a collection of objects that implement the `Comparable` interface. (The natural ordering is the ordering imposed by the objects' own `compareTo` method.) This enables a simple idiom for sorting (or maintaining) collections (or arrays) of objects that implement the `Comparable` interface in reverse-natural-order. For example, suppose `a` is an array of strings. Then

```
Arrays.sort(a, Collections.reverseOrder());
```

sorts the array in reverse-lexicographic (alphabetical) order. The returned comparator is serializable.

Package [java.util](#)  
Class [Collections](#)

**Public Method [shuffle](#)**

```
static void shuffle(List l)
static void shuffle(List l, Random r)
```

**Description:**

Permute list, traversing the list backwards, from the last element up to the second, repeatedly swapping a randomly selected element into the "current position". Elements are randomly selected from the portion of the list that runs from the first element to the current position, inclusive. This method runs in linear time for a "random access" list (which provides near-constant-time positional access). It may require quadratic time for a "sequential access" list.

**shuffle(List l) method:**

Randomly permutes the specified list using a default source of randomness. All permutations occur with approximately equal likelihood.

**shuffle(List l, Random r) method:**

Randomly permute the specified list using the specified source of randomness.

Package [java.util](#)  
Class [Collections](#)

**Public Method [singleton](#)**

```
static Set singleton(Object o)
```

**Description:**

Returns an immutable set containing only the specified object. The returned set is serializable.

Package [java.util](#)  
Class [Collections](#)

**Public Method [singletonList](#)**

```
static List singletonList(Object o)
```

**Description:**

Returns an immutable list containing only the specified object. The returned list is serializable.

Package [java.util](#)  
Class [Collections](#)

**Public Method [singletonMap](#)**

```
static Map singletonMap(Object key, Object value)
```

**Description:**

Returns an immutable map, mapping only the specified key to the specified value. The returned map is serializable.

Package [java.util](#)

**Class [Collections](#)****Public Method [sort](#)**

```
static void sort(List l)

static void sort(List l, Comparator c)
```

**Description:**

Sorts the specified list into ascending order. All elements in the list must be mutually comparable using the specified comparator (that is, `c.compare(e1, e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list). All elements in the list must implement the `Comparable` interface.

This sort is guaranteed to be stable: equal elements will not be reordered as a result of the sort.

**`sort(List l)` method:**

Sorts the specified list into ascending order, according to the natural ordering of its elements.

**`sort(List l, Comparator c)` method:**

Sorts the specified list according to the order induced by the specified comparator.

Package [java.util](#)

**Class [Collections](#)****Public Method [synchronizedCollection](#)**

```
static Collection synchronizedCollection(Collection c)
```

**Description:**

Returns a synchronized (thread-safe) collection backed by the specified collection. Changes in the returned sorted set are reflected in this set, and vice-versa. In order to guarantee serial access, it is critical that all access to the backing collection is accomplished through the returned collection.

The user has to manually synchronize on the returned collection when iterating over it:

```
synchronized(MyCollection) { // iterate in a synchronized block
    Iterator i = MyCollection.iterator();
    while (i.hasNext())
        myFun(i.next());
}
```

The returned collection does not pass the `hashCode` and `equals` operations through to the backing collection, but relies on `Object`'s `equals` and `hashCode` methods. This is necessary to preserve the

contracts of these operations in the case that the backing collection is a set or a list. The returned collection will be serializable if the specified collection is serializable.

Package [java.util](#)

## Class [Collections](#)

### Public Method `synchronizedList`

```
static List synchronizedList(List l)
```

#### **Description:**

Returns a synchronized (thread-safe) list backed by the specified list. Changes in the returned list are reflected in this list, and vice-versa. In order to guarantee serial access, it is critical that all access to the backing list is accomplished through the returned list.

The user has to manually synchronize on the returned list when iterating over it:

```
synchronized(MyList) { // iterate in a synchronized block
    Iterator i = MyList.iterator();
    while (i.hasNext())
        myFun(i.next());
}
```

The returned collection will be serializable if the specified list is serializable.

Package [java.util](#)

## Class [Collections](#)

### Public Method `synchronizedMap`

```
static Map synchronizedMap(Map m)
```

#### **Description:**

Returns a synchronized (thread-safe) map backed by the specified map. Changes in the returned sorted map are reflected in this map, and vice-versa. In order to guarantee serial access, it is critical that all access to the backing map is accomplished through the returned map.

The user has to manually synchronize on the returned map when iterating over it:

```
Set s = MyMap.keySet(); // doesn't have to be in the synchronized block
synchronized(MyMap) { // iterate in a synchronized block
    Iterator i = s.iterator();
    while (i.hasNext())
        myFun(i.next());
}
```

The returned collection will be serializable if the specified map is serializable.



Package [java.util](#)  
Class [Collections](#)

**Public Method `synchronizedSet`**

```
static Set synchronizedSet(Set s)
```

**Description:**

Returns a synchronized (thread-safe) set backed by the specified set. Changes in the returned set are reflected in this set, and vice-versa. In order to guarantee serial access, it is critical that all access to the backing set is accomplished through the returned set.

The user has to manually synchronize on the returned set when iterating over it:

```
synchronized(MySet) { // iterate in a synchronized block
    Iterator i = MySet.iterator();
    while (i.hasNext())
        myFun(i.next());
}
```

The returned collection will be serializable if the specified set is serializable.

Package [java.util](#)  
Class [Collections](#)

**Public Method `synchronizedSortedMap`**

```
static SortedMap synchronizedSortedMap(SortedMap m)
```

**Description:**

Returns a synchronized (thread-safe) sorted map backed by the specified sorted map. Changes in the returned sorted map are reflected in this map, and vice-versa. In order to guarantee serial access, it is critical that all access to the backing sorted map is accomplished through the returned sorted map.

The user has to manually synchronize on the returned sorted map when iterating over it:

```
Set s = MySortedMap.keySet(); // doesn't have to be in the synchronized
block
synchronized(MySortedMap) { // iterate in a synchronized block
    Iterator i = s.iterator();
    while (i.hasNext())
        myFun(i.next());
}
```

The returned collection will be serializable if the specified sorted map is serializable.

Package [java.util](#)  
Class [Collections](#)

**Public Method `synchronizedSortedSet`**

```
static SortedSet synchronizedSortedSet(SortedSet s)
```

**Description:**

Returns a synchronized (thread-safe) sorted set backed by the specified sorted set. In order to guarantee serial access, it is critical that all access to the backing sorted set is accomplished through the returned sorted set.

The user has to manually synchronize on the returned sorted set when iterating over it:

```
synchronized(MySortedSet) { // iterate in a synchronized block
    Iterator i = MySortedSet.iterator();
    while (i.hasNext())
        myFun(i.next());
}
```

The returned collection will be serializable if the specified sorted set is serializable.

**Package [java.util](#)  
Class [Collections](#)****Public Method [unmodifiableCollection](#)**

```
static Collection unmodifiableCollection(Collection c)
```

**Description:**

Returns an unmodifiable view of the specified collection. This method allows modules to provide users with "read-only" access to internal collections. Query operations on the returned collection "read through" to the specified collection, and attempts to modify the returned collection, whether direct or via its iterator, result in an UnsupportedOperationException.

The returned collection does not pass the hashCode and equals operations through to the backing collection, but relies on Object's equals and hashCode methods. This is necessary to preserve the contracts of these operations in the case that the backing collection is a set or a list.

The returned collection will be serializable if the specified collection is serializable.

**Package [java.util](#)  
Class [Collections](#)****Public Method [unmodifiableList](#)**

```
static List unmodifiableList(List l)
```

**Description:**

Returns an unmodifiable view of the specified list. This method allows modules to provide users with "read-only" access to internal lists. Query operations on the returned list "read through" to the specified list, and attempts to modify the returned list, whether direct or via its iterator, result in an UnsupportedOperationException.

The returned list will be serializable if the specified list is serializable.

Package [java.util](#)

## Class [Collections](#)

### Public Method `unmodifiableMap`

```
static Map unmodifiableMap(Map m)
```

#### Description:

Returns an unmodifiable view of the specified map. This method allows modules to provide users with "read-only" access to internal maps. Query operations on the returned map "read through" to the specified map, and attempts to modify the returned map, whether direct or via its collection views, result in an `UnsupportedOperationException`.

The returned map will be serializable if the specified map is serializable.

Package [java.util](#)

## Class [Collections](#)

### Public Method `unmodifiableSet`

```
static Set unmodifiableSet(Set s)
```

#### Description:

Returns an unmodifiable view of the specified set. This method allows modules to provide users with "read-only" access to internal sets. Query operations on the returned set "read through" to the specified set, and attempts to modify the returned set, whether direct or via its iterator, result in an `UnsupportedOperationException`.

The returned set will be serializable if the specified set is serializable.

Package [java.util](#)

## Class [Collections](#)

### Public Method `unmodifiableSortedMap`

```
static SortedMap unmodifiableSortedMap(SortedMap m)
```

#### Description:

Returns an unmodifiable view of the specified sorted map. This method allows modules to provide users with "read-only" access to internal sorted maps. Query operations on the returned sorted map "read through" to the specified sorted map. Attempts to modify the returned sorted map, whether direct, via its collection views, or via its `subMap`, `headMap`, or `tailMap` views, result in an `UnsupportedOperationException`.

The returned sorted map will be serializable if the specified sorted map is serializable.

Package [java.util](#)

## Class [Collections](#)

### Public Method `unmodifiableSortedSet`

```
static SortedSet unmodifiableSortedSet(SortedSet s)
```

**Description:**

Returns an unmodifiable view of the specified sorted set. This method allows modules to provide users with "read-only" access to internal sorted sets. Query operations on the returned sorted set "read through" to the specified sorted set. Attempts to modify the returned sorted set, whether direct, via its iterator, or via its subSet, headSet, or tailSet views, result in an UnsupportedOperationException.

The returned sorted set will be serializable if the specified sorted set is serializable.

Package [java.util](#)

**Interface Comparator****Description:**

A comparison function, which imposes a total ordering on some collection of objects. Comparators can be passed to a sort method (such as `Collections.sort`) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as `TreeSet` or `TreeMap`).

The ordering imposed by a Comparator  $c$  on a set of elements  $S$  is said to be consistent with equals if and only if  $(compare((Object)e1, (Object)e2) == 0)$  has the same boolean value as  $e1.equals((Object)e2)$  for every  $e1$  and  $e2$  in  $S$ .

Caution should be exercised when using a comparator capable of imposing an ordering inconsistent with equals to order a sorted set (or sorted map). Suppose a sorted set (or sorted map) with an explicit Comparator  $c$  is used with elements (or keys) drawn from a set  $S$ . If the ordering imposed by  $c$  on  $S$  is inconsistent with equals, the sorted set (or sorted map) will behave "strangely." In particular the sorted set (or sorted map) will violate the general contract for set (or map), which is defined in terms of `equals`.

**Public Methods**

 [compare](#)

Package [java.util](#)

**Interface [Comparator](#)****Public Method [compare](#)**

```
abstract int compare(Object o1, Object o2)
```

**Description:**

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

The implementor must ensure that  $sgn(compare(x, y)) == -sgn(compare(y, x))$  for all  $x$  and  $y$ . This implies that `compare(x, y)` must throw an exception if and only if `compare(y, x)` throws an exception.

The implementor must also ensure that the relation is transitive:  $((compare(x, y) > 0) \ \&\& \ (compare(y, z) > 0))$  implies  $compare(x, z) > 0$ .

Finally, the implementer must ensure that  $compare(x, y) == 0$  implies that  $sgn(compare(x, z)) == sgn(compare(y, z))$  for all  $z$ .

It is generally the case, but not strictly required that  $(compare(x, y) == 0) == (x.equals(y))$ . Generally speaking, any comparator that violates this condition should clearly indicate this fact. The recommended language is "Note: this comparator imposes orderings that are inconsistent with equals."

## Package [java.util](#)

### Class **ConcurrentModificationException**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

This exception may be thrown by methods that have detected concurrent modification of a backing object when such modification is not permissible.

#### Public Constructors

☞ [ConcurrentModificationException](#)

#### Methods inherited from [java.lang.Throwable](#)

- ☞ [getMessage](#)
- ☞ [getLocalizedMessage](#)
- ☞ [toString](#)
- ☞ [fillInStackTrace](#)
- ☞ [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- ☞ [equals](#)
- ☞ [toString](#)
- ☞ [wait](#)
- ☞ [hashCode](#)
- ☞ [notify](#)
- ☞ [notifyAll](#)

## Package [java.util](#)

### Class **[ConcurrentModificationException](#)**

#### **Public Constructor [ConcurrentModificationException](#)**

```
ConcurrentModificationException()
```

```
ConcurrentModificationException(String detail)
```

#### **Description:**

##### **constructor [ConcurrentModificationException\(\)](#):**

Constructs a *ConcurrentModificationException* with no detail message.

##### **constructor [ConcurrentModificationException\(String detail\)](#):**

Constructs a *ConcurrentModificationException* with the specified detail message.

## Package [java.util](#)

### Class **Date**

extends [Object](#)

Represents a specific instant in time, with seconds precision.

#### Public Constructors

☐ [Date](#)

#### Public Methods

☐ [getDay](#)  
☐ [getHour](#)  
☐ [getMinute](#)  
☐ [getMonth](#)  
☐ [getSecond](#)  
☐ [getYear](#)  
☐ [toString](#)

#### Methods inherited from [java.lang.Object](#)

☐ [equals](#)  
☐ [toString](#)  
☐ [wait](#)  
☐ [hashCode](#)  
☐ [notify](#)  
☐ [notifyAll](#)

#### See also:

[hw.Date](#)

[hw.Time](#)

## Package [java.util](#)

### Class [Date](#)

#### Public Constructor Date

```
Date(int year, int month, int day, int hour, int minute, int second)
```

#### Parameters

☐ **year** – the year minus 1900.  
☐ **month** – the month between 0-11.  
☐ **day** – the day of the month between 1-31.  
☐ **hour** – the hours between 0-23.  
☐ **minute** – the minutes between 0-59.  
☐ **second** – the seconds between 0-59.

#### Description:

Allocates a Date object and initializes it to the specified time.

Package [java.util](#)

Class [Date](#)

**Public Method getDay**

```
int getDay ()
```

**Description:**

Returns the day of the week represented by this date (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday).

Package [java.util](#)

Class [Date](#)

**Public Method getHour**

```
int getHour ()
```

**Description:**

Returns a number representing the hour of the day represented by this Date object (0 through 23).

Package [java.util](#)

Class [Date](#)

**Public Method getMinute**

```
int getMinute ()
```

**Description:**

Returns the number of minutes past the hour represented by this date (0-59).

Package [java.util](#)

Class [Date](#)

**Public Method getMonth**

```
int getMonth ()
```

**Description:**

Returns a number representing the month of the year represented by this date object (0-11):

Package [java.util](#)

Class [Date](#)

**Public Method getSecond**

```
int getSecond ()
```

**Description:**

Returns the number of seconds past the minute represented by this date (0-59).

**Package [java.util](#)****Class [Date](#)****Public Method [getYear](#)**

```
int getYear()
```

**Description:**

Returns the number of years since 1900, i.e. the returned value + 1900 gives the actual year.

**Package [java.util](#)****Class [Date](#)****Public Method [toString](#)**

```
String toString()
```

**Overrides:**

↳ [toString](#) in class [Object](#)

**Description:**

Returns a string representation of this Date object, e.g. "2017/7/6/23/2/3" (year/month/day/hour/minute/second).

Note that the year is shown correctly (i.e. + 1900).

**Package [java.util](#)****Abstract Class [Dictionary](#)**

extends [Object](#)

The *Dictionary* class is the abstract parent of any class, which maps keys to values. In any one *Dictionary* object, every key is associated with at most one value. Given a *Dictionary* and a key, the associated element can be looked up. Any non-null object can be used as a key and as a value.

**Public Constructors**

↳ [Dictionary](#)

**Public Methods**

↳ [elements](#)

↳ [get](#)

↳ [isEmpty](#)

↳ [keys](#)

↳ [put](#)

↳ [remove](#)

↳ [size](#)

**Methods inherited from [java.lang.Object](#)**

↳ [equals](#)

↳ [toString](#)



- ☐ [wait](#)
- ☐ [hashCode](#)
- ☐ [notify](#)
- ☐ [notifyAll](#)

Package [java.util](#)  
Class [Dictionary](#)

**Public Constructor Dictionary**

```
Dictionary ()
```

**Description:**

Is implicit called by the subclass.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method elements**

```
abstract Enumeration elements ()
```

**Description:**

Returns an enumeration of the values in this dictionary. The general contract for the *elements* method is that an *Enumeration* is returned that will generate all the elements contained in this dictionary.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method get**

```
abstract Object get (Object key)
```

**Description:**

Returns the value to which the key is mapped in this dictionary. The general contract for the *isEmpty* method is that if this dictionary contains an entry for the specified key, the associated value is returned. Otherwise, *null* is returned.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method isEmpty**

```
abstract boolean isEmpty ()
```

**Description:**

Tests if this dictionary maps no keys to value. The general contract for the *isEmpty* method is that the result is true if this dictionary contains no entries.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method keys**

```
abstract Enumeration keys ()
```

**Description:**

Returns an enumeration of the keys in this dictionary. The general contract for the *keys* method is that an *Enumeration* object is returned that will generate all the keys for which this dictionary contains entries.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method put**

```
abstract Object put (Object key, Object value)
```

**Description:**

Maps the specified *key* to the specified *value* in this dictionary. Neither the key nor the value can be *null*.

If this dictionary already contains an entry for the specified *key*, the value already in this dictionary for that *key* is returned, after modifying the entry to contain the new element.

If this dictionary does not already have an entry for the specified *key*, an entry is created for the specified *key* and *value*, and *null* is returned.

The *value* can be retrieved by calling the *get* method with a *key* that is equal to the original *key*.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method remove**

```
abstract Object remove (Object key)
```

**Description:**

Removes the *key* and the corresponding *value* from this dictionary. This method does nothing if the *key* is not in this dictionary.

Package [java.util](#)  
Class [Dictionary](#)

**Public Method size**

```
abstract int size ()
```

**Description:**

Returns the number of entries in this dictionary.

Package [java.util](#)

**Class `EmptyStackException`**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown by methods in the `Stack` class to indicate the stack is empty.

**Public Constructors**

⇒ [EmptyStackException](#)

**Methods inherited from [java.lang.Throwable](#)**

- ⇒ [getMessage](#)
- ⇒ [getLocalizedMessage](#)
- ⇒ [toString](#)
- ⇒ [fillInStackTrace](#)
- ⇒ [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- ⇒ [equals](#)
- ⇒ [toString](#)
- ⇒ [wait](#)
- ⇒ [hashCode](#)
- ⇒ [notify](#)
- ⇒ [notifyAll](#)

Package [java.util](#)

**Class `EmptyStackException`****Public Constructor `EmptyStackException`**

```
EmptyStackException()
```

**Description:**

Constructs a new `EmptyStackException` with `null` as its error message string.

Package [java.util](#)

**Interface Enumeration****Public Methods**

- ⇒ [hasMoreElements](#)
- ⇒ [nextElement](#)

An object that implements the Enumeration interface generates a series of elements, one at a time. Successive calls to the `nextElement` method return successive elements of the series.

For example, to print all elements of a vector `v`:

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
    System.out.println(e.nextElement());
}
```

Methods are provided to enumerate through the elements of a vector, the keys of a hashtable, and the values in a hashtable. Enumerations are also used to specify the input streams to a *SequenceInputStream*.

## Note

- ☐ The functionality of this interface is duplicated by the Iterator interface.
- ☐ In addition, Iterator adds an optional remove operation, and has shorter method names.
- ☐ New implementations should consider using Iterator in preference to Enumeration.

Package [java.util](#)

## Interface [Enumeration](#)

### Public Method hasMoreElements

```
abstract boolean hasMoreElements ()
```

#### Description:

Tests if this enumeration contains more elements.

Package [java.util](#)

## Interface [Enumeration](#)

### Public Method nextElement

```
abstract Object nextElement ()
```

#### Description:

Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Package [java.util](#)

## Interface [EventListener](#)

A tagging interface that all event listener interfaces must extend.

Package [java.util](#)

## Class [EventObject](#)

extends [Object](#)

implements [Serializable](#)

The *EventObject* class is the root class which all state objects shall be derived.

All events are constructed with a reference to the object. This object is logically deemed to be the object upon which the event in question initially occurred upon.

## Public Constructors

☞ [EventObject](#)

## Public Methods

☞ [getSource](#)

☞ [toString](#)

## Methods inherited from [java.lang.Object](#)

☞ [equals](#)

☞ [toString](#)

☞ [wait](#)

☞ [hashCode](#)

☞ [notify](#)

☞ [notifyAll](#)

Package [java.util](#)

## Class [EventObject](#)

### Public Constructor EventObject

```
EventObject (Object source)
```

#### Description:

Constructs a prototypical event.

Package [java.util](#)

## Class [EventObject](#)

### Public Method getSource

```
Object getSource ()
```

#### Description:

The object on which the event initially occurred.

Package [java.util](#)

## Class [EventObject](#)

### Public Method toString

```
String toString ()
```

#### Overrides:

☞ [toString](#) in class [Object](#)

#### Description:

Returns the string representation of this *EventObject*.

## Package [java.util](#)

### Class `HashMap`

extends [AbstractMap](#) → [Object](#)

implements [Map](#), [Cloneable](#), [Serializable](#)

Hash table based implementation of the *Map* interface. This implementation provides all of the optional map operations, and permits *null* values and the *null* key. The *HashMap* class is roughly equivalent to *Hashtable*, except that it is unsynchronized and permits nulls. This class makes no guarantees as to the order of the map. It does not guarantee that the order will remain constant over time.

#### Public Constructors

☞ [HashMap](#)

#### Public Methods

☞ [clear](#)  
☞ [clone](#)  
☞ [containsKey](#)  
☞ [containsValue](#)  
☞ [entrySet](#)  
☞ [get](#)  
☞ [isEmpty](#)  
☞ [put](#)  
☞ [putAll](#)  
☞ [remove](#)  
☞ [size](#)

#### Methods inherited from [java.util.AbstractMap](#)

☞ [size](#)  
☞ [isEmpty](#)  
☞ [containsValue](#)  
☞ [containsKey](#)  
☞ [get](#)  
☞ [put](#)  
☞ [remove](#)  
☞ [putAll](#)  
☞ [clear](#)  
☞ [keySet](#)  
☞ [values](#)  
☞ [entrySet](#)  
☞ [equals](#)  
☞ [hashCode](#)  
☞ [toString](#)  
☞ [clone](#)

#### Methods inherited from [java.lang.Object](#)

☞ [equals](#)  
☞ [toString](#)  
☞ [wait](#)  
☞ [hashCode](#)  
☞ [notify](#)  
☞ [notifyAll](#)

Package [java.util](#)  
Class [HashMap](#)

### Public Constructor HashMap

```
HashMap ()
```

```
HashMap (int initialCapacity)
```

```
HashMap (int initialCapacity, float loadFactor)
```

#### Throws:

☞ [IllegalArgumenteception](#)

```
HashMap (Map t)
```

#### Description:

##### constructor HashMap():

Constructs a new, empty map with a default capacity and load factor, which is 0.75.

##### constructor HashMap(int initialCapacity):

Constructs a new and empty map with the specified initial capacity and default load factor, which is 0.75.

##### constructor HashMap(int initialCapacity, float loadFactor):

Constructs a new and empty map with the specified initial capacity and the specified load factor.

##### constructor HashMap(Map t):

Constructs a new map with the same mappings as the given map. The map is created with a capacity of twice the number of mappings in the given map of 11, and a default load factor, which is 0.75.

Package [java.util](#)  
Class [HashMap](#)

### Public Method clear

```
void clear ()
```

#### Overrides:

☞ [clear](#) in class [AbstractMap](#)

#### Description:

Removes all mappings from this map.

## Example

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");
Logger.log("Map element: " + map.get(i));
map.clear();
Logger.log("Map element: " + map.get(i));
```

The example above clears the new created map and writes the result to the logger.

## Package [java.util](#) Class [HashMap](#)

### Public Method clone

Object **clone**()

#### Overrides:

☞ [clone](#) in class [AbstractMap](#)

#### Description:

Returns a shallow copy of this *HashMap* instance. The *keys* and *values* themselves are not cloned.

## Example

```
HashMap map1 = new HashMap();
HashMap map2 = new HashMap();
Integer i = new Integer(0);
map1.put(i, "value");
map2 =(HashMap)map1.clone();
Logger.log("Map1 element: " + map1.get(i));
Logger.log("Map2 element: " + map2.get(i));
map1.clear();
Logger.log("Map1 element: " + map1.get(i));
Logger.log("Map2 element: " + map2.get(i));
```

The example above clears the new created map and writes the result to the logger.

## Package [java.util](#) Class [HashMap](#)

### Public Method containsKey

boolean **containsKey**(Object key)

#### Overrides:

☞ [containsKey](#) in class [AbstractMap](#)



**Description:**

Returns *true* if this map contains a mapping for the specified key.

**Example**

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");

if (map.containsKey(i))
    Logger.log("This map contains the specified key!");
```

The example above verifies that the map contains the specified key and writes the result to the logger.

Package [java.util](#)

Class [HashMap](#)

**Public Method containsValue**

```
boolean containsValue(Object val)
```

**Overrides:**

☞ [containsValue](#) in class [AbstractMap](#)

**Description:**

Returns *true* if this map maps one or more keys to the specified value.

**Example**

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");

if (map.containsValue("value"))
    Logger.log("This map contains the specified value!");
```

The example above verifies that the map contains the specified value and writes the result to the logger.

Package [java.util](#)

Class [HashMap](#)

**Public Method entrySet**

```
Set entrySet()
```

**Overrides:**

☞ [entrySet](#) in class [AbstractMap](#)

### Description:

Returns a collection view of the mappings contained in this map. Each element in the returned collection is a *Map.Entry*. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. The collection supports element removal, which removes the corresponding mapping from the map, via the *Iterator.remove*, *Collection.remove*, *removeAll*, *retainAll*, and *clear* operations. It does not support the *add* or *addAll* operations.

### Example

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");

Set set = map.entrySet();
Logger.log("entrySet: " + set);
```

The example above creates a new map and assigns it to a set.

Package [java.util](#)

Class [HashMap](#)

### Public Method [get](#)

Object **get**(Object key)

### Overrides:

☐ [get](#) in class [AbstractMap](#)

### Description:

Returns the value to which this map maps the specified key. Returns *null* if the map contains no mapping for this key. A return value of *null* does not necessarily indicate that the map contains no mapping for the key. It's also possible that the map explicitly maps the key to *null*. The *containsKey* operation may be used to distinguish these two cases.

### Example

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");
Logger.log("Map element: " + map.get(i));
```

The example above clears the new created map and writes the result to the logger.

Package [java.util](#)

Class [HashMap](#)

### Public Method [isEmpty](#)

boolean **isEmpty**()

**Overrides:**

⇒ [isEmpty](#) in class [AbstractMap](#)

**Description:**

Returns the number of key value mappings in this map.

**Example**

```
HashMap map = new HashMap();

if (map.isEmpty())
    Logger.log("This map is empty!");
```

The example above creates a new map, verifies that the map is empty and writes the result to the logger.

Package [java.util](#)

Class [HashMap](#)

**Public Method put**

```
Object put(Object key, Object val)
```

**Overrides:**

⇒ [put](#) in class [AbstractMap](#)

**Description:**

Associates the specified value with the specified key in this map. If the map previously contained a mapping for this key, the old value is replaced.

**Example**

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");
Logger.log("Initial map elements: " + map.get(i));
```

The example above creates a new map, puts a value into and writes the result to the logger.

Package [java.util](#)

Class [HashMap](#)

**Public Method putAll**

```
void putAll(Map t)
```

**Overrides:**

⇒ [putAll](#) in class [AbstractMap](#)

**Description:**

Copies all of the mappings from the specified map to this one. These mappings replace any mappings that this map had for any of the keys currently in the specified map.

Package [java.util](#)

Class [HashMap](#)

**Public Method remove**

```
Object remove(Object key)
```

**Overrides:**

⇒ [remove](#) in class [AbstractMap](#)

**Description:**

Removes the mapping for this key from this map if present.

**Example**

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");

Logger.log("map: " + map.get(i));
map.remove(i);
Logger.log("map: " + map.get(i));
```

The example above removes the specified element from the map.

Package [java.util](#)

Class [HashMap](#)

**Public Method size**

```
int size()
```

**Overrides:**

⇒ [size](#) in class [AbstractMap](#)

**Description:**

Returns the number of key value mappings in this map.

**Example**

```
HashMap map = new HashMap();
Integer i = new Integer(0);
map.put(i, "value");
```

```
Logger.log("map size: " + map.size());
```

The example above returns the number key values.

## Package [java.util](#)

### Class HashSet

extends [AbstractSet](#) → [AbstractCollection](#) → [Object](#)  
implements [Set](#), [Cloneable](#), [Serializable](#), [Collection](#)

This class implements the *Set* interface, backed by a hash table. It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the *null* element.

#### Public Constructors

☞ [HashSet](#)

#### Public Methods

☞ [add](#)  
☞ [clear](#)  
☞ [clone](#)  
☞ [contains](#)  
☞ [isEmpty](#)  
☞ [iterator](#)  
☞ [remove](#)  
☞ [size](#)

#### Methods inherited from [java.util.AbstractSet](#)

☞ [equals](#)  
☞ [hashCode](#)

#### Methods inherited from [java.util.AbstractCollection](#)

☞ [iterator](#)  
☞ [size](#)  
☞ [add](#)  
☞ [addAll](#)  
☞ [clear](#)  
☞ [contains](#)  
☞ [containsAll](#)  
☞ [isEmpty](#)  
☞ [remove](#)  
☞ [removeAll](#)  
☞ [retainAll](#)  
☞ [toArray](#)  
☞ [toString](#)

#### Methods inherited from [java.lang.Object](#)

☞ [equals](#)  
☞ [toString](#)  
☞ [wait](#)  
☞ [hashCode](#)  
☞ [notify](#)  
☞ [notifyAll](#)

Package [java.util](#)  
Class [HashSet](#)

### **Public Constructor HashSet**

```
HashSet ()
```

```
HashSet (Collection c)
```

```
HashSet (int initialCapacity)
```

```
HashSet (int initialCapacity, float loadFactor)
```

#### **Description:**

##### **constructor HashSet():**

Constructs a new, empty set and the backing *HashMap* instance has default capacity and load factor, which is 0.75.

##### **constructor HashSet(Collection c):**

Constructs a new set containing the elements in the specified collection. The capacity of the backing *HashMap* instance is twice the size of the specified collection or 11, and a default load factor 0.75 is used.

##### **constructor HashSet(int initialCapacity):**

Constructs a new and empty set. The backing *HashMap* instance has the specified initial capacity and default load factor of 0.75.

##### **constructor HashSet(int initialCapacity, float loadFactor):**

Construct a new and empty set. The backing *HashMap* instance has the specified initial capacity and the specified load factor.

Package [java.util](#)  
Class [HashSet](#)

### **Public Method add**

```
boolean add (Object o)
```

#### **Overrides:**

☐ [add](#) in class [AbstractCollection](#)

#### **Description:**

Adds the specified element to this set if it is not already present. Implements interface method [Set.add](#).

## Example

```
HashSet set = new HashSet();
set.add("value1");
set.add("value2");

Logger.log("set: " + set);
```

The example above creates a new hash set and adds values to the set. Afterwards, the result is written to the logger.

Package [java.util](#)

Class [HashSet](#)

### Public Method clear

```
void clear()
```

#### Overrides:

▣ [clear](#) in class [AbstractCollection](#)

#### Description:

Removes all of the elements from this set. Implements interface method [Set.clear](#).

## Example

```
HashSet set = new HashSet();
set.add("value1");
set.add("value2");

Logger.log("set: " + set);
set.clear();
Logger.log("set: " + set);
```

The example above creates a new hash set and adds values to the set. Afterwards, the set is cleared.

Package [java.util](#)

Class [HashSet](#)

### Public Method clone

```
Object clone()
```

#### Overrides:

▣ [clone](#) in class [Object](#)

#### Throws:

▣ [Error](#)

**Description:**

Returns a shallow copy of this *HashSet* instance. A shallow copy just copies the values of the references in the class (A deep copy copies the values). The elements themselves are not cloned.

**Example**

```
HashSet set1 = new HashSet();
HashSet set2 = new HashSet();
set1.add("value1");
set1.add("value2");

set2 = (HashSet)set1.clone();

Logger.log("set1: " + set1);
Logger.log("set2: " + set2);
```

The example above creates a new hash set and clones it to another instance of a hash set.

**Package [java.util](#)  
Class [HashSet](#)****Public Method contains**

```
boolean contains(Object o)
```

**Overrides:**

☰ [contains](#) in class [AbstractCollection](#)

**Description:**

Returns *true* if this set contains the specified element.

**Example**

```
HashSet set = new HashSet();
set.add("value1");
set.add("value2");

if (set.contains("value1"))
    Logger.log("This set contains the value: value1");
```

The example above checks if set contains the *value1* string.

**Package [java.util](#)  
Class [HashSet](#)****Public Method isEmpty**

```
boolean isEmpty()
```



**Overrides:**

☰ [isEmpty](#) in class [AbstractCollection](#)

**Description:**

Returns *true* if this set contains no elements.

**Example**

```
HashSet set = new HashSet();
if (set.isEmpty())
    Logger.log("This set is empty");
```

The example above checks if the set is empty.

**Package [java.util](#)**  
**Class [HashSet](#)****Public Method iterator**

```
Iterator iterator()
```

**Overrides:**

☰ [iterator](#) in class [AbstractCollection](#)

**Description:**

Returns an iterator over the elements in this set. The elements are returned in no particular order. Implements interface method [Set.iterator](#).

**Example**

```
HashSet set = new HashSet();
set.add("value1");
set.add("value2");

Iterator iterator = set.iterator();

while (iterator.hasNext())
    Logger.log("value: " + iterator.next());
```

The example above iterates over the set.

**Package [java.util](#)**  
**Class [HashSet](#)****Public Method remove**

```
boolean remove(Object o)
```

**Overrides:**

☞ [remove](#) in class [AbstractCollection](#)

**Description:**

Removes the given element from this set if it is present.

**Example**

```
HashSet set = new HashSet();
set.add("value1");
set.add("value2");

set.remove("value2");
Logger.log("set: " + set);
```

The example above removes the *value2* string from the set.

Package [java.util](#)

**Class [HashSet](#)****Public Method size**

```
int size()
```

**Overrides:**

☞ [size](#) in class [AbstractCollection](#)

**Description:**

Returns the number of elements in this set.

**Example**

```
HashSet set = new HashSet();
set.add("value1");
set.add("value2");

Logger.log("set size: " + set.size());
```

The example above writes the number of elements to the logger.

Package [java.util](#)

**Class [Hashtable](#)**

extends [Dictionary](#) → [Object](#)

implements [Map](#), [Cloneable](#), [Serializable](#)

This *Hashtable* class maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the *hashCode* method and the *equals* method.

It's recommended to use Use [HashMap](#) instead of Hashtable.

## Public Constructors

- ▣ [Hashtable](#)

## Public Methods

- ▣ [clear](#)
- ▣ [clone](#)
- ▣ [contains](#)
- ▣ [containsKey](#)
- ▣ [containsValue](#)
- ▣ [elements](#)
- ▣ [entrySet](#)
- ▣ [equals](#)
- ▣ [get](#)
- ▣ [hashCode](#)
- ▣ [isEmpty](#)
- ▣ [keys](#)
- ▣ [keySet](#)
- ▣ [put](#)
- ▣ [putAll](#)
- ▣ [remove](#)
- ▣ [size](#)
- ▣ [toString](#)
- ▣ [values](#)

## Methods inherited from [java.util.Dictionary](#)

- ▣ [elements](#)
- ▣ [get](#)
- ▣ [isEmpty](#)
- ▣ [keys](#)
- ▣ [put](#)
- ▣ [remove](#)
- ▣ [size](#)

## Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
- ▣ [toString](#)
- ▣ [wait](#)
- ▣ [hashCode](#)
- ▣ [notify](#)
- ▣ [notifyAll](#)

## Package [java.util](#)

## Class [Hashtable](#)

### Public Constructor Hashtable

```
Hashtable ()
```

```
Hashtable (int initialCapacity)
```

```
Hashtable (int initialCapacity, float loadFactor)
```

```
Hashtable (Map t)
```

**Description:****constructor Hashtable():**

Constructs a new and empty hashtable with a default capacity and load factor of 0.75.

**constructor Hashtable(int initialCapacity):**

Constructs a new and empty hashtable with the specified initial capacity and default load factor of 0.75.

**constructor Hashtable(int initialCapacity, float loadFactor):**

Constructs a new and empty hashtable with the specified initial capacity and the specified load factor.

**constructor Hashtable(Map t):**

Constructs a new Hashtable with the same mappings as the given map. The hashtable is created with a capacity of twice the number of mappings in the given map of 11, and a default load factor of 0.75.

Package [java.util](#)

**Class [Hashtable](#)****Public Method clear**

```
void clear()
```

**Description:**

Clears this hashtable so that it contains no keys.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log("hashTable: " + hashTable.get("2"));
hashTable.clear();
Logger.log("hashTable: " + hashTable.get("2"));
```

The example above creates a new hashtable with values. Afterwards, the hashtable is cleared.

Package [java.util](#)

**Class [Hashtable](#)****Public Method clone**

```
Object clone()
```

## Overrides:

☰ [clone](#) in class [Object](#)

## Description:

Creates a shallow copy of this hashtable. All the structure of this hashtable itself is copied, but the keys and values are not cloned.

## Example

```
Hashtable hashTable1 = new Hashtable();
Hashtable hashTable2 = new Hashtable();
hashTable1.put("2", "a");

Logger.log("hashTable: " + hashTable1.get("2"));
Logger.log("hashTable: " + hashTable2.get("2"));
hashTable2 = (Hashtable)hashTable1.clone();
Logger.log("hashTable: " + hashTable1.get("2"));
Logger.log("hashTable: " + hashTable2.get("2"));
```

The example above creates two hashtables and clones one. Afterwards, the result is written to the logger.

## Package [java.util](#) Class [Hashtable](#)

### Public Method contains

```
boolean contains(Object val)
```

### Throws:

☰ [NullPointerException](#)

## Description:

Tests if some key maps into the specified value in this hashtable. This method is identical in functionality to *containsValue*.

## Example

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

if (hashTable.contains("h"))
    Logger.log("The hashtable contains a h");
```

The example above checks if the hashtable contains a *h* string.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method containsKey**

```
boolean containsKey(Object key)
```

**Throws:**

☞ [NullPointerException](#)

**Description:**

Tests if the specified object is a key in this hashtable.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

if (hashTable.containsKey("1"))
    Logger.log("The hashtable contains a h");
```

The example above checks if the hashtable contains a *h* string.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method containsValue**

```
boolean containsValue(Object val)
```

**Throws:**

☞ [NullPointerException](#)

**Description:**

Returns *true* if this hashtable maps one or more keys to this value. This method is identical in functionality to the *contains* method.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");
```

```
if (hashTable.containsValue("h"))
    Logger.log("The hashtable contains a h");
```

The example above checks if the hashtable contains a *h* string.

Package [java.util](#)  
Class [Hashtable](#)

### Public Method elements

```
Enumeration elements()
```

#### Overrides:

☰ [elements](#) in class [Dictionary](#)

#### Description:

Returns an enumeration of the values in this hashtable. Use the enumeration methods on the returned object to fetch the element sequentially.

#### Example

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Enumeration enum = hashTable.elements();

while (enum.hasMoreElements())
    Logger.log("element: " + enum.nextElement());
```

The example above returns the elements from the hashtable and writes the result to the logger.

Package [java.util](#)  
Class [Hashtable](#)

### Public Method entrySet

```
Set entrySet()
```

#### Description:

Returns a set view of the entries contained in this hashtable. Each element in this collection is a *Map.Entry*. The set is backed by the hashtable, so changes to the hashtable are reflected in the set, and vice-versa. The set supports element *removal*, but not element *add* methods.

#### Example

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
```

```
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Set set = hashTable.entrySet();
Logger.log("Set: "+ set);
```

The `entrySet()` method returns a set which is written to the logger.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method equals**

```
boolean equals(Object o)
```

**Overrides:**

☞ [equals](#) in class [Object](#)

**Description:**

Compares the specified object with this map for equality, as per the definition in the map interface.

**Example**

```
Hashtable hashTable1 = new Hashtable();
Hashtable hashTable2 = new Hashtable();
hashTable1.put("1", "h");
hashTable1.put("2", "a");
hashTable1.put("3", "s");
hashTable1.put("4", "h");

hashTable2 = (Hashtable)hashTable1.clone();

if (hashTable1.equals(hashTable2))
    Logger.log("hashTable1 equals hashTable2");
```

The example above compares both hashtables with each other.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method get**

```
Object get(Object key)
```

**Overrides:**

☞ [get](#) in class [Dictionary](#)



**Throws:**

⇒ [NullPointerException](#)

**Description:**

Returns the value to which the specified key is mapped in this hashtable.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log("hashTable: " + hashTable.get("3"));
```

The example above writes the third element to the logger.

Package [java.util](#)

Class [Hashtable](#)

**Public Method hashCode**

```
int hashCode()
```

**Overrides:**

⇒ [hashCode](#) in class [Object](#)

**Description:**

Returns the hash code value for this map as per the definition in the map interface.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log("" + hashTable.hashCode());
```

The example above writes the hashcode to the logger.

Package [java.util](#)

Class [Hashtable](#)

**Public Method isEmpty**

```
boolean isEmpty()
```

**Overrides:**

☞ [isEmpty](#) in class [Dictionary](#)

**Description:**

Tests if this hashtable maps no keys to values.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log("is the hash table empty: " + hashTable.isEmpty());
```

The example above writes the result of the *isEmpty()* method to the logger.

Package [java.util](#)

**Class [Hashtable](#)****Public Method keys**

Enumeration **keys()**

**Overrides:**

☞ [keys](#) in class [Dictionary](#)

**Description:**

Returns an enumeration of the keys in this hashtable.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Enumeration enum = hashTable.keys();

while (enum.hasMoreElements())
    Logger.log("element: " + enum.nextElement());
```

The example above returns the keys from the hashtable and writes the result to the logger.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method [keySet](#)**

```
Set keySet ()
```

**Description:**

Returns a set view of the keys contained in this hashtable. The set is backed by the hashtable, so changes to the hashtable are reflected in the set, and vice-versa. The set supports element *removal*, but not element *add* methods.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Set set = hashTable.keySet();
Logger.log("Set: " + set);
```

The example above returns the keys from the hashtable and writes the result to the logger.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method [put](#)**

```
Object put(Object key, Object val)
```

**Overrides:**

⇒ [put](#) in class [Dictionary](#)

**Throws:**

⇒ [NullPointerException](#)

**Description:**

This method maps the specified *key* to the specified *value* in this hashtable. Neither the *key* nor the *value* can be *null*.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
```

The example above returns *puts* the specified element into the hashtable.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method putAll**

```
void putAll(Map t)
```

**Description:**

Copies all of the mappings from the specified map to this hashtable. These mappings will replace any mappings that this hashtable had for any of the keys currently in the specified map.

**Example**

```
Hashtable hashTable = new Hashtable();  
Map map = new HashMap();  
map.put("1", "h");  
map.put("2", "a");  
map.put("3", "s");  
map.put("4", "h");  
  
hashTable.putAll(map);
```

The example above puts all elements from the hashmap into the hashtable.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method remove**

```
Object remove(Object key)
```

**Overrides:**

☞ [remove](#) in class [Dictionary](#)

**Description:**

Removes the key from this hashtable. This method does nothing if the key is not in the hashtable.

**Example**

```
Hashtable hashTable = new Hashtable();  
hashTable.put("1", "h");  
hashTable.put("2", "a");  
hashTable.put("3", "s");  
hashTable.put("4", "h");  
  
Set set = hashTable.keySet();  
Logger.log("Set: " + set);  
  
hashTable.remove("2");
```

```
set = hashTable.keySet();
Logger.log("Set: "+ set);
```

The example above removes the specified element from the hashtable and writes the result to the logger.

Package [java.util](#)  
Class [Hashtable](#)

**Public Method `size`**

```
int size()
```

**Overrides:**

☞ [size](#) in class [Dictionary](#)

**Description:**

Returns the number of keys in this hashtable.

**Example**

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log("hashTable size: " + hashTable.size());
```

The example above returns the number of the currently stored elements

Package [java.util](#)  
Class [Hashtable](#)

**Public Method `toString`**

```
String toString()
```

**Overrides:**

☞ [toString](#) in class [Object](#)

**Description:**

Returns a string representation of this hashtable in the form of a set of entries, enclosed in braces and separated by the ASCII characters comma and space. Each entry is rendered as the key, an equals sign, and the associated element, where the `toString` method is used to convert the key and element to string.

## Example

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log(hashTable.toString());
```

The example above returns the elements from the hashtable and writes the result to the logger.

## Package [java.util](#) Class [Hashtable](#)

### Public Method values

```
Collection values()
```

#### Description:

Returns a collection view of the values contained in this hashtable. The collection is backed by the hashtable, so changes to the hashtable are reflected in the collection, and vica-versa. The collection supports element *removal*, but not element *add* methods.

## Example

```
Hashtable hashTable = new Hashtable();
hashTable.put("1", "h");
hashTable.put("2", "a");
hashTable.put("3", "s");
hashTable.put("4", "h");

Logger.log("" + hashTable.values());
```

The example above returns the collection view of the values in the hashtable.

## Package [java.util](#)

### Interface Iterator

All containers have methods to put objects in and get objects out. An iterator is an object used to get elements out of a container and present them to the user. Via the iterator abstraction the container simply appears to be a sequence.

An object that implements the Iterator interface generates a series of elements, one at a time. Calls to the next method return successive elements of the series, i.e. an iterator over an collection allows to walk through the elements in the collection in a well defined way, allowing also to remove elements (in contrast to [Enumeration](#)).

Every container allows to retrieve an iterator to move through the elements in the container.

## Public Methods

- [hasNext](#)
- [next](#)
- [remove](#)

## Examples

```
Iterator it = MyHashMap.entrySet().iterator(); // get HashMap iterator
Iterator it = MyArrayList.iterator(); // get ArrayList iterator
Iterator it = MyHashSet.iterator(); // get HashSet iterator
```

Package [java.util](#)

## Interface [Iterator](#)

### Public Method [hasNext](#)

```
abstract boolean hasNext()
```

#### Description:

Returns true if the container has more elements, i.e. a call to next would return an element.

Package [java.util](#)

## Interface [Iterator](#)

### Public Method [next](#)

```
abstract Object next()
```

#### Description:

Returns the next element in the container. Note that at this level of abstraction there is no order of the elements presumed.

Note that the returned object has to be cast to the desired type (see the example below).

## Examples

```
// Map Example
Map MyMap = new HashMap(); // create Map
MyMap.put("First", new Integer(1)); // put Map element
Iterator it = MyMap.entrySet().iterator(); // get Map iterator
Integer element = (Integer)it.next(); // get next element from iterator
```

```
// List Example
List MyList = new ArrayList(); // create List
MyList.add(new Float(1.23f)); // add List element
Iterator it = MyList.iterator(); // get List iterator
Float element = (Float)it.next(); // get next element from iterator
```

```
// Set Example
Set MySet = new HashSet(); // create Set
MySet.add(new Byte((byte)0x88)); // add Set element
```

```
Iterator it = MySet.iterator(); // get Set iterator
Byte element = (Byte)it.next(); // get next element from iterator
```

Package [java.util](#)

## Interface [Iterator](#)

### Public Method [remove](#)

```
abstract void remove()
```

#### Description:

Removes the last element returned by the iterator (last call to next) from the corresponding collection. This method can be called only once per call to next.

Package [java.util](#)

## Class [LinkedList](#)

extends [AbstractSequentialList](#) → [AbstractList](#) → [AbstractCollection](#) → [Object](#)

implements [List](#), [Cloneable](#), [Serializable](#), [Collection](#)

Linked list implementation of the *List* interface. It implements all the optional list operations, and provides uniformly named methods to *get*, *remove*, and *insert* an element at the beginning and end of the list. This implementation is not synchronized.

### Public Constructors

☞ [LinkedList](#)

### Public Methods

☞ [add](#)  
☞ [addAll](#)  
☞ [addFirst](#)  
☞ [addLast](#)  
☞ [clear](#)  
☞ [clone](#)  
☞ [contains](#)  
☞ [get](#)  
☞ [getFirst](#)  
☞ [getLast](#)  
☞ [indexOf](#)  
☞ [lastIndexOf](#)  
☞ [listIterator](#)  
☞ [remove](#)  
☞ [removeFirst](#)  
☞ [removeLast](#)  
☞ [set](#)  
☞ [size](#)  
☞ [toArray](#)

### Methods inherited from [java.util.AbstractSequentialList](#)

☞ [listIterator](#)  
☞ [add](#)  
☞ [addAll](#)  
☞ [get](#)  
☞ [iterator](#)  
☞ [remove](#)



▣ [set](#)

### Methods inherited from [java.util.AbstractList](#)

▣ [get](#)  
▣ [add](#)  
▣ [addAll](#)  
▣ [clear](#)  
▣ [equals](#)  
▣ [hashCode](#)  
▣ [indexOf](#)  
▣ [iterator](#)  
▣ [lastIndexOf](#)  
▣ [listIterator](#)  
▣ [remove](#)  
▣ [set](#)  
▣ [subList](#)

### Methods inherited from [java.util.AbstractCollection](#)

▣ [iterator](#)  
▣ [size](#)  
▣ [add](#)  
▣ [addAll](#)  
▣ [clear](#)  
▣ [contains](#)  
▣ [containsAll](#)  
▣ [isEmpty](#)  
▣ [remove](#)  
▣ [removeAll](#)  
▣ [retainAll](#)  
▣ [toArray](#)  
▣ [toString](#)

### Methods inherited from [java.lang.Object](#)

▣ [equals](#)  
▣ [toString](#)  
▣ [wait](#)  
▣ [hashCode](#)  
▣ [notify](#)  
▣ [notifyAll](#)

Package [java.util](#)

## Class [LinkedList](#)

### Public Constructor [LinkedList](#)

```
LinkedList ()
```

```
LinkedList (Collection c)
```

#### Description:

#### constructor [LinkedList](#)():

Constructs an empty list.

**constructor `LinkedList(Collection c)`:**

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Package [java.util](#)

**Class [LinkedList](#)****Public Method `add`**

```
void add(int index, Object element)
```

**Overrides:**

⇒ [add](#) in class [AbstractSequentialList](#)

```
boolean add(Object o)
```

**Overrides:**

⇒ [add](#) in class [AbstractList](#)

**Description:****`add(int index, Object element)` method:**

Inserts the specified element at the position specified by the index argument. Shifts the element currently at that position and any subsequent elements to the right.

**`add(Object element)` method:**

Adds the specified element to the end of this list.

**Example**

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Logger.log(linkedList.toString());
```

The example above constructs a new `LinkedList`, adds elements to the list, and writes the elements to the logger.

Package [java.util](#)

**Class [LinkedList](#)****Public Method `addAll`**

```
boolean addAll(Collection c)
```

**Overrides:**

⇒ [addAll](#) in class [AbstractCollection](#)

```
boolean addAll(int index, Collection c)
```

**Overrides:**

⇒ [addAll](#) in class [AbstractSequentialList](#)

**Throws:**

⇒ [IndexOutOfBoundsException](#)

**Description:****addAll(Collection c) method:**

Adds all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. The behavior of this operation is undefined if the specified collection is modified while the operation is in progress.

**addAll(int index, Collection c) method:**

Inserts all the elements in the specified collection into this list, starting at the position specified by the index argument. Shifts the element currently at that position and any subsequent elements to the right. The new elements will appear in the list in the order that they are returned by the specified collection's iterator.

**Example**

```
ArrayList arrayList = new ArrayList();
arrayList.add("L");
arrayList.add("i");
arrayList.add("n");
arrayList.add("k");
arrayList.add("e");
arrayList.add("d");

LinkedList linkedList = new LinkedList();
linkedList.addAll(arrayList);

Logger.log(linkedList.toString());
```

The example above constructs a new *LinkedList*, and adds all elements of the *ArrayList*. The result is written to the logger.

Package [java.util](#)

Class [LinkedList](#)

**Public Method addFirst**

```
void addFirst(Object o)
```

**Description:**

Inserts the given element at the beginning of this list.

**Example**

```
LinkedList linkedList = new LinkedList();
linkedList.add("List");

linkedList.addFirst("Linked");

Logger.log(linkedList.toString());
```

The example above constructs a new *LinkedList*, adds elements to the list, and inserts an element at the beginning of this list.

Package [java.util](#)

Class [LinkedList](#)

**Public Method addLast**

```
void addLast(Object o)
```

**Description:**

Appends the given element to the end of this list.

**Example**

```
LinkedList linkedList = new LinkedList();
linkedList.add("List");

linkedList.addLast("Linked");

Logger.log(linkedList.toString());
```

The example above constructs a new *LinkedList*, adds elements to the list, and appends an element at the end of this list.

Package [java.util](#)

Class [LinkedList](#)

**Public Method clear**

```
void clear()
```

**Overrides:**

☰ [clear](#) in class [AbstractList](#)

**Description:**

Removes all the elements from the list.

## Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("LinkedList");

Logger.log(linkedList.toString());
linkedList.clear();
Logger.log(linkedList.toString());
```

The example above constructs a new *LinkedList*, adds an element to the list, and clears the list afterwards.

## Package [java.util](#) Class [LinkedList](#)

### Public Method clone

```
Object clone()
```

#### Overrides:

☞ [clone](#) in class [Object](#)

#### Throws:

☞ [Error](#)

#### Description:

Returns a shallow copy of this *LinkedList*.

## Example

```
LinkedList linkedList1 = new LinkedList();
LinkedList linkedList2 = new LinkedList();
linkedList1.add("LinkedList");

Logger.log(linkedList1.toString());
Logger.log(linkedList2.toString());
linkedList2 = (LinkedList)linkedList1.clone();
Logger.log(linkedList1.toString());
Logger.log(linkedList2.toString());
```

The example above creates a shallow copy of the *linkedList1* object.

## Package [java.util](#) Class [LinkedList](#)

### Public Method contains

```
boolean contains(Object o)
```

**Overrides:**

☞ [contains](#) in class [AbstractCollection](#)

**Description:**

Returns *true* if this list contains the specified element. It returns *true* if this list contains at least one element *e* such that  $(o \neq null \ ? \ e \neq null \ : o.equals(e))$ .

**Example**

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Logger.log("Does the linked list contain an i string? " +
linkedList.contains("i"));
```

The example above checks if there is the specified element in the list.

Package [java.util](#)

Class [LinkedList](#)

**Public Method [get](#)**

Object **get**(int index)

**Overrides:**

☞ [get](#) in class [AbstractSequentialList](#)

**Description:**

Returns the element at the position specified by the *index* argument.

**Example**

```
LinkedList linkedList = new LinkedList();
linkedList.add("Linked");
linkedList.add("List");

Logger.log(linkedList.get(0) + " " + linkedList.get(1));
```

The *get* method returns the specified element.

Package [java.util](#)  
Class [LinkedList](#)

### Public Method `getFirst`

Object `getFirst()`

#### Throws:

☞ [NoSuchElementException](#)

#### Description:

Returns the first element in this list.

#### Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("Linked");
linkedList.add("List");

Logger.log("" + linkedList.getFirst());
```

The `getFirst` method returns the element first element in the list.

Package [java.util](#)  
Class [LinkedList](#)

### Public Method `getLast`

Object `getLast()`

#### Throws:

☞ [NoSuchElementException](#)

#### Description:

Returns the last element in this list.

#### Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("Linked");
linkedList.add("List");

Logger.log("" + linkedList.getLast());
```

The `getLast` method returns the last element in this list.

Package [java.util](#)  
Class [LinkedList](#)

### Public Method `indexOf`

```
int indexOf(Object o)
```

#### Overrides:

⇒ [indexOf](#) in class [AbstractList](#)

#### Description:

Returns the index in this list of the first occurrence of the specified element, or -1 if the list does not contain this element. This method returns the lowest index *i* such that *(o==null ? get(i)==null : o.equals(get(i)))*, or -1 if there is no such index.

#### Example

```
LinkedList linkedList = new LinkedList();  
linkedList.add("Linked");  
linkedList.add("List");  
  
Logger.log("The index of List: " + linkedList.indexOf("List"));
```

The `indexOf` method returns the index of the specified element.

Package [java.util](#)  
Class [LinkedList](#)

### Public Method `lastIndexOf`

```
int lastIndexOf(Object o)
```

#### Overrides:

⇒ [lastIndexOf](#) in class [AbstractList](#)

#### Description:

Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element. This method returns the highest index *i* such that *(o==null ? get(i)==null : o.equals(get(i)))*, or -1 if there is no such index.

#### Example

```
LinkedList linkedList = new LinkedList();  
linkedList.add("Linked");  
linkedList.add("List");  
linkedList.add("Linked");  
linkedList.add("List");  
  
Logger.log("the last index: " + linkedList.lastIndexOf("Linked"));
```



The *lastIndexOf* method returns the index of the last occurrence of the specified element in this list.

Package [java.util](#)

Class [LinkedList](#)

### Public Method `listIterator`

```
ListIterator listIterator(int index)
```

#### Overrides:

↳ [listIterator](#) in class [AbstractSequentialList](#)

#### Description:

Returns a list-iterator of the elements in this list, starting at the specified position in the list. The list-iterator is fail-fast if the list is structurally modified at any time after the iterator is created, in any way except through the list-iterator's own *remove* or *add* methods, the list-iterator will throw a *ConcurrentModificationException*.

#### Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

ListIterator iterator = linkedList.listIterator();
while (iterator.hasNext())
    Logger.log("" + iterator.next());
```

The *listIterator* method returns a *ListIterator* object with specific iterator functionalities for a list.

Package [java.util](#)

Class [LinkedList](#)

### Public Method `remove`

```
Object remove(int index)
```

#### Overrides:

↳ [remove](#) in class [AbstractSequentialList](#)

```
boolean remove(Object o)
```

## Overrides:

☞ [remove](#) in class [AbstractCollection](#)

## Description:

Removes the element at the specified position in this list. Shifts any subsequent elements to the left, and returns the element that was removed from the list.

## Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("List");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Logger.log(linkedList.toString());
linkedList.remove("List");
Logger.log(linkedList.toString());
```

The `remove` method removes the specified element for the list.

Package [java.util](#)

Class [LinkedList](#)

## Public Method `removeFirst`

Object `removeFirst()`

## Throws:

☞ [NoSuchElementException](#)

## Description:

Removes and returns the first element from this list.

## Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("List");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");
```

```
Logger.log(linkedList.toString());
linkedList.removeFirst();
Logger.log(linkedList.toString());
```

The *removeFirst* method removes the first element in the list.

Package [java.util](#)  
Class [LinkedList](#)

### **Public Method removeLast**

Object **removeLast**()

#### **Throws:**

⇒ [NoSuchElementException](#)

#### **Description:**

Removes and returns the last element from this list.

#### **Example**

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("List");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Logger.log(linkedList.toString());
linkedList.removeLast();
Logger.log(linkedList.toString());
```

The *removeLast* method removes the last element in the list.

Package [java.util](#)  
Class [LinkedList](#)

### **Public Method set**

Object **set**(int index, Object element)

#### **Overrides:**

⇒ [set](#) in class [AbstractSequentialList](#)

#### **Description:**

Replaces the element at the specified position in this list with the specified element.

## Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("List");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Logger.log(linkedList.toString());
linkedList.set(2, "n");
Logger.log(linkedList.toString());
```

The `set` method replaces the element at the specified position with the element.

## Package [java.util](#) Class [LinkedList](#)

### Public Method size

```
int size()
```

#### Overrides:

☰ [size](#) in class [AbstractCollection](#)

#### Description:

Returns the number of elements in this list.

## Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("List");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Logger.log("" + linkedList.size());
```

The `size` method returns the number of elements in the list.

## Package [java.util](#) Class [LinkedList](#)

### Public Method toArray

```
Object[] toArray()
```

## Overrides:

☞ [toArray](#) in class [AbstractCollection](#)

## Description:

Returns an array that contains all the elements in this list in the correct order. The runtime type of the returned array is that of the specified array. If the list fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this list. If the list fits the specified array with room to spare, the element in the array immediately following the end of the collection is set to null.

## Example

```
LinkedList linkedList = new LinkedList();
linkedList.add("L");
linkedList.add("i");
linkedList.add("n");
linkedList.add("List");
linkedList.add("k");
linkedList.add("e");
linkedList.add("d");

Object[] arr = linkedList.toArray();
for (int i = 0; i < linkedList.size(); i++)
    Logger.log("" + arr[i]);
```

The example above returns an array that contains the elements from the list.

## Package [java.util](#)

## Interface List

implements [Collection](#)

The interface *List* is an ordered collection. The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index, and search for elements in the list.

It is allowed to duplicate elements. More formally, lists typically allow pairs of elements  $e_1$  and  $e_2$  such that  $e_1.equals(e_2)$ , and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

Use the concrete implementations [ArrayList](#) and [LinkedList](#).

## Public Methods

- ☞ [add](#)
- ☞ [addAll](#)
- ☞ [clear](#)
- ☞ [contains](#)
- ☞ [containsAll](#)
- ☞ [equals](#)
- ☞ [get](#)

- [hashCode](#)
- [indexOf](#)
- [isEmpty](#)
- [iterator](#)
- [lastIndexOf](#)
- [listIterator](#)
- [remove](#)
- [removeAll](#)
- [retainAll](#)
- [set](#)
- [size](#)
- [subList](#)
- [toArray](#)

## Package [java.util](#) Interface [List](#)

### Public Method add

```
abstract void add(int index, Object o)
abstract boolean add(Object o)
```

#### Description:

##### **add(int index, Object o) method:**

Appends the specified element to the end of this list (optional operation).

Lists that support this operation may place limitations on what elements may be added to this list. In particular, some lists will refuse to add null elements, and others will impose restrictions on the type of elements that may be added. List classes should clearly specify in their documentation any restrictions on what elements may be added.

##### **add(Object o) method:**

Inserts the specified element at the specified position in this list (optional operation).

Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

Note that basic types cannot be used with this method because they are not of type object. If you want to add e.g. a float value use the Float class, see the ArrayList example below.

#### Example

```
List myList = new ArrayList(); // ArrayList implements List
myList.add(new Float(1.23f)); // create a Float-object
```

## Package [java.util](#) Interface [List](#)

### Public Method addAll

```
abstract boolean addAll(Collection c)
```

```
abstract boolean addAll(int index, Collection c)
```

**Description:****addAll(Collection c) method:**

Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).

**Note**

☞ The behavior of this operation is unspecified if the specified collection is modified while the operation is in progress. This will occur if the specified collection is this list, and it's nonempty.

**addAll(int index, Collection c) method:**

Inserts all of the elements in the specified collection into this list at the specified position (optional operation).

Shifts the element currently at that position (if any) and any subsequent elements to the right (increases their indices). The new elements will appear in this list in the order that they are returned by the specified collection's iterator.

**Note**

☞ The behavior of this operation is unspecified if the specified collection is modified while the operation is in progress. This will occur if the specified collection is this list, and it's nonempty.

**Package [java.util](#)****Interface [List](#)****Public Method clear**

```
abstract void clear()
```

**Description:**

Removes all of the elements from this list (optional operation).

This list will be empty after this call returns (unless it throws an exception).

**Package [java.util](#)****Interface [List](#)****Public Method contains**

```
abstract boolean contains(Object o)
```

**Description:**

Returns *true* if this list contains the specified element. More formally, returns *true* if and only if this list contains at least one element *e* such that  $(o == null ? e == null : o.equals(e))$ .

## Package [java.util](#) Interface [List](#)

### Public Method containsAll

```
abstract boolean containsAll(Collection c)
```

#### Description:

Returns *true* if this list contains all of the elements of the specified collection.

## Package [java.util](#) Interface [List](#)

### Public Method equals

```
abstract boolean equals(Object o)
```

#### Overrides:

⇒ [equals](#) in class [Object](#)

#### Description:

Compares the specified object with this list for equality. Returns *true* if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are equal. Two elements *e1* and *e2* are equal if  $(e1 == null ? e2 == null : e1.equals(e2))$ . In other words, two lists are defined to be equal if they contain the same elements in the same order. This definition ensures that the equals method works properly across different implementations of the *List* interface.

## Package [java.util](#) Interface [List](#)

### Public Method get

```
abstract Object get(int index)
```

#### Description:

Returns the element at the specified position in this list.

## Package [java.util](#) Interface [List](#)

### Public Method hashCode

```
abstract int hashCode()
```

#### Overrides:

⇒ [hashCode](#) in class [Object](#)



**Description:**

Returns the hash code value for this list.

Package [java.util](#)

Interface [List](#)

**Public Method `indexOf`**

```
abstract int indexOf(Object o)
```

**Description:**

Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.

Package [java.util](#)

Interface [List](#)

**Public Method `isEmpty`**

```
abstract boolean isEmpty()
```

**Description:**

Returns *true* if this list contains no elements.

Package [java.util](#)

Interface [List](#)

**Public Method `iterator`**

```
abstract Iterator iterator()
```

**Description:**

Returns an iterator over the elements in this list in proper sequence.

Package [java.util](#)

Interface [List](#)

**Public Method `lastIndexOf`**

```
abstract int lastIndexOf(Object o)
```

**Description:**

Returns the index in this list of the last occurrence of the specified element, of -1 if this list does not contain this element.

## Package [java.util](#) Interface [List](#)

### **Public Method listIterator**

```
abstract ListIterator listIterator()  
  
abstract ListIterator listIterator(int index)
```

#### **Description:**

##### **listIterator() method:**

Returns a list iterator of the elements in this list in a proper sequence.

##### **listIterator(int index) method:**

Returns a list iterator of the elements in this list in a proper sequence, starting at the specified position in this list. The specified index indicates the first element that would be returned by an initial call to the *next* method. An initial call to the *previous* method would return the element with the specified index minus one.

## Package [java.util](#) Interface [List](#)

### **Public Method remove**

```
abstract Object remove(int index)  
  
abstract boolean remove(Object o)
```

#### **Description:**

##### **remove(int index) method:**

Removes the element at the specified position in this list (optional operation).

Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the list.

##### **remove(Object o) method:**

Returns the index in this list of the first occurrence of the specified element, or -1 if this list does not contain this element.

## Package [java.util](#) Interface [List](#)

### **Public Method removeAll**

```
abstract boolean removeAll(Collection c)
```

**Description:**

Removes from this list all the elements that are contained in the specified collection (optional operation).

Package [java.util](#)

Interface [List](#)

**Public Method retainAll**

```
abstract boolean retainAll(Collection c)
```

**Description:**

Retains only the elements in this list that are contained in the specified collection (optional operation).

In other words, removes from this list all the elements that are not contained in the specified collection.

Package [java.util](#)

Interface [List](#)

**Public Method set**

```
abstract Object set(int index, Object o)
```

**Description:**

Replaces the element at the specified position in this list with the specified element (optional operation).

Package [java.util](#)

Interface [List](#)

**Public Method size**

```
abstract int size()
```

**Description:**

Returns the number of elements in this list. If this list contains more than *Integer.MAX\_VALUE* elements, returns *Integer.MAX\_VALUE*

Package [java.util](#)

Interface [List](#)

**Public Method subList**

```
abstract List subList(int fromIndex, int toIndex)
```

**Description:**

Returns a view of the portion of this list between the specified *fromIndex*, inclusive, and *toIndex*, exclusive. If *fromIndex* and *toIndex* are equal, the returned list is empty. The returned list is

backed by this list, so changes in the returned list are reflected in this list, and vice-versa. The returned list supports all of the optional list operations supported by this list.

This method eliminates the need for explicit range operations (of the sort that commonly exist for arrays). Any operation that expects a list can be used as a range operation by passing a `subList` view instead of a whole list.

Similar idioms may be constructed for `indexOf` and `lastIndexOf`, and all of the algorithms in the `Collections` class can be applied to a `subList`. The semantics of this list returned by this method become undefined if the backing list (i.e., this list) is structurally modified in any way other than via the returned list.

Package [java.util](#)

## Interface [List](#)

### Public Method `toArray`

```
abstract Object[] toArray()
```

#### Description:

Returns an array containing all of the elements in this list in a proper sequence. Obeys the general contract of the `Collection.toArray` method.

Package [java.util](#)

## Interface `ListIterator`

implements [Iterator](#)

An iterator for lists that allows the programmer to traverse the list in either direction and modify the list during iteration.

### Public Methods

- [add](#)
- [hasNext](#)
- [hasPrevious](#)
- [next](#)
- [nextIndex](#)
- [previous](#)
- [previousIndex](#)
- [remove](#)
- [set](#)

Package [java.util](#)

## Interface [ListIterator](#)

### Public Method `add`

```
abstract void add(Object o)
```

#### Description:

Inserts the specified element into the list (optional operation).

The element is inserted immediately before the next element that would be returned by *next*, if any, and after the next element that would be returned by *previous*, if any. If the list contains no elements, the new element becomes the sole element on the list. The new element is inserted before the implicit cursor: a subsequent call to *next* would be unaffected, and a subsequent call to *previous* would return the new element. This call increases by one the value that would be returned by a call to *nextIndex* or *previousIndex*.

Package [java.util](#)

## Interface [ListIterator](#)

### Public Method hasNext

```
abstract boolean hasNext ()
```

#### Description:

Returns *true* if this list iterator has more elements when traversing the list in the forward direction. (Returns *true* if *next* would return an element rather than throwing an exception.)

Package [java.util](#)

## Interface [ListIterator](#)

### Public Method hasPrevious

```
abstract boolean hasPrevious ()
```

#### Description:

Returns *true* if this list iterator has more elements when traversing the list in the reverse direction. (Returns *true* if *previous* would return an element rather than throwing an exception.)

Package [java.util](#)

## Interface [ListIterator](#)

### Public Method next

```
abstract Object next ()
```

#### Description:

Returns the next element in the list. This method may be called repeatedly to iterate through the list, of intermixed with calls to *previous* to go back and forth. The alternating call to *next* and *previous* will return the same element repeatedly.

Package [java.util](#)

## Interface [ListIterator](#)

### Public Method nextIndex

```
abstract int nextIndex ()
```

**Description:**

Returns the index of the element that would be returned by a subsequent call to *next*. Returns list size if the list iterator is at the end of the list.

Package [java.util](#)

**Interface [ListIterator](#)****Public Method previous**

```
abstract Object previous ()
```

**Description:**

Returns the previous element in the list. This method may be called repeatedly to iterate through the list backwards, or intermixed with calls to *next* to go back and forth. The alternating calls to *next* and *previous* will return the same element repeatedly.

Package [java.util](#)

**Interface [ListIterator](#)****Public Method previousIndex**

```
abstract int previousIndex ()
```

**Description:**

Returns the index of the element that would be returned by a subsequent call to *previous*. Returns -1 if the list iterator is at the beginning of the list.

Package [java.util](#)

**Interface [ListIterator](#)****Public Method remove**

```
abstract void remove ()
```

**Description:**

Removes from the list the last element that was returned by *next* or *previous*> (optional operation).

This call can only be made once per call to *next* or *previous*. It can be made only if *ListIterator.add* has not been called after the last call to *next* or *previous*.

Package [java.util](#)

**Interface [ListIterator](#)****Public Method set**

```
abstract void set (Object o)
```

## Description:

Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

This call can be made only if neither `ListIterator.remove` nor `ListIterator.add` have been called after the last call to `next` or `previous`.

## Package [java.util](#)

## Class `Locale`

extends [Object](#)

implements [Cloneable](#), [Serializable](#)

A `Locale` object represents a specific geographical, political, or cultural region. An operation that requires a `Locale` to perform its task is used to tailor information for the user.

Because a `Locale` object is just an identifier for a region, no validity check is performed when a `Locale` is constructed. The information can be queried once the `Locale` is created.

## Public Constructors

☐ [Locale](#)

## Public Methods

☐ [clone](#)  
☐ [equals](#)  
☐ [getCountry](#)  
☐ [getDefault](#)  
☐ [getDisplayCountry](#)  
☐ [getDisplayLanguage](#)  
☐ [getDisplayName](#)  
☐ [getDisplayVariant](#)  
☐ [getISO3Country](#)  
☐ [getISO3Language](#)  
☐ [getLanguage](#)  
☐ [getVariant](#)  
☐ [hashCode](#)  
☐ [setDefault](#)  
☐ [toString](#)

## Methods inherited from [java.lang.Object](#)

☐ [equals](#)  
☐ [toString](#)  
☐ [wait](#)  
☐ [hashCode](#)  
☐ [notify](#)  
☐ [notifyAll](#)

## Public Fields

☐ static final `Locale` CANADA  
☐ static final `Locale` CANADA\_FRENCH  
☐ static final `Locale` CHINA  
☐ static final `Locale` CHINESE  
☐ static final `Locale` ENGLISH  
☐ static final `Locale` FRANCE  
☐ static final `Locale` FRENCH

- ☞ static final Locale GERMAN
- ☞ static final Locale GERMANY
- ☞ static final Locale ITALIAN
- ☞ static final Locale ITALY
- ☞ static final Locale JAPAN
- ☞ static final Locale JAPANESE
- ☞ static final Locale KOREA
- ☞ static final Locale KOREAN
- ☞ static final Locale PRC
- ☞ static final Locale SIMPLIFIED\_CHINESE
- ☞ static final Locale TAIWAN
- ☞ static final Locale TRADITIONAL\_CHINESE
- ☞ static final Locale UK
- ☞ static final Locale US

## Package [java.util](#)

### Class [Locale](#)

#### Public Constructor Locale

```
Locale(String language, String country)
```

```
Locale(String language, String country, String variant)
```

#### Description:

##### constructor Locale(String language, String country):

Constructs a locale from language, and country. To create a locale that only identifies a language, use "" for the *country*.

##### constructor Locale(String language, String country, String variant):

Constructs a locale from language, country, and variant.

## Package [java.util](#)

### Class [Locale](#)

#### Public Method clone

```
Object clone()
```

#### Overrides:

- ☞ [clone](#) in class [Object](#)

#### Description:

Following copy from the class java.lang.Object.

#### Example

```
Locale loc1 = Locale.GERMAN;
Locale loc2 = (Locale)loc1.clone();
Logger.log(loc1.toString());
Logger.log(loc2.toString());
```



The example above copies the data from `loc1` to the `loc2` object.


Package [java.util](#)

Class [Locale](#)

### Public Method equals

```
boolean equals (Object obj)
```

#### Overrides:

 [equals](#) in class [Object](#)

#### Description:

Returns true if this *Locale* is equal to another object. A *Locale* is deemed equal to another *Locale* with identical language, country, variant, and is unequal to all other objects.

#### Example

```
Locale loc1 = Locale.GERMAN;
Locale loc2 = (Locale)loc1.clone();
Logger.log("" + loc1.equals(loc2));
```

The example above compares the two objects and writes the result to the logger.

Package [java.util](#)

Class [Locale](#)

### Public Method getCountry

```
String getCountry()
```

#### Description:

Returns the country/region code for this *Locale*, which will either be the empty string of an uppercase ISO 3166 2-letter code.

#### Example

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getCountry());
```

The *getCountry* method returns the specified country.

Package [java.util](#)

Class [Locale](#)

### Public Method getDefault

```
static Locale getDefault()
```

**Description:**

Sets the default locale for this instance of the Java Virtual Machine. The Java Virtual Machine sets the default locale during startup. It is used by many locale sensitive methods if no locale is explicitly specified.

**Example**

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getDefault());
```

The *getDefault* method returns the default value of the language and the country.

Package [java.util](#)

Class [Locale](#)

**Public Method [getDisplayCountry](#)**

```
final String getDisplayCountry()
String getDisplayCountry(Locale inLocale)
```

**Description:****getDisplayCountry() method:**

Returns a name for the locale's country that is appropriate for display to the user.

**getDisplayCountry(Locale inLocale) method:**

Returns a name for the locale's country that is appropriate for display to the user.

**Example**

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getDisplayCountry());
```

The *getDisplayCountry* method returns the specified country.

Package [java.util](#)

Class [Locale](#)

**Public Method [getDisplayLanguage](#)**

```
final String getDisplayLanguage()
String getDisplayLanguage(Locale inLocale)
```

**Description:****getDisplayLanguage() method:**

Returns a name for the locale's language that is appropriate for display to the user.

**getDisplayLanguage(Locale inLocale) method:**

Returns a name for the locale's language that is appropriate for display to the user.

**Example**

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getDisplayLanguage());
```

The *getDisplayLanguage* method returns the specified language.

Package [java.util](#)

Class [Locale](#)

**Public Method getDisplayName**

```
final String getDisplayName()
String getDisplayName(Locale inLocale)
```

**Description:****getDisplayName() method:**

Returns a whole name for the locale's that is appropriate for display to the user.

**getDisplayName(Locale inLocale) method:**

Returns a whole name for the locale's that is appropriate for display to the user.

**Example**

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getDisplayName());
```

The *getDisplayName* method returns the specified name with the language, country, and variant in one string.

Package [java.util](#)

Class [Locale](#)

**Public Method getDisplayVariant**

```
final String getDisplayVariant()
String getDisplayVariant(Locale inLocale)
```

**Description:****getDisplayVariant() method:**

Returns a name for the locale's variant that is appropriate for display to the user.

**getDisplayVariant(Locale inLocale) method:**

Returns a name for the locale's variant that is appropriate for display to the user.

**Example**

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getDisplayVariant());
```

The *getDisplayVariant* method returns the specified variant.

Package [java.util](#)

Class [Locale](#)

**Public Method [getISO3Country](#)**

```
String getISO3Country()
```

**Throws:**

⇒ [MissingResourceException](#)

**Description:**

Every call of this method results in an *MissingResourceException*. This method is currently not supported.

Package [java.util](#)

Class [Locale](#)

**Public Method [getISO3Language](#)**

```
String getISO3Language()
```

**Throws:**

⇒ [MissingResourceException](#)

**Description:**

Every call of this method results in an *MissingResourceException*. This method is currently not supported.

Package [java.util](#)

Class [Locale](#)

**Public Method [getLanguage](#)**

```
String getLanguage()
```

**Description:**

Returns the language code for this locale, which will either be the empty string or a lowercast ISO 639 code.

## Example

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getLanguage());
```

The *getLanguage* method returns the specified language.

Package [java.util](#)

Class [Locale](#)

### Public Method `getVariant`

```
String getVariant()
```

#### Description:

Returns the variant code for this locale.

## Example

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getVariant());
```

The *getVariant* method returns the specified variant.

Package [java.util](#)

Class [Locale](#)

### Public Method `hashCode`

```
int hashCode()
```

#### Overrides:

↳ [hashCode](#) in class [Object](#)

#### Description:

This method overrides hashcode from the base class.

## Example

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.hashCode());
```

The *hashCode* method returns the hashcode of this *Locale* object.

Package [java.util](#)

Class [Locale](#)

### Public Method `setDefault`

```
static void setDefault(Locale newLocale)
```

## Description:

Sets the default locale for this instance of the Java Virtual Machine. The Java Virtual Machine sets the default locale during startup based on the host environment.

## Example

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.getDefault());
loc.setDefault(loc);
Logger.log("" + loc.getDefault());
```

The example above sets up a new default value for the *Locale*.

Package [java.util](#)

Class [Locale](#)

## Public Method toString

```
final String toString()
```

## Overrides:

↳ [toString](#) in class [Object](#)

## Description:

Returns the entire locale, with the language, country, and variant.

## Example

```
Locale loc = new Locale("de", "AT", "vienna");
Logger.log("" + loc.toString());
```

The example above returns the string representation of this *Locale* object.

Package [java.util](#)

## Interface Map

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.

Use the concrete implementations [HashMap](#) and [TreeMap](#) (they differ only in performance).

## Public Methods

↳ [clear](#)  
↳ [containsKey](#)  
↳ [containsValue](#)  
↳ [entrySet](#)  
↳ [equals](#)

- ▣ [get](#)
- ▣ [hashCode](#)
- ▣ [isEmpty](#)
- ▣ [keySet](#)
- ▣ [put](#)
- ▣ [putAll](#)
- ▣ [remove](#)
- ▣ [size](#)
- ▣ [values](#)

Package [java.util](#)  
**Interface [Map](#)**

**Public Method clear**

```
abstract void clear()
```

**Description:**

Removes all mappings from this map (optional operation).

Package [java.util](#)  
**Interface [Map](#)**

**Public Method containsKey**

```
abstract boolean containsKey(Object key)
```

**Description:**

Returns *true* if this map contains no key-value mappings.

Package [java.util](#)  
**Interface [Map](#)**

**Public Method containsValue**

```
abstract boolean containsValue(Object value)
```

**Description:**

Returns *true* if this map maps one or more keys to the specified value. More formally, returns *true* if and only if this map contains at least one mapping to a value *v* such that  $(value == null ? v == null : value.equals(v))$ . This operation will probably require time linear in the map size of most implementations of the *Map* interface.

Package [java.util](#)  
**Interface [Map](#)**

**Public Method entrySet**

```
abstract Set entrySet()
```

**Description:**

Returns a set view of the mappings contained in this map. Each element in the returned set is a *Map.Entry*. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress, the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the *Iterator.remove*, *Set.remove*, *removeAll*, *retainAll* and *clear* operations. It does not support the *add* or *addAll* operations.

**Package [java.util](#)  
Interface [Map](#)****Public Method equals**

```
abstract boolean equals(Object o)
```

**Overrides:**

☞ [equals](#) in class [Object](#)

**Description:**

Compares the specified object with this map for equality. Returns *true* if the given object is also a map and the two Maps represent the same mappings. More formally, two maps *t1* and *t2* represent the same mappings if *t1.entrySet().equals(t2.entrySet())*. This ensures that the *equals* method works properly across different implementations of the *Map* interface.

**Package [java.util](#)  
Interface [Map](#)****Public Method get**

```
abstract Object get(Object key)
```

**Description:**

Returns the value to which this map maps the specified key. Returns *null* if the map contains no mapping for this key. A return value of *null* does *not* necessarily indicate that the map contains no mapping for the key; it's also possible that the map explicitly maps the key to *null*. The *containsKey* operation may be used to distinguish these two cases.

**Package [java.util](#)  
Interface [Map](#)****Public Method hashCode**

```
abstract int hashCode()
```

**Overrides:**

☞ [hashCode](#) in class [Object](#)



**Description:**

Returns the hash code value for this map. The hash code of a map is defined to be the sum of the hashCodes of each entry in the map's entrySet view. This ensures that *t1.equals(t2)* implies that *t1.hashCode()==t2.hashCode()* for any two maps *t1* and *t2*, as required by the general contract of *Object.hashCode*.

Package [java.util](#)  
Interface [Map](#)

**Public Method isEmpty**

```
abstract boolean isEmpty()
```

**Description:**

Returns *true* if this map contains no key-value mappings.

Package [java.util](#)  
Interface [Map](#)

**Public Method keySet**

```
abstract Set keySet()
```

**Description:**

Returns a set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress, the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the *Iterator.remove*, *Set.remove*, *removeAll*, *retainAll*, and *clear* operations. It does not support the *add* or *addAll* operations.

Package [java.util](#)  
Interface [Map](#)

**Public Method put**

```
abstract Object put(Object key, Object value)
```

**Description:**

Associates the specified value with the specified key in this map (optional operation).

If the map previously contained a mapping for this key, the old value is replaced.

Note that basic types cannot be used with this method because they are not of type object. If you want to map e.g. to an integer use the Integer class, see the HashMap example below.

**Example**

```
Map vehicles = new HashMap(); // HashMap implements Map
vehicles.put("Train", new Integer(10)); // create an Integer-object
```

Package [java.util](#)  
Interface [Map](#)

**Public Method putAll**

```
abstract void putAll (Map m)
```

**Description:**

Copies all of the mappings from the specified map to this map (optional operation).

These mappings will replace any mappings that this map had for any of the keys currently in the specified map.

Package [java.util](#)  
Interface [Map](#)

**Public Method remove**

```
abstract Object remove (Object o)
```

**Description:**

Removes the mapping for this key from this map if present (optional operation).

Package [java.util](#)  
Interface [Map](#)

**Public Method size**

```
abstract int size ()
```

**Description:**

Returns the number of key-value mappings in this map. If the map contains more than *Integer.MAX\_VALUE* elements, returns *Integer.MAX\_VALUE*.

Package [java.util](#)  
Interface [Map](#)

**Public Method values**

```
abstract Collection values ()
```

**Description:**

Returns a collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress, the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the *Iterator.remove*, *Collection.remove*, *removeAll*, *retainAll* and *clear* operations. It does not support the *add* or *addAll* operations.

## Package [java.util](#)

### Interface [Map.Entry](#)

A map entry (key-value pair). The `Map.entrySet` method returns a collection-view of the map, whose elements are of this class. The only way to obtain a reference to a map entry is from the iterator of this collection-view. These `Map.Entry` objects are valid only for the duration of the iteration. More formally, the behavior of a map entry is undefined if the backing map has been modified after the entry was returned by the iterator, except through the iterator's own `remove` operation, or through the `setValue` operation on a map entry returned by the iterator.

#### Public Methods

- ☞ [equals](#)
- ☞ [getKey](#)
- ☞ [getValue](#)
- ☞ [hashCode](#)
- ☞ [setValue](#)

## Package [java.util](#)

### Interface [Map.Entry](#)

#### Public Method `equals`

```
abstract boolean equals(Object o)
```

#### Overrides:

- ☞ [equals](#) in class [Object](#)

#### Description:

Compares the specified object with this entry for equality. Returns `true` if the given object is also a map entry and the two entries represent the same mapping. More formally, two entries `e1` and `e2` represent the same mapping if

```
(e1.getKey() == null ?
e2.getKey() == null : e1.getKey().equals(e2.getKey())) &&
(e1.getValue() == null ?
e2.getValue() == null : e1.getValue().equals(e2.getValue()))
```

This ensures that the `equals` method works properly across different implementations of the `Map.Entry` interface.

## Package [java.util](#)

### Interface [Map.Entry](#)

#### Public Method `getKey`

```
abstract Object getKey()
```

#### Description:

Returns the key corresponding to this entry.

Package [java.util](#)

## Interface [Map.Entry](#)

### Public Method `getValue`

```
abstract Object getValue()
```

#### Description:

Returns the value corresponding to this entry. If the mapping has been removed from the backing map (by the iterator's `remove` operation), the results of this call are undefined.


Package [java.util](#)

## Interface [Map.Entry](#)

### Public Method `hashCode`

```
abstract int hashCode()
```

#### Overrides:

 [hashCode](#) in class [Object](#)

#### Description:

Returns the hash code value for this map entry. The hash code of a map entry `e` is defined to be:

$$(e.getKey() == null ? 0 : e.getKey().hashCode()) ^ (e.getValue() == null ? 0 : e.getValue().hashCode())$$

This ensures that `e1.equals(e2)` implies that `e1.hashCode() == e2.hashCode()` for any two Entries `e1` and `e2`, as required by the general contract of `Object.hashCode`.

Package [java.util](#)

## Interface [Map.Entry](#)

### Public Method `setValue`

```
abstract Object setValue(Object value)
```

#### Description:

Replaces the value corresponding to this entry with the specified value (optional operation).

Writes through to the map. The behavior of this call is undefined if the mapping has already been removed from the map (by the iterator's `remove` operation).

Package [java.util](#)

## Class `MissingResourceException`

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

This exception signals that a resource is missing

## Public Constructors

☞ [MissingResourceException](#)

## Public Methods

☞ [getClassName](#)  
☞ [getKey](#)

## Methods inherited from [java.lang.Throwable](#)

☞ [getMessage](#)  
☞ [getLocalizedMessage](#)  
☞ [toString](#)  
☞ [fillInStackTrace](#)  
☞ [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

☞ [equals](#)  
☞ [toString](#)  
☞ [wait](#)  
☞ [hashCode](#)  
☞ [notify](#)  
☞ [notifyAll](#)

Package [java.util](#)

## Class [MissingResourceException](#)

### Public Constructor [MissingResourceException](#)

```
MissingResourceException(String s, String className, String key)
```

#### Description:

Constructs a *MissingResourceException* with the specified information. A detail message is a *String* that describes this particular exception.

Package [java.util](#)

## Class [MissingResourceException](#)

### Public Method [getClassName](#)

```
String getClassName()
```

#### Description:

Returns the class name passed by the constructor.

Package [java.util](#)

## Class [MissingResourceException](#)

### Public Method [getKey](#)

```
String getKey()
```

**Description:**

Returns the key value passed by the constructor.

Package [java.util](#)

**Class NoSuchElementException**

extends [RuntimeException](#) → [Exception](#) → [Throwable](#) → [Object](#)

Thrown by the *nextElement* method of an *Enumeration* to indicate that there are no more elements in the enumeration.

**Public Constructors**

☞ [NoSuchElementException](#)

**Methods inherited from [java.lang.Throwable](#)**

- ☞ [getMessage](#)
- ☞ [getLocalizedMessage](#)
- ☞ [toString](#)
- ☞ [fillInStackTrace](#)
- ☞ [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- ☞ [equals](#)
- ☞ [toString](#)
- ☞ [wait](#)
- ☞ [hashCode](#)
- ☞ [notify](#)
- ☞ [notifyAll](#)

Package [java.util](#)

**Class [NoSuchElementException](#)****Public Constructor NoSuchElementException**

```
NoSuchElementException()
```

```
NoSuchElementException(String s)
```

**Description:****constructor NoSuchElementException():**

Constructs a *NoSuchElementException* with *null* as its error message string.

**constructor NoSuchElementException(String s):**

Constructs a *NoSuchElementException*, saving a reference to the error message for later retrieval by the *getMessage* method.

Package [java.util](#)

**Class NotImplemented**

extends [Error](#) → [Throwable](#) → [Object](#)

Thrown when a request for functionality, which has not be impemented, has been made.

### Public Constructors

- ☞ [NotImplemented](#)

### Methods inherited from [java.lang.Throwable](#)

- ☞ [getMessage](#)
- ☞ [getLocalizedMessage](#)
- ☞ [toString](#)
- ☞ [fillInStackTrace](#)
- ☞ [getStackTrace](#)

### Methods inherited from [java.lang.Object](#)

- ☞ [equals](#)
- ☞ [toString](#)
- ☞ [wait](#)
- ☞ [hashCode](#)
- ☞ [notify](#)
- ☞ [notifyAll](#)

### Package [java.util](#)

## Class [NotImplemented](#)

### Public Constructor [NotImplemented](#)

```
NotImplemented()
```

```
NotImplemented(String s)
```

#### Description:

#### **constructor [NotImplemented](#)():**

Construct a *NotImplemented* with no detail message.

#### **constructor [NotImplemented](#)(String s):**

Construct a [NotImplemented](#) with a detailed message for later retrieval.

### Package [java.util](#)

## Class [Random](#)

extends [Object](#)

implements [Serializable](#)

This class is used to generate a random number. The class uses a 48-bit seed, which is modified using a linear congruential formula. If two instances of [Random](#) are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers.

### Public Constructors

- ☞ [Random](#)

## Public Methods

- ▣ [nextBoolean](#)
- ▣ [nextBytes](#)
- ▣ [nextFloat](#)
- ▣ [nextGaussian](#)
- ▣ [nextInt](#)
- ▣ [setSeed](#)

## Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
- ▣ [toString](#)
- ▣ [wait](#)
- ▣ [hashCode](#)
- ▣ [notify](#)
- ▣ [notifyAll](#)

Package [java.util](#)

## Class [Random](#)

### Public Constructor Random

```
Random ()
```

```
Random (int seed)
```

#### Description:

##### constructor Random()

Constructs a new random number generator. It's seed is initialized to a value based on the current time.

##### constructor Random(int seed)

Constructs a new random number generator using a single *int* seed.

Package [java.util](#)

## Class [Random](#)

### Public Method nextBoolean

```
boolean nextBoolean ()
```

#### Description:

Returns the next random, uniformly distributed *boolean* value from this random nubmer generator's sequence. The general contract of *nextBoolean* is that one *boolean* value is randomly generated and returned.

#### Example

```
Random rand = new Random();  
Logger.log("random boolean: " + rand.nextBoolean());
```



The *nextBoolean* method returns the next random boolean value.

Package [java.util](#)  
Class [Random](#)

**Public Method nextBytes**

```
void nextBytes(byte[] bytes)
```

**Description:**

The *nextByte* method generates random bytes and places them into the *bytes* array. The number of random bytes produced is equal to the length of the *byte* array.

**Example**

```
byte randByte[] = new byte[10];
rand.nextBytes(randByte);
for (int i = 0; i < 10; i++)
    Logger.log("random value: " + randByte[i]);
```

The *nextByte* method returns the *bytes* array with random values.

Package [java.util](#)  
Class [Random](#)

**Public Method nextFloat**

```
float nextFloat()
```

**Description:**

This method generates and returns the next random *float* value between *0.0* and *1.0*. The returned values ranging from the values *0.0f*, which is inclusiv, and *1.0f* which is exclusive.

**Example**

```
Random rand = new Random();
Logger.log("random float: " + rand.nextFloat());
```

The *nextFloat* method returns the next random *float* value.

Package [java.util](#)  
Class [Random](#)

**Public Method nextGaussian**

```
float nextGaussian()
```

**Description:**

This method returns the next random, Gaussian distributed *float* value.

## Example

```
Random rand = new Random();
Logger.log("random Gaussian: " + rand.nextGaussian());
```

The *nextGaussian* method returns the next random value.

## Package [java.util](#) Class [Random](#)

### Public Method nextInt

```
int nextInt()
int nextInt(int n)
```

#### Throws:

☞ [IllegalArgumentException](#)

#### Description:

##### **nextInt()** method:

This method returns the next random, uniformly distributed *int* value from this random number generator's sequence.

##### **nextInt(int n)** method:

This method returns a random, uniformly distributed *int* value between *1*, which is inclusive, and the specified value, which is exclusive.

## Example

```
Random rand = new Random();
Logger.log("random integer: " + rand.nextInt());
```

The *nextInt* method returns the next random *int* value.

## Package [java.util](#) Class [Random](#)

### Public Method setSeed

```
void setSeed(int seed)
```

#### Description:

Sets the *seed* value of the random generator using the *int* parameter.

## Example

```
Random rand = new Random();
rand.setSeed(23);
```

The `setSeed` method sets up a new `seed` value.

## Package [java.util](#)

### Interface **Set**

implements [Collection](#)

A collection that contains no duplicate elements. More formally, sets contain no pair of elements  $e1$  and  $e2$  such that  $e1.equals(e2)$ , and at most one null element. As implied by its name, this interface models the mathematical *set* abstraction.

The *Set* interface places additional stipulations, beyond those inherited from the *Collection* interface, on the contracts of all constructors and on the contracts of the `add`, `equals` and `hashCode` methods. Declarations for other inherited methods are also included here for convenience. The specifications accompanying these declarations have been tailored to the *Set* interface, but they do not contain any additional stipulations.

The additional stipulation on constructors is, that all constructors must create a set that contains no duplicate elements as defined above.

Use the concrete implementations [HashSet](#) and [TreeSet](#) (they differ only in performance).

#### Public Methods

- ▣ [add](#)
- ▣ [addAll](#)
- ▣ [clear](#)
- ▣ [contains](#)
- ▣ [containsAll](#)
- ▣ [equals](#)
- ▣ [hashCode](#)
- ▣ [isEmpty](#)
- ▣ [iterator](#)
- ▣ [remove](#)
- ▣ [removeAll](#)
- ▣ [retainAll](#)
- ▣ [size](#)
- ▣ [toArray](#)

## Package [java.util](#)

### Interface **Set**

#### Public Method **add**

```
abstract boolean add(Object o)
```

#### Description:

Adds the specified element to this set if it is not already present (optional operation).

More formally, adds the specified element, *o*, to this set if this set contains no element *e* such that  $(o == null ? e == null : o.equals(e))$ . If this set already contains the specified element, the call leaves this set unchanged and returns *false*. In combination with the restriction on constructors, this ensures that sets never contain duplicate elements.

The stipulation above does not imply that sets must accept all elements; sets may refuse to add any particular element, including null, and throwing an exception, as described in the specification for *Collection.add*. Individual set implementations should clearly document any restrictions on the the elements that they may contain.

Note that basic types cannot be used with this method because they are not of type object. If you want to add e.g. a byte value use the Byte class, see the HashSet example below.

### Example

```
Set MyHashSet = new HashSet(); // HashSet implements Set
MyHashSet.add(new Byte((byte) 0xAA)); // create a Byte-object
```

Package [java.util](#)

Interface [Set](#)

### Public Method addAll

```
abstract boolean addAll(Collection c)
```

#### Description:

Adds all of the elements in the specified collection to this set if they're not already present (optional operation).

If the specified collection is also a set, the *addAll* operation effectively modifies this set so that its value is the union of the two sets. The behavior of this operation is unspecified if the specified collection is modified while the operation is in progress.

Package [java.util](#)

Interface [Set](#)

### Public Method clear

```
abstract void clear()
```

#### Description:

Removes all of the elements from this set (optional operation).

This set will be empty after this call returns unless it throws an exception.

Package [java.util](#)

Interface [Set](#)

### Public Method contains

```
abstract boolean contains(Object o)
```

**Description:**

Returns *true* if this set contains the specified element. More formally, returns *true* if and only if this set contains an element *e* such that  $(o==null ? e==null : o.equals(e))$ .

**Package [java.util](#)  
Interface [Set](#)****Public Method containsAll**

```
abstract boolean containsAll(Collection c)
```

**Description:**

Returns *true* if this set contains all of the elements of the specified collection. If the specified collection is also a set, this method returns *true* if it is a subset of this set.

**Package [java.util](#)  
Interface [Set](#)****Public Method equals**

```
abstract boolean equals(Object o)
```

**Overrides:**

☞ [equals](#) in class [Object](#)

**Description:**

Compares the specified object with this set for equality. Returns *true* if the specified object is also a set, the two sets have the same size, and every member of the specified set is contained in this set (or equivalently, every member of this set is contained in the specified set). This definition ensures that the equals method works properly across different implementations of the set interface.

**Package [java.util](#)  
Interface [Set](#)****Public Method hashCode**

```
abstract int hashCode()
```

**Overrides:**

☞ [hashCode](#) in class [Object](#)

**Description:**

Returns the hash code value for this set. The hash code of a set is defined to be the sum of the hash codes of the elements in the set, where the hashcode of a *null* element is defined to be zero. This ensures that  $s1.equals(s2)$  implies that  $s1.hashCode() == s2.hashCode()$  for any two sets *s1* and *s2*, as required by the general contract of the *Object.hashCode* method.

**Package [java.util](#)  
Interface [Set](#)****Public Method isEmpty**

```
abstract boolean isEmpty()
```

**Description:**

Returns *true* if this set contains no elements.

**Package [java.util](#)  
Interface [Set](#)****Public Method iterator**

```
abstract Iterator iterator()
```

**Description:**

Returns an iterator over the elements in this set. The elements are returned in no particular order unless this set is an instance of some class that provides a guarantee.

**Package [java.util](#)  
Interface [Set](#)****Public Method remove**

```
abstract boolean remove(Object o)
```

**Description:**

Removes the specified element from this set if it is present (optional operation).

More formally, removes an element *e* such that  $(o == null ? e == null : o.equals(e))$ , if the set contains such an element. Returns *true* if the set contained the specified element or equivalently, if the set changed as a result of the call. The set will not contain the specified element once the call returns.

**Package [java.util](#)  
Interface [Set](#)****Public Method removeAll**

```
abstract boolean removeAll(Collection c)
```

**Description:**

Removes from this set all of its elements that are contained in the specified collection (optional operation).

If the specified collection is also a set, this operation effectively modifies this set so that its value is the asymmetric set difference of the two sets.

## Package [java.util](#) Interface [Set](#)

### Public Method retainAll

```
abstract boolean retainAll(Collection c)
```

#### Description:

Retains only the elements in this set that are contained in the specified collection (optional operation).

In other words, removes from this set all of its elements that are not contained in the specified collection. If the specified collection is also a set, this operation effectively modifies this set so that its value is the intersection of the two sets.

## Package [java.util](#) Interface [Set](#)

### Public Method size

```
abstract int size()
```

#### Description:

Returns the number of elements in this set (its cardinality). If this set contains more *Integer.MAX\_VALUE* elements, returns *Integer.MAX\_VALUE*.

## Package [java.util](#) Interface [Set](#)

### Public Method toArray

```
abstract Object[] toArray()
```

#### Description:

Returns an array containing all of the elements in this set whose runtime type is that of the specified array. Obeys the general contract of the *Collection.toArray(Object[])* method.

## Package [java.util](#)

## Interface SortedMap

implements [Map](#)

A map that further guarantees that it will be in ascending key order, sorted according to the natural ordering of its keys (see the *Comparable* interface), or by a comparator provided at sorted map creation time. This order is reflected when iterating over the sorted map's collection views (returned by the *entrySet*, *keySet* and *values* methods). Several additional operations are provided to take advantage of the ordering. (This interface is the map analogue of the *SortedSet* interface.)

All keys inserted into a sorted map must implement the *Comparable* interface (or be accepted by the specified comparator). Furthermore, all such keys must be mutually comparable:

*k1.compareTo(k2)* (or *comparator.compare(k1, k2)*) must not throw a

ClassCastException for any elements *k1* and *k2* in the sorted map. Attempts to violate this restriction will cause the offending method or constructor invocation to throw a *ClassCastException*.

Note that the ordering maintained by a sorted map (whether or not an explicit comparator is provided) must be consistent with equals if the sorted map is to correctly implement the *Map* interface. See the *Comparable* interface or *Comparator* interface for a precise definition of consistent with equals. This is so because the *Map* interface is defined in terms of the *equals* operation, but a sorted map performs all key comparisons using its *compareTo* (or *compare*) method, so two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal. The behavior of a tree map is well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the *Map* interface.

All general-purpose sorted map implementation classes should provide four "standard" constructors:

- ☐ A void (no arguments) constructor, which creates an empty sorted map sorted according to the natural order of its keys.
- ☐ A constructor with a single argument of type *Comparator*, which creates an empty sorted map sorted according to the specified comparator.
- ☐ A constructor with a single argument of type *Map*, which creates a new map with the same key-value mappings as its argument, sorted according to the keys' natural ordering.
- ☐ A constructor with a single argument of type sorted map, which creates a new sorted map with the same key-value mappings and the same ordering as the input sorted map.

There is no way to enforce this recommendation (as interfaces cannot contain constructors) but the SDK implementation (TreeMap) complies.

## Public Methods

- ☐ [comparator](#)
- ☐ [firstKey](#)
- ☐ [headMap](#)
- ☐ [lastKey](#)
- ☐ [subMap](#)
- ☐ [tailMap](#)

## Package [java.util](#)

### Interface [SortedMap](#)

#### Public Method [comparator](#)

```
abstract Comparator comparator()
```

#### Description:

Returns a view of the portion of this sorted map whose keys range from *fromKey*, inclusive, to *toKey*, exclusive. If *fromKey* and *toKey* are equal, the returned sorted map is empty. The returned sorted map is backed by this sorted map, so changes in the returned sorted map are reflected in this sorted map, and vice-versa. The returned Map supports all optional map operations that this sorted map supports.

The map returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a key outside the specified range.

This method always returns a half-open range which includes its low endpoint but not its high endpoint. If you need a closed range (which includes both endpoints), and the key type allows for calculation of the successor a given key, merely request the subrange from `lowEndpoint` to `successor(highEndpoint)`



Package [java.util](#)

## Interface [SortedMap](#)

### Public Method [firstKey](#)

```
abstract Object firstKey()
```

#### Description:

Returns the first and lowest key currently in this sorted map.

Package [java.util](#)

## Interface [SortedMap](#)

### Public Method [headMap](#)

```
abstract SortedMap headMap(Object toKey)
```

#### Description:

Returns a view of the portion of this sorted map whose keys are strictly less than *toKey*. The returned sorted map is backed by this sorted map, so changes in the returned sorted map are reflected in this sorted map, and vice-versa. The returned map supports all optional map operations that this sorted map supports. The map returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a key outside the specified range.

Package [java.util](#)

## Interface [SortedMap](#)

### Public Method [lastKey](#)

```
abstract Object lastKey()
```

#### Description:

Returns the last and highest key currently in this sorted map.

Package [java.util](#)

## Interface [SortedMap](#)

### Public Method [subMap](#)

```
abstract SortedMap subMap(Object fromKey, Object toKey)
```

#### Description:

Returns a view of the portion of this sorted map whose keys range from *fromKey*, inclusive, to *toKey*, exclusive. If *fromKey* and *toKey* are equal, the returned sorted map is empty. The returned sorted map is backed by this sorted map, so changes in the returned sorted map are reflected in this sorted map, and vice-versa. The returned Map supports all optional map operations that this sorted map supports.

The map returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a key outside the specified range.

This method always returns a half-open range (which includes its low endpoint but not its high endpoint). If you need a closed range (which includes both endpoints), and the key type allows for calculation of the successor a given key, merely request the subrange from *lowEndpoint* to *successor(highEndpoint)*.

Package [java.util](#)

## Interface [SortedMap](#)

### Public Method [tailMap](#)

```
abstract SortedMap tailMap(Object fromKey)
```

#### Description:

Returns a view of the portion of this sorted map whose keys are strictly less than *toKey*. The returned sorted map is backed by this sorted map, so changes in the returned sorted map are reflected in this sorted map, and vice-versa. The returned map supports all optional map operations that this sorted map supports.

The map returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a key outside the specified range.

This method always returns a view that does not contain its (high) endpoint. If you need a view that does contain this endpoint, and the key type allows for calculation of the successor a given key, merely request a *headMap* bounded by *successor(highEndpoint)*.

Package [java.util](#)

## Interface [SortedSet](#)

implements [Set](#)

A set that further guarantees that its iterator will traverse the set in ascending element order, sorted according to the natural ordering of its elements (see *Comparable*), or by a *Comparator* provided at sorted set creation time. Several additional operations are provided to take advantage of the ordering. This interface is the set analogue of *SortedMap*.

All elements inserted into an sorted set must implement the *Comparable* interface or be accepted by the specified *Comparator*. Furthermore, all such elements must be mutually comparable: *e1.compareTo(e2)* (or *comparator.compare(e1, e2)*) must not throw a *ClassCastException* for any elements *e1* and *e2* in the sorted set. Attempts to violate this restriction will cause the offending method or constructor invocation to throw a *ClassCastException*.

Note that the ordering maintained by a sorted set (whether or not an explicit comparator is provided) must be consistent with equals if the sorted set is to correctly implement the *Set* interface. See the *Comparable* interface or *Comparator* interface for a precise definition of consistent with equals. This is so because the *Set* interface is defined in terms of the equals operation, but a sorted set performs all element comparisons using its *compareTo* (or *compare*) method, so two elements that are deemed equal by this method are, from the standpoint of the sorted set, equal. The behavior of a sorted set is well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the *Set* interface.

All general-purpose sorted set implementation classes should provide four "standard" constructors:  
1) A void (no arguments) constructor, which creates an empty sorted set sorted according to the natural order of its elements.

- 2) A constructor with a single argument of type *Comparator*, which creates an empty sorted set sorted according to the specified comparator.
- 3) A constructor with a single argument of type *Collection*, which creates a new sorted set with the same elements as its argument, sorted according to the elements' natural ordering.
- 4) A constructor with a single argument of type *SortedSet*, which creates a new sorted set with the same elements and the same ordering as the input sorted set.

There is no way to enforce this recommendation (as interfaces cannot contain constructors) but the SDK implementation (the *TreeSet* class) complies.

## Public Methods

- [comparator](#)
- [first](#)
- [headSet](#)
- [last](#)
- [subSet](#)
- [tailSet](#)

## Package [java.util](#)

### Interface [SortedSet](#)

#### Public Method [comparator](#)

```
abstract Comparator comparator()
```

#### Description:

Returns a view of the portion of this sorted set whose elements range from *fromElement*, inclusive, to *toElement*, exclusive. If *fromElement* and *toElement* are equal, the returned sorted set is empty. The returned sorted set is backed by this sorted set, so changes in the returned sorted set are reflected in this sorted set, and vice-versa. The returned sorted set supports all optional set operations that this sorted set supports.

The sorted set returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a element outside the specified range.

This method always returns a half-open range which includes its low endpoint but not its high endpoint. If you need a closed range which includes both endpoints, and the element type allows for calculation of the successor a given value, merely request the subrange from lowEndpoint to *successor(highEndpoint)*.

## Package [java.util](#)

### Interface [SortedSet](#)

#### Public Method [first](#)

```
abstract Object first()
```

#### Description:

Returns the last and highest element currently in this sorted set.

Package [java.util](#)

## Interface [SortedSet](#)

### Public Method headSet

```
abstract SortedSet headSet (Object toElement)
```

#### Description:

Returns a view of the portion of this sorted set whose elements are strictly less than *toElement*. The returned sorted set is backed by this sorted set, so changes in the returned sorted set are reflected in this sorted set, and vice-versa. The returned sorted set supports all optional set operations.

The sorted set returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a element outside the specified range.

This method always returns a view that does not contain its (high) endpoint. If you need a view that does contain this endpoint, and the element type allows for calculation of the successor a given value, merely request a *headSet* bounded by *successor (highEndpoint)*.

Package [java.util](#)

## Interface [SortedSet](#)

### Public Method last

```
abstract Object last ()
```

#### Description:

Returns the last and highest element currently in this sorted set.

Package [java.util](#)

## Interface [SortedSet](#)

### Public Method subSet

```
abstract SortedSet subSet (Object fromElement, Object toElement)
```

#### Description:

Returns a view of the portion of this sorted set whose elements range from *fromElement*, inclusive, to *toElement*, exclusive. If *fromElement* and *toElement* are equal, the returned sorted set is empty. The returned sorted set is backed by this sorted set, so changes in the returned sorted set are reflected in this sorted set, and vice-versa. The returned sorted set supports all optional set operations that this sorted set supports.

The sorted set returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a element outside the specified range.

This method always returns a half-open range which includes its low endpoint but not its high endpoint. If you need a closed range which includes both endpoints, and the element type allows for calculation of the successor a given value, merely request the subrange from *lowEndpoint* to *successor (highEndpoint)*.

**Package [java.util](#)****Interface [SortedSet](#)****Public Method [tailSet](#)**

```
abstract SortedSet tailSet(Object fromElement)
```

**Description:**

Returns a view of the portion of this sorted set whose elements are greater than or equal to *fromElement*. The returned sorted set is backed by this sorted set, so changes in the returned sorted set are reflected in this sorted set, and vice-versa. The returned sorted set supports all optional set operations.

The sorted set returned by this method will throw an *IllegalArgumentException* if the user attempts to insert a element outside the specified range.

This method always returns a view that contains its (low) endpoint. If you need a view that does not contain this endpoint, and the element type allows for calculation of the successor a given value, merely request a tailSet bounded by *successor(lowEndpoint)*.

**Package [java.util](#)****Class [Stack](#)**

extends [Vector](#) → [AbstractList](#) → [AbstractCollection](#) → [Object](#)  
implements [List](#), [Cloneable](#), [Serializable](#), [Collection](#)

The *Stack* class represents a last-in-first-out object. It extends the *Vector* class with five operations that allow a vektor to be threaded as a stack. The usual *push* and *pop* operations are available, as well as a method to *peek* at the top item on the stack. Furthermore, a method to test for whether the stack is *empty* or not, and a method to *search* the stack for an item and discover how far it is from the top.

**Public Constructors**

☞ [Stack](#)

**Public Methods**

☞ [empty](#)  
☞ [peek](#)  
☞ [pop](#)  
☞ [push](#)  
☞ [search](#)

**Methods inherited from [java.util.Vector](#)**

☞ [copyInto](#)  
☞ [trimToSize](#)  
☞ [ensureCapacity](#)  
☞ [setSize](#)  
☞ [capacity](#)  
☞ [size](#)  
☞ [isEmpty](#)  
☞ [indexOf](#)  
☞ [contains](#)  
☞ [lastIndexOf](#)  
☞ [elementAt](#)

- ▣ [firstElement](#)
- ▣ [lastElement](#)
- ▣ [setElementAt](#)
- ▣ [set](#)
- ▣ [removeElementAt](#)
- ▣ [insertElementAt](#)
- ▣ [addElement](#)
- ▣ [removeElement](#)
- ▣ [removeAllElements](#)
- ▣ [clone](#)
- ▣ [toArray](#)
- ▣ [get](#)
- ▣ [remove](#)
- ▣ [add](#)
- ▣ [remove](#)
- ▣ [clear](#)
- ▣ [containsAll](#)
- ▣ [addAll](#)
- ▣ [removeAll](#)
- ▣ [retainAll](#)
- ▣ [addAll](#)
- ▣ [equals](#)
- ▣ [hashCode](#)
- ▣ [toString](#)
- ▣ [elements](#)
- ▣ [subList](#)

#### Methods inherited from [java.util.AbstractList](#)

- ▣ [get](#)
- ▣ [add](#)
- ▣ [addAll](#)
- ▣ [clear](#)
- ▣ [equals](#)
- ▣ [hashCode](#)
- ▣ [indexOf](#)
- ▣ [iterator](#)
- ▣ [lastIndexOf](#)
- ▣ [listIterator](#)
- ▣ [remove](#)
- ▣ [set](#)
- ▣ [subList](#)

#### Methods inherited from [java.util.AbstractCollection](#)

- ▣ [iterator](#)
- ▣ [size](#)
- ▣ [add](#)
- ▣ [addAll](#)
- ▣ [clear](#)
- ▣ [contains](#)
- ▣ [containsAll](#)
- ▣ [isEmpty](#)
- ▣ [remove](#)
- ▣ [removeAll](#)
- ▣ [retainAll](#)
- ▣ [toArray](#)
- ▣ [toString](#)

#### Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)

- ☰ [toString](#)
- ☰ [wait](#)
- ☰ [hashCode](#)
- ☰ [notify](#)
- ☰ [notifyAll](#)

## Package [java.util](#)

### Class [Stack](#)

#### Public Constructor Stack

```
Stack ()
```

#### Description:

Constructs a new and empty stack.

## Package [java.util](#)

### Class [Stack](#)

#### Public Method empty

```
boolean empty ()
```

#### Description:

Verifies whether the stack contains no elements.

#### Example

```
Stack stack = new Stack();
Logger.log("is the stack empty: " + stack.empty());
```

The example above returns a boolean value and writes it to the logger.

## Package [java.util](#)

### Class [Stack](#)

#### Public Method peek

```
Object peek ()
```

#### Throws:

- ☰ [EmptyStackException](#)

#### Description:

This method returns the element from the top of the stack without removing it from the stack.

#### Example

```
Stack stack = new Stack();
stack.push("s");
stack.push("t");
```

```
stack.push("a");
stack.push("c");
stack.push("k");
Logger.log("top element on the stack: " + stack.peek());
```

The example above returns the element from the top of the stack.

Package [java.util](#)

Class [Stack](#)

**Public Method pop**

Object **pop**()

**Description:**

This method returns and removes the element from the top of the stack.

**Example**

```
Stack stack = new Stack();
stack.push("s");
stack.push("t");
stack.push("a");
stack.push("c");
stack.push("k");
Logger.log("remove the top element from the stack: " + stack.pop());
Logger.log("top element on the stack: " + stack.peek());
```

The example above removes and returns the element from the top of the stack. Afterwards, the element from the top of the stack is written to the logger.

Package [java.util](#)

Class [Stack](#)

**Public Method push**

Object **push**(Object item)

**Description:**

This method pushes the *item* onto the stack.

**Example**

```
Stack stack = new Stack();
stack.push("s");
stack.push("t");
stack.push("a");
stack.push("c");
stack.push("k");
Logger.log("top element on the stack: " + stack.peek());
```



The example above pushes several elements on the stack. The element from the top of the stack is written to the logger.

Package [java.util](#)

## Class [Stack](#)

### Public Method [search](#)

```
int search(Object o)
```

#### Description:

Returns the 1-based position where an element is on this stack. The distance from the top of the stack, if the element *o* occurs as an item in this stack. The element on the top of the stack is considered to be at the distance 1. This method returns -1 if the stack doesn't contain the specified element.

#### Example

```
Stack stack = new Stack();
stack.push("s");
stack.push("t");
stack.push("a");
stack.push("c");
stack.push("k");
Logger.log("position from the 'a' element: " + stack.search("a"));
```

The example above returns the position from the specified element within the stack.

Package [java.util](#)

## Class [StringTokenizer](#)

extends [Object](#)

implements [Enumeration](#)

This class allows an application to break a string into tokens. The *stringTokenizer* methods do not distinguish among identifiers, numbers, and quoted strings, nor do they recognize and skip comments. The set of delimiters may be specified either at creation time or on a per-token basis. The instance behaves in two ways depending on the flag (*true* or *false*) during object creation:

- ☞ If the flag is set to *false*, the delimiter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.
- ☞ If the flag is set to *true*, the delimiter characters are themselves considered to be tokens. A token is thus either one delimiter character, or a maximal sequence of consecutive characters that are not delimiters.

### Public Constructors

- ☞ [StringTokenizer](#)

### Public Methods

- ☞ [countTokens](#)
- ☞ [hasMoreElements](#)
- ☞ [hasMoreTokens](#)
- ☞ [nextElement](#)
- ☞ [nextToken](#)

## Methods inherited from [java.lang.Object](#)

- ☐ [equals](#)
- ☐ [toString](#)
- ☐ [wait](#)
- ☐ [hashCode](#)
- ☐ [notify](#)
- ☐ [notifyAll](#)

## Package [java.util](#)

### Class [StringTokenizer](#)

#### Public Constructor [StringTokenizer](#)

```
StringTokenizer(String str)

StringTokenizer(String str, String delim)

StringTokenizer(String input, String delims, boolean retDelim)
```

#### Description:

##### constructor [StringTokenizer\(String str\)](#)

Constructs an instance of a *StringTokenizer* for the specified string *str*. The tokenizer uses the default delimiter set, which are the space character, the tab character, the newline character, the carriage-return character, and the form-feed character. Delimiter characters themselves will not be treated as tokens.

##### constructor [StringTokenizer\(String str, String delims\)](#)

Constructs an instance of a *StringTokenizer* for the specified string *str*. The characters in the *delims* argument are the delimiters for separating tokens. Delimiter characters themselves will not be treated as tokens.

##### constructor [StringTokenizer\(String input, String delims, boolean retDelim\)](#)

Constructs an instance of a *StringTokenizer* for the specified string *str*. The characters in the *delims* argument are the delimiters for separating tokens. If the returnDelim flag is *true*, then the delimiter characters are also returned as tokens. Each delimiter is returned as a string of length one. If the flag is *false*, the delimiter characters are skipped and only serve as separators between tokens.

## Package [java.util](#)

### Class [StringTokenizer](#)

#### Public Method [countTokens](#)

```
int countTokens ()
```

#### Description:

The *countTokens* method calculates the number of times that this tokenizer's *nextToken* method can be called before it generates an exception. The current position is not advanced.

## Example

```
StringTokenizer strTok = new StringTokenizer("Hello World\rThis is a\
t\ttest string!\n");
Logger.log("count of tokens: " + strTok.countTokens());
```

The example above iterates as long as tokens available from the specified tokenizer string.

Package [java.util](#)

## Class [StringTokenizer](#)

### Public Method hasMoreElements

```
boolean hasMoreElements()
```

#### Description:

Returns the same value as the *hasMoreTokens* method. It exists so that this class can implement the *Enumeration* interface.

## Example

```
StringTokenizer strTok = new StringTokenizer("Hello World\rThis is a\
t\ttest string!\n");
while (strTok.hasMoreElements())
    Logger.log(strTok.nextToken());
```

The different elements are written to the logger as long as further tokens are available.

Package [java.util](#)

## Class [StringTokenizer](#)

### Public Method hasMoreTokens

```
boolean hasMoreTokens()
```

#### Description:

Checks if there are more tokens available from this tokenizer's string. If this method returns *true*, then a subsequent call to *nextToken* with no argument will successfully return a token.

## Example

```
StringTokenizer strTok = new StringTokenizer("Hello World\rThis is a\
t\ttest string!\n");
while (strTok.hasMoreTokens())
    Logger.log(strTok.nextToken());
```

The different elements are written to the logger as long as further tokens are available.

**Package [java.util](#)****Class [StringTokenizer](#)****Public Method [nextElement](#)**

```
Object nextElement()
```

**Description:**

Returns the same value as the *nextToken* method, except that it's declared return value is *Object* rather than *String*. It exists so that this class can implement the *Enumeration* interface.

**Example**

```
StringTokenizer strTok = new StringTokenizer("Hello World\rThis is a \t\ttest string!\n");
while (strTok.hasMoreTokens())
    Logger.log(strTok.nextElement());
```

The different elements are written to the logger as long as further tokens are available.

**Package [java.util](#)****Class [StringTokenizer](#)****Public Method [nextToken](#)**

```
String nextToken()
```

**Throws:**

⇒ [NoSuchElementException](#)

```
String nextToken(String delims)
```

**Description:****nextToken() method:**

Returns the next token from this string tokenizer.

**nextToken(String delims) method:**

Returns the next token in this string tokenizer's string. First, the set of characters considered to be delimiters by this *StringTokenizer* object is changed to be the characters in the sting *delims*. Then the next token in the string after the current position is returned. The current position is advanced beyond the recognized token. The new delimiter set remains the default after this call.

**Example**

```
StringTokenizer strTok = new StringTokenizer("Hello World\rThis is a \t\ttest string!\n");
while (strTok.hasMoreTokens())
    Logger.log(strTok.nextToken());
```

The different elements are written to the logger as long as further tokens are available.

## Package [java.util](#)

### Class [TooManyListenersException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

The *TooManyListenersException* exception is used as part of the Java event model to annotate and implement an unicast special case for a multicast event source.

#### Public Constructors

☞ [TooManyListenersException](#)

#### Methods inherited from [java.lang.Throwable](#)

- ☞ [getMessage](#)
- ☞ [getLocalizedMessage](#)
- ☞ [toString](#)
- ☞ [fillInStackTrace](#)
- ☞ [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- ☞ [equals](#)
- ☞ [toString](#)
- ☞ [wait](#)
- ☞ [hashCode](#)
- ☞ [notify](#)
- ☞ [notifyAll](#)

## Package [java.util](#)

### Class [TooManyListenersException](#)

#### Public Constructor [TooManyListenersException](#)

```
TooManyListenersException ()
```

```
TooManyListenersException(String detail)
```

#### Description:

##### constructor [TooManyListenersException](#)():

Constructs a *TooManyListenersException* with no detailed message.

##### constructor [TooManyListenersException](#)(String s):

Constructs a *TooManyListenersException* with a detailed message for later retrieval.

## Package [java.util](#)

### Class [TreeMap](#)

extends [AbstractMap](#) → [Object](#)

implements [SortedMap](#), [Cloneable](#), [Serializable](#), [Map](#)

Red-Black tree based implementation of the *SortedMap* interface. This class guarantees that the map will be in ascending key order, sorted according to the natural order for the key's class, or by the comparator provided at creation time, depending on which constructor is used.

The ordering maintained by a sorted map must be consistent with equals if this sorted map correctly implement the *Map* interface. The *Map* interface is defined in terms of the equals operation, but a map performs all key comparisons using its *compareTo* method. Two keys that are deemed equal by this method are, from the standpoint of the sorted map, equal.

## Public Constructors

- ▣ [TreeMap](#)

## Public Methods

- ▣ [clear](#)
- ▣ [clone](#)
- ▣ [comparator](#)
- ▣ [containsKey](#)
- ▣ [containsValue](#)
- ▣ [entrySet](#)
- ▣ [firstKey](#)
- ▣ [get](#)
- ▣ [headMap](#)
- ▣ [lastKey](#)
- ▣ [put](#)
- ▣ [putAll](#)
- ▣ [remove](#)
- ▣ [size](#)
- ▣ [subMap](#)
- ▣ [tailMap](#)

## Methods inherited from [java.util.AbstractMap](#)

- ▣ [size](#)
- ▣ [isEmpty](#)
- ▣ [containsValue](#)
- ▣ [containsKey](#)
- ▣ [get](#)
- ▣ [put](#)
- ▣ [remove](#)
- ▣ [putAll](#)
- ▣ [clear](#)
- ▣ [keySet](#)
- ▣ [values](#)
- ▣ [entrySet](#)
- ▣ [equals](#)
- ▣ [hashCode](#)
- ▣ [toString](#)
- ▣ [clone](#)

## Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
- ▣ [toString](#)
- ▣ [wait](#)
- ▣ [hashCode](#)
- ▣ [notify](#)
- ▣ [notifyAll](#)

Package [java.util](#)  
Class [TreeMap](#)

### Public Constructor TreeMap

```
TreeMap ()
```

```
TreeMap (Comparator c)
```

```
TreeMap (Map m)
```

```
TreeMap (SortedMap m)
```

#### **Description:**

##### **constructor TreeMap()**

This method constructs a new, empty, and sorted map according to the keys natural order. All keys inserted into the map must implement the *Comparable* interface. All such keys must be mutually comparable, and must not throw a *ClassCastException* for any elements in the map.

##### **constructor TreeMap(Comparator c)**

This method constructs a new, empty, and sorted map according to the given comparator. All keys inserted into the map must be mutually comparable by the given comparator, and must not throw a *ClassCastException* or any elements in the map.

##### **constructor TreeMap(Map m)**

This method constructs a new map containing the same mappings as the given map, and is sorted to the keys natural order. All keys inserted into the new map must implement the *Comparable* interface. All such keys must be mutually comparable, and must not throw a *ClassCastException* for any elements in the map.

##### **constructor TreeMap(SortedMap m)**

This method constructs a new map containing the same mappings, and is sorted according to the same ordering as the given *SortedMap*.

Package [java.util](#)  
Class [TreeMap](#)

### Public Method clear

```
void clear ()
```

#### **Overrides:**

☐ [clear](#) in class [AbstractMap](#)

#### **Description:**

This method removes all mappings from this *TreeMap*.

## Example

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
treeMap.put("2", "r");
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
Logger.log("TreeMap: " + treeMap);

treeMap.clear();
Logger.log("TreeMap: " + treeMap);
```

The example above removes all mappings after creation, and writes the result to the logger.

Package [java.util](#)

Class [TreeMap](#)

### Public Method clone

Object **clone()**

#### Overrides:

☰ [clone](#) in class [AbstractMap](#)

#### Description:

Returns a shallow copy of this *TreeMap* instance.

## Example

```
TreeMap treeMap1 = new TreeMap();
TreeMap treeMap2 = new TreeMap();
treeMap1.put("1", "T");
treeMap1.put("2", "r");
treeMap1.put("3", "e");
treeMap1.put("4", "e");
treeMap1.put("5", "M");
treeMap1.put("6", "a");
treeMap1.put("7", "p");
Logger.log("TreeMap1: " + treeMap1);

treeMap2 = (TreeMap)treeMap1.clone();
Logger.log("TreeMap2: " + treeMap2);
```

The example above makes a shallow copy of the *treeMap1*, and writes it to the logger.



Package [java.util](#)  
Class [TreeMap](#)

### Public Method comparator

```
Comparator comparator()
```

#### Description:

This method returns a comparator instance which is used to order this map, or *null* if this map uses its key's natural order.

#### Example

```
byte randByte[] = new byte[10];
rand.nextBytes(randByte);
for (int i = 0; i < 10; i++)
    Logger.log("random value: " + randByte[i]);
```

The *nextByte* method returns the *bytes* array with random values.

Package [java.util](#)  
Class [TreeMap](#)

### Public Method containsKey

```
boolean containsKey(Object key)
```

#### Overrides:

↳ [containsKey](#) in class [AbstractMap](#)

#### Description:

Returns *true* if this map contains a mapping for the specified key.

#### Example

```
TreeMap treeMap1 = new TreeMap();
treeMap1.put("1", "T");
treeMap1.put("2", "r");
treeMap1.put("3", "e");
treeMap1.put("4", "e");
treeMap1.put("5", "M");
treeMap1.put("6", "a");
treeMap1.put("7", "p");

if (treeMap.containsKey("2"))
    Logger.log("This treeMap contains the key 2");
```

The example above prints the result to the logger if the *treeMap* contains the key string *2*.

Package [java.util](#)  
Class [TreeMap](#)

### Public Method `containsValue`

```
boolean containsValue(Object value)
```

#### Overrides:

↳ [containsValue](#) in class [AbstractMap](#)

#### Description:

Returns *true* if this map maps one keys to the specified value.

#### Example

```
TreeMap treeMap1 = new TreeMap();
treeMap1.put("1", "T");
treeMap1.put("2", "r");
treeMap1.put("3", "e");
treeMap1.put("4", "e");
treeMap1.put("5", "M");
treeMap1.put("6", "a");
treeMap1.put("7", "p");

if (treeMap.containsValue("r"))
    Logger.log("This treeMap contains the value r");
```

The example above prints the result to the logger if the `treeMap` contains the value string *r*.

Package [java.util](#)  
Class [TreeMap](#)

### Public Method `entrySet`

```
Set entrySet()
```

#### Overrides:

↳ [entrySet](#) in class [AbstractMap](#)

#### Description:

Returns a set view of the mappings contained in this map. The set's iterator returns the mappings in ascending key order. Each element in the returned set is a *Map.Entry*. The set is backed by this map, so changes to this map are reflected in the set, and vice-versa.

#### Example

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
treeMap.put("2", "r");
```

```
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
Logger.log("TreeMap: " + treeMap.entrySet());
```

The example above returns a set view of the mappings contained in this map.

Package [java.util](#)

Class [TreeMap](#)

**Public Method [firstKey](#)**

Object **firstKey()**

**Throws:**

⇒ [NoSuchElementException](#)

**Description:**

Returns the first key currently in this sorted map.

**Example**

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
treeMap.put("2", "r");
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
Logger.log("TreeMap: " + treeMap.firstKey());
```

The example above returns the first key in this map.

Package [java.util](#)

Class [TreeMap](#)

**Public Method [get](#)**

Object **get**(Object key)

**Overrides:**

⇒ [get](#) in class [AbstractMap](#)

**Description:**

Returns the value to which this map maps the specified key. *null* is returned, if the map contains no mapping for this key. A return value of *null* dose not necessarily indicate that the map contains no mapping for this key. It's also possible that the map explicitly maps the key to *null*.

**Example**

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
treeMap.put("2", "r");
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
Logger.log("TreeMap: " + treeMap.get("2"));
```

The example above returns the second element in this map.

Package [java.util](#)

Class [TreeMap](#)

**Public Method headMap**

```
SortedMap headMap(Object toKey)
```

**Throws:**

☞ [NotImplemented](#)

**This method is currently not available.**

Package [java.util](#)

Class [TreeMap](#)

**Public Method lastKey**

```
Object lastKey()
```

**Throws:**

☞ [NoSuchElementException](#)

**Description:**

Returns the last key currently in this sorted map.

**Example**

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
```

```
treeMap.put("2", "r");
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
Logger.log("TreeMap: " + treeMap.lastKey());
```

The example above returns the last key in this map.

Package [java.util](#)  
Class [TreeMap](#)

**Public Method put**

```
Object put(Object key, Object value)
```

**Overrides:**

☰ [put](#) in class [AbstractMap](#)

**Description:**

Associates the specified value with the specified key in this map. If the map previously contained a mapping for this key, the old value is replaced.

**Example**

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
treeMap.put("2", "r");
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
```

The example above puts seven elements with the according keys in this map.

Package [java.util](#)  
Class [TreeMap](#)

**Public Method putAll**

```
void putAll(Map map)
```

**Overrides:**

☰ [putAll](#) in class [AbstractMap](#)

**Description:**

Copies all of the mapping from the specified map to this map. These mappings replace any mappings that this map had for any of this keys currently in the specified map.

**Example**

```
TreeMap treeMap1 = new TreeMap();
treeMap1.put("1", "T");
treeMap1.put("2", "r");
treeMap1.put("3", "e");
treeMap1.put("4", "e");
treeMap1.put("5", "M");
treeMap1.put("6", "a");
treeMap1.put("7", "p");

TreeMap treeMap2 = new TreeMap();
treeMap2.putAll(treeMap1);
Logger.log("treeMap1: " + treeMap1);
Logger.log("treeMap2: " + treeMap2);
```

The example above puts all the elements with the according keys from the `treeMap1` into the `treeMap2`.

Package [java.util](#)

Class [TreeMap](#)

**Public Method `remove`**

Object **`remove`**(Object key)

**Overrides:**

☰ [remove](#) in class [AbstractMap](#)

**Description:**

Removes the mapping for the specified key from this *TreeMap* if present.

**Example**

```
TreeMap treeMap1 = new TreeMap();
treeMap1.put("1", "T");
treeMap1.put("2", "r");
treeMap1.put("3", "e");
treeMap1.put("4", "e");
treeMap1.put("5", "M");
treeMap1.put("6", "a");
treeMap1.put("7", "p");

TreeMap treeMap2 = new TreeMap();
treeMap2.putAll(treeMap1);
```

```
Logger.log("treeMap1: " + treeMap1);
treeMap2.remove("3");
Logger.log("treeMap2: " + treeMap2);
```

The example above returns puts all elements from the treeMap1 into the treeMap2 and removes the third element from the treeMap2.

Package [java.util](#)

Class [TreeMap](#)

**Public Method size**

```
int size()
```

**Overrides:**

⇒ [size](#) in class [AbstractMap](#)

**Description:**

Returns the number of key-value mappings in this *TreeMap*.

**Example**

```
TreeMap treeMap = new TreeMap();
treeMap.put("1", "T");
treeMap.put("2", "r");
treeMap.put("3", "e");
treeMap.put("4", "e");
treeMap.put("5", "M");
treeMap.put("6", "a");
treeMap.put("7", "p");
Logger.log("treeMap: " + treeMap.size());
```

The example above returns the size of this map.

Package [java.util](#)

Class [TreeMap](#)

**Public Method subMap**

```
SortedMap subMap(Object fromKey, Object toKey)
```

**Throws:**

⇒ [NotImplemented](#)

**This method is currently not available.**

Package [java.util](#)

Class [TreeMap](#)

**Public Method** [tailMap](#)

```
SortedMap tailMap(Object fromKey)
```

**Throws:**

☞ [NotImplemented](#)

**This method is currently not available.**

Package [java.util](#)

Class [TreeSet](#)

extends [AbstractSet](#) → [AbstractCollection](#) → [Object](#)

implements [SortedSet](#), [Cloneable](#), [Serializable](#), [Set](#), [Collection](#)

Implements the Set interface, backed by a TreeMap instance. This class guarantees that the sorted set will be in ascending order, according to natural order or according to order induced by the comparator provided at set creation time.

Note that the ordering must be consistent with the equals relation.

Note that this implementation is not synchronized. If multiple threads access a set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set. If no such object exists, the set should be "wrapped" using the Collections.synchronizedSet method. Iterators of this class are fail-fast, i.e. if the set is modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationException.

This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).

**Public Constructors**

☞ [TreeSet](#)

**Public Methods**

☞ [add](#)  
☞ [addAll](#)  
☞ [clear](#)  
☞ [clone](#)  
☞ [comparator](#)  
☞ [contains](#)  
☞ [first](#)  
☞ [headSet](#)  
☞ [isEmpty](#)  
☞ [iterator](#)  
☞ [last](#)  
☞ [remove](#)  
☞ [size](#)



- ▣ [subSet](#)
- ▣ [tailSet](#)

#### Methods inherited from [java.util.AbstractSet](#)

- ▣ [equals](#)
- ▣ [hashCode](#)

#### Methods inherited from [java.util.AbstractCollection](#)

- ▣ [iterator](#)
- ▣ [size](#)
- ▣ [add](#)
- ▣ [addAll](#)
- ▣ [clear](#)
- ▣ [contains](#)
- ▣ [containsAll](#)
- ▣ [isEmpty](#)
- ▣ [remove](#)
- ▣ [removeAll](#)
- ▣ [retainAll](#)
- ▣ [toArray](#)
- ▣ [toString](#)

#### Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
- ▣ [toString](#)
- ▣ [wait](#)
- ▣ [hashCode](#)
- ▣ [notify](#)
- ▣ [notifyAll](#)

### Package [java.util](#)

## Class [TreeSet](#)

### Public Constructor TreeSet

```
TreeSet ()
```

```
TreeSet (Collection c)
```

```
TreeSet (Comparator c)
```

```
TreeSet (SortedSet s)
```

#### Description:

##### constructor [TreeSet\(\)](#):

Constructs a new, empty set, sorted according to the elements' natural order. All elements inserted into the set must implement the Comparable interface. Furthermore, all such elements must be mutually comparable: `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the set. If the user attempts to add an element to the set that violates this constraint (for example, the user attempts to add a string element to a set whose elements are integers), the `add(Object)` call will throw a `ClassCastException`.

**constructor TreeSet(Collection c):**

Constructs a new set containing the elements in the specified collection, sorted according to the elements' natural order. All keys inserted into the set must implement the Comparable interface. Furthermore, all such keys must be mutually comparable: `k1.compareTo(k2)` must not throw a `ClassCastException` for any elements `k1` and `k2` in the set.

**constructor TreeSet(Comparator c):**

Constructs a new, empty set, sorted according to the given comparator. All elements inserted into the set must be mutually comparable by the given comparator: `comparator.compare(e1, e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the set. If the user attempts to add an element to the set that violates this constraint, the `add(Object)` call will throw a `ClassCastException`.

**constructor TreeSet(SortedSet s):**

Constructs a new set containing the same elements as the given sorted set, sorted according to the same ordering.

Package [java.util](#)

Class [TreeSet](#)

**Public Method add**

```
boolean add(Object o)
```

**Overrides:**

☞ [add](#) in class [AbstractCollection](#)

**Description:**

Adds the specified element to this set if it is not already present.

**Example**

```
TreeSet set = new TreeSet();
set.add("value1");
set.add("value2");

Logger.log("set: " + set);
```

The example above creates a new tree set and adds values to the set. Afterwards, the result is written to the logger.

Package [java.util](#)

Class [TreeSet](#)

**Public Method addAll**

```
boolean addAll(Collection c)
```

**Overrides:**

☞ [addAll](#) in class [AbstractCollection](#)

**Description:**

Adds all of the elements in the specified collection to this set. Implements interface method [Set.addAll](#).

Package [java.util](#)

Class [TreeSet](#)

**Public Method clear**

```
void clear ()
```

**Overrides:**

⇒ [clear](#) in class [AbstractCollection](#)

**Description:**

Removes all of the elements from this set. Implements interface method [Set.clear](#).

**Example**

```
TreeSet set = new TreeSet();
set.add("value1");
set.add("value2");

Logger.log("set: " + set);
set.clear();
Logger.log("set: " + set);
```

The example above creates a new tree set and adds values to the set. Afterwards, the set is cleared.

Package [java.util](#)

Class [TreeSet](#)

**Public Method clone**

```
Object clone ()
```

**Overrides:**

⇒ [clone](#) in class [Object](#)

**Throws:**

⇒ [Error](#)

**Description:**

Returns a shallow copy of this *TreeSet* instance. A shallow copy just copies the values of the references in the class (A deep copy copies the values). The elements themselves are not cloned.

## Example

```
TreeSet set1 = new TreeSet();
TreeSet set2 = new TreeSet();
set1.add("value1");
set1.add("value2");

set2 = (TreeSet)set1.clone();

Logger.log("set1: " + set1);
Logger.log("set2: " + set2);
```

The example above creates a new tree set and clones it to another instance of a tree set.

## Package [java.util](#) Class [TreeSet](#)

### Public Method comparator

```
Comparator comparator()
```

#### Description:

Returns the comparator used to order this sorted set, or null if this tree set uses its elements natural ordering. Implements interface method [SortedSet.comparator](#).

## Package [java.util](#) Class [TreeSet](#)

### Public Method contains

```
boolean contains(Object o)
```

#### Overrides:

☞ [contains](#) in class [AbstractCollection](#)

#### Description:

Returns *true* if this set contains the specified element. Implements interface method [Set.contains](#).

## Example

```
TreeSet set = new TreeSet();
set.add("value1");
set.add("value2");

if (set.contains("value1"))
    Logger.log("This set contains the value: value1");
```

The example above checks if set contains the *value1* string.

Package [java.util](#)  
Class [TreeSet](#)

**Public Method first**

```
Object first()
```

**Description:**

Returns the first (lowest) element currently in this sorted set. Implements interface method [SortedSet.first](#).

Package [java.util](#)  
Class [TreeSet](#)

**Public Method headSet**

```
SortedSet headSet(Object toElement)
```

**This method is currently not available.**

Package [java.util](#)  
Class [TreeSet](#)

**Public Method isEmpty**

```
boolean isEmpty()
```

**Overrides:**

☐ [isEmpty](#) in class [AbstractCollection](#)

**Description:**

Returns true if this set contains no elements. Implements interface method [Set.isEmpty](#).

Package [java.util](#)  
Class [TreeSet](#)

**Public Method iterator**

```
Iterator iterator()
```

**Overrides:**

☐ [iterator](#) in class [AbstractCollection](#)

**Description:**

Returns an iterator over the elements in this set. The elements are returned in ascending order. Implements interface method [Set.iterator](#).

## Example

```
Treeset = new Treeset();
set.add("value1");
set.add("value2");

Iterator iterator = set.iterator();

while (iterator.hasNext())
    Logger.log("value: " + iterator.next());
```

The example above iterates over the set.

Package [java.util](#)

Class [TreeSet](#)

### Public Method last

```
Object last()
```

#### Description:

Returns the last (highest) element currently in this sorted set. Implements interface method [SortedSet.last](#).

Package [java.util](#)

Class [TreeSet](#)

### Public Method remove

```
boolean remove(Object o)
```

#### Overrides:

☞ [remove](#) in class [AbstractCollection](#)

#### Description:

Removes the given element from this set if it is present. Implements interface method [Set.remove](#).

Package [java.util](#)

Class [TreeSet](#)

### Public Method size

```
int size()
```

#### Overrides:

☞ [size](#) in class [AbstractCollection](#)

#### Description:

Returns the number of elements in this set (its cardinality). Implements interface method [Set.size](#).

Package [java.util](#)  
Class [TreeSet](#)

**Public Method [subSet](#)**

```
SortedSet subSet(Object fromElement, Object toElement)
```

**This method is currently not available.**

Package [java.util](#)  
Class [TreeSet](#)

**Public Method [tailSet](#)**

```
SortedSet tailSet(Object fromElement)
```

**This method is currently not available.**

Package [java.util](#)

**Class Vector**

extends [AbstractList](#) → [AbstractCollection](#) → [Object](#)  
implements [List](#), [Cloneable](#), [Serializable](#), [Collection](#)

The Vector class implements an array of objects that can change in size. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

Each vector tries to optimize storage management by maintaining a capacity and a capacityIncrement. The capacity is always at least as large as the vector size; it is usually larger because as components are added to the vector, the vector's storage increases in chunks the size of capacityIncrement. An application can increase the capacity of a vector before inserting a large number of components; this reduces the amount of incremental reallocation.

**Public Constructors**

☞ [Vector](#)

**Public Methods**

☞ [add](#)  
☞ [addAll](#)  
☞ [addElement](#)  
☞ [capacity](#)  
☞ [clear](#)  
☞ [clone](#)  
☞ [contains](#)  
☞ [containsAll](#)  
☞ [copyInto](#)  
☞ [elementAt](#)  
☞ [elements](#)  
☞ [ensureCapacity](#)  
☞ [equals](#)

- ▣ [firstElement](#)
- ▣ [get](#)
- ▣ [hashCode](#)
- ▣ [indexOf](#)
- ▣ [insertElementAt](#)
- ▣ [isEmpty](#)
- ▣ [lastElement](#)
- ▣ [lastIndexOf](#)
- ▣ [remove](#)
- ▣ [removeAll](#)
- ▣ [removeAllElements](#)
- ▣ [removeElement](#)
- ▣ [removeElementAt](#)
- ▣ [retainAll](#)
- ▣ [set](#)
- ▣ [setElementAt](#)
- ▣ [setSize](#)
- ▣ [size](#)
- ▣ [subList](#)
- ▣ [toArray](#)
- ▣ [toString](#)
- ▣ [trimToSize](#)

#### Methods inherited from [java.util.AbstractList](#)

- ▣ [get](#)
- ▣ [add](#)
- ▣ [addAll](#)
- ▣ [clear](#)
- ▣ [equals](#)
- ▣ [hashCode](#)
- ▣ [indexOf](#)
- ▣ [iterator](#)
- ▣ [lastIndexOf](#)
- ▣ [listIterator](#)
- ▣ [remove](#)
- ▣ [set](#)
- ▣ [subList](#)

#### Methods inherited from [java.util.AbstractCollection](#)

- ▣ [iterator](#)
- ▣ [size](#)
- ▣ [add](#)
- ▣ [addAll](#)
- ▣ [clear](#)
- ▣ [contains](#)
- ▣ [containsAll](#)
- ▣ [isEmpty](#)
- ▣ [remove](#)
- ▣ [removeAll](#)
- ▣ [retainAll](#)
- ▣ [toArray](#)
- ▣ [toString](#)

#### Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
- ▣ [toString](#)
- ▣ [wait](#)
- ▣ [hashCode](#)
- ▣ [notify](#)



☰ [notifyAll](#)

Package [java.util](#)  
Class [Vector](#)

**Public Constructor Vector**

```
Vector()
```

```
Vector(Collection c)
```

```
Vector(int initialCapacity)
```

**Throws:**

☰ [IllegalArgumentException](#)

```
Vector(int initialCapacity, int capacityIncrement)
```

**Throws:**

☰ [IllegalArgumentException](#)

**Description:**

**constructor Vector():**

Constructs an empty vector so that its internal data array has size 10 and its standard capacity increment is zero.

**constructor Vector(Collection c):**

Constructs a vector containing the elements of the specified collection, in the order they are returned by the collection's iterator.

**constructor Vector(int initialCapacity):**

Constructs an empty vector with the specified initial capacity and with its capacity increment equal to zero.

**constructor Vector(int initialCapacity, int capacityIncrement):**

Constructs an empty vector with the specified initial capacity and capacity increment. The capacity increment is the amount by which the capacity of the vector is automatically incremented when its size becomes greater than its capacity. If the capacity increment is less than or equal to zero, the capacity of the vector is doubled each time it needs to grow.

Package [java.util](#)  
Class [Vector](#)

**Public Method add**

```
void add(int index, Object element)
```

**Overrides:**

☞ [add](#) in class [AbstractList](#)

```
boolean add(Object o)
```

**Overrides:**

☞ [add](#) in class [AbstractList](#)

**Description:**

Implements interface method [List.add](#).

**add(int index, Object element) method:**

Inserts the specified element at the specified position in this Vector. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

**add(Object o) method:**

Appends the specified element to the end of this Vector.

**Package [java.util](#)****Class [Vector](#)****Public Method addAll**

```
boolean addAll(Collection c)
```

**Overrides:**

☞ [addAll](#) in class [AbstractCollection](#)

```
boolean addAll(int index, Collection c)
```

**Overrides:**

☞ [addAll](#) in class [AbstractList](#)

**Throws:**

☞ [ArrayIndexOutOfBoundsException](#)

**Description:**

Implements interface method [List.addAll](#).

**addAll(Collection c) method:**

Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection's Iterator. The behavior of this operation is undefined if the specified Collection is modified while the operation is in progress. (This implies that the behavior of this call is undefined if the specified Collection is this Vector, and this Vector is nonempty.)

**addAll(int index, Collection c) method:**

Inserts all of the elements in in the specified Collection into this Vector at the specified position. Shifts the element currently at that position (if any) and any subsequent elements to the right (increases their indices). The new elements will appear in the Vector in the order that they are returned by the specified Collection's iterator.

Package [java.util](#)

Class [Vector](#)

**Public Method addElement**

```
void addElement(Object obj)
```

**Description:**

This method is identical in functionality to the add(Object) method (which is part of the List interface).

Package [java.util](#)

Class [Vector](#)

**Public Method capacity**

```
int capacity()
```

**Description:**

Returns the current capacity of this vector.


Package [java.util](#)

Class [Vector](#)

**Public Method clear**

```
void clear()
```

**Overrides:**

 [clear](#) in class [AbstractList](#)

**Description:**

Implements interface method [List.clear](#).

Removes all of the elements from this Vector. The Vector will be empty after this call returns (unless it throws an exception).

Package [java.util](#)

Class [Vector](#)

**Public Method clone**

```
Object clone()
```

**Overrides:**

☞ [clone](#) in class [Object](#)

**Throws:**

☞ [InternalError](#)

**Description:**

Returns a clone of this vector. The copy will contain a reference to a clone of the internal data array, not a reference to the original internal data array of this Vector object.

Package [java.util](#)

**Class [Vector](#)****Public Method contains**

```
boolean contains(Object elem)
```

**Overrides:**

☞ [contains](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.contains](#).

Tests if the specified object is a component in this vector.

Package [java.util](#)

**Class [Vector](#)****Public Method containsAll**

```
boolean containsAll(Collection c)
```

**Overrides:**

☞ [containsAll](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.containsAll](#).

Returns true if this Vector contains all of the elements in the specified Collection.

Package [java.util](#)

**Class [Vector](#)****Public Method copyInto**

```
void copyInto(Object[] anArray)
```

**Description:**

Copies the components of this vector into the specified array. The array must be big enough to hold all the objects in this vector, else an `IndexOutOfBoundsException` is thrown.

Package [java.util](#)

Class [Vector](#)

**Public Method `elementAt`**

```
Object elementAt(int index)
```

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)

**Description:**

This method is identical in functionality to the `get` method (which is part of the `List` interface).

Package [java.util](#)

Class [Vector](#)

**Public Method `elements`**

```
Enumeration elements()
```

**Description:**

Returns an enumeration of the components of this vector. The first item generated is the item at index 0, then the item at index 1, and so on.

Package [java.util](#)

Class [Vector](#)

**Public Method `ensureCapacity`**

```
void ensureCapacity(int minCapacity)
```

**Description:**

Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

If the current capacity of this vector is less than `minCapacity`, then its capacity is increased by replacing its internal data array, kept in the field `elementData`, with a larger one. The size of the new data array will be the old size plus `capacityIncrement`, unless the value of `capacityIncrement` is less than or equal to zero, in which case the new capacity will be twice the old capacity; but if this new size is still smaller than `minCapacity`, then the new capacity will be `minCapacity`.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method equals**

```
boolean equals (Object c)
```

**Overrides:**

⇒ [equals](#) in class [AbstractList](#)

**Description:**

Implements interface method [List.equals](#).

Compares the specified Object with this Vector for equality. Returns true if and only if the specified Object is also a List, both Lists have the same size, and all corresponding pairs of elements in the two Lists are equal. (Two elements e1 and e2 are equal if (e1==null ? e2==null : e1.equals(e2)).) In other words, two Lists are defined to be equal if they contain the same elements in the same order.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method firstElement**

```
Object firstElement ()
```

**Throws:**

⇒ [NoSuchElementException](#)

**Description:**

Returns the first component (the item at index 0) of this vector.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method get**

```
Object get (int index)
```

**Overrides:**

⇒ [get](#) in class [AbstractList](#)

**Description:**

Implements interface method [List.get](#).

Returns the element at the specified position in this Vector.

Package [java.util](#)  
Class [Vector](#)

**Public Method hashCode**

```
int hashCode()
```

**Overrides:**

↳ [hashCode](#) in class [AbstractList](#)

**Description:**

Implements interface method [List.hashCode](#).

Returns the hash code value for this Vector.

Package [java.util](#)  
Class [Vector](#)

**Public Method indexOf**

```
int indexOf(Object elem)
```

**Overrides:**

↳ [indexOf](#) in class [AbstractList](#)

```
int indexOf(Object e, int index)
```

**Description:**

Implements interface method [List.indexOf](#).

**indexOf(Object elem) method:**

Searches for the first occurrence of the given argument, testing for equality using the equals method.

**indexOf(Object e, int index) method:**

Searches for the first occurrence of the given argument, beginning the search at index, and testing for equality using the equals method.

Package [java.util](#)  
Class [Vector](#)

**Public Method insertElementAt**

```
void insertElementAt(Object obj, int index)
```

**Throws:**

↳ [ArrayIndexOutOfBoundsException](#)

**Description:**

Inserts the specified object as a component in this vector at the specified index. Each component in this vector with an index greater or equal to the specified index is shifted upward to have an index one greater than the value it had previously.

This method is identical in functionality to the `add(Object, int)` method (which is part of the `List` interface). Note that the `add` method reverses the order of the parameters, to more closely match array usage.

Package [java.util](#)

Class [Vector](#)

**Public Method isEmpty**

```
boolean isEmpty()
```

**Overrides:**

⇒ [isEmpty](#) in class [AbstractCollection](#)

Package [java.util](#)

Class [Vector](#)

**Public Method lastElement**

```
Object lastElement()
```

**Throws:**

⇒ [NoSuchElementException](#)

**Description:**

Returns the last component of the vector, i.e. the component at index `size() - 1`.

Package [java.util](#)

Class [Vector](#)

**Public Method lastIndexOf**

```
int lastIndexOf(Object elem)
```

**Overrides:**

⇒ [lastIndexOf](#) in class [AbstractList](#)

```
int lastIndexOf(Object e, int index)
```

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)



**Description:**

Implements interface method [List.hashCode](#).

**lastIndexOf(Object elem) method:**

Returns the index of the last occurrence of the specified object in this vector.

**lastIndexOf(Object e, int index) method:**

Searches backwards for the specified object, starting from the specified index, and returns the index of the last occurrence of the specified object in this vector at position less than or equal to index in the vector, that is, the largest value k such that `elem.equals(elementData[k]) && (k ≤ index)` is true; -1 if the object is not found. (Returns -1 if index is negative.)

Package [java.util](#)

**Class [Vector](#)****Public Method [remove](#)**

```
Object remove(int index)
```

**Overrides:**

⇒ [remove](#) in class [AbstractList](#)

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)

```
boolean remove(Object o)
```

**Overrides:**

⇒ [remove](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.remove](#).

**remove(int index) method:**

Removes the element at the specified position in this Vector. shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the Vector.

**remove(Object o) method:**

Removes the first occurrence of the specified element in this Vector. If the Vector does not contain the element, it is unchanged. More formally, removes the element with the lowest index i such that `(o==null ? get(i)==null : o.equals(get(i)))` (if such an element exists).

Package [java.util](#)  
Class [Vector](#)

**Public Method [removeAll](#)**

```
boolean removeAll(Collection c)
```

**Overrides:**

↳ [removeAll](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.removeAll](#).

Removes from this Vector all of its elements that are contained in the specified Collection.

Package [java.util](#)  
Class [Vector](#)

**Public Method [removeAllElements](#)**

```
void removeAllElements()
```

**Description:**

This method is identical in functionality to the clear method (which is part of the List interface).

Package [java.util](#)  
Class [Vector](#)

**Public Method [removeElement](#)**

```
boolean removeElement(Object obj)
```

**Description:**

This method is identical in functionality to the remove(Object) method (which is part of the List interface).

Package [java.util](#)  
Class [Vector](#)

**Public Method [removeElementAt](#)**

```
void removeElementAt(int index)
```

**Throws:**

↳ [ArrayIndexOutOfBoundsException](#)

**Description:**

This method is identical in functionality to the remove method (which is part of the List interface). Note that the remove method returns the old value that was stored at the specified position.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method [retainAll](#)**

```
boolean retainAll(Collection c)
```

**Overrides:**

⇒ [retainAll](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.retainAll](#).

Retains only the elements in this Vector that are contained in the specified Collection. In other words, removes from this Vector all of its elements that are not contained in the specified Collection.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method [set](#)**

```
Object set(int index, Object element)
```

**Overrides:**

⇒ [set](#) in class [AbstractList](#)

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)

**Description:**

Implements interface method [List.set](#).

Replaces the element at the specified position in this Vector with the specified element and returns the element previously at the specified position.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method [setElementAt](#)**

```
void setElementAt(Object obj, int index)
```

**Throws:**

⇒ [ArrayIndexOutOfBoundsException](#)

**Description:**

This method is identical in functionality to the set method (which is part of the List interface). Note that the set method reverses the order of the parameters, to more closely match array usage. Note also that the set method returns the old value that was stored at the specified position.

Package [java.util](#)

Class [Vector](#)

**Public Method `setSize`**

```
void setSize(int newSize)
```

**Description:**

Sets the size of this vector. If the new size is greater than the current size, new null items are added to the end of the vector. If the new size is less than the current size, all components at index newSize and greater are discarded.

Package [java.util](#)

Class [Vector](#)

**Public Method `size`**

```
int size()
```

**Overrides:**

☞ [size](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.size](#).

Returns the number of components in this vector.

Package [java.util](#)

Class [Vector](#)

**Public Method `subList`**

```
List subList(int fromIndex, int toIndex)
```

**Overrides:**

☞ [subList](#) in class [AbstractList](#)

**Description:**

Implements interface method [List.subList](#).

Returns a view of the portion of this List between fromIndex, inclusive, and toIndex (excluding toIndex). If fromIndex and ToIndex are equal, the returned List is empty. The returned List is backed by this List, so changes in the returned List are reflected in this List, and vice-versa.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method toArray**

```
Object[] toArray()
```

**Overrides:**

↳ [toArray](#) in class [AbstractCollection](#)

**Description:**

Implements interface method [List.toArray](#).

Returns an array containing all of the elements in this Vector in the correct order. The runtime type of the returned array is that of the specified array. If the Vector fits in the specified array, it is returned therein. Otherwise, a new array is allocated with the runtime type of the specified array and the size of this Vector.

If the Vector fits in the specified array with room to spare (i.e., the array has more elements than the Vector), the element in the array immediately following the end of the Vector is set to null. This is useful in determining the length of the Vector only if the caller knows that the Vector does not contain any null elements.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method toString**

```
String toString()
```

**Overrides:**

↳ [toString](#) in class [AbstractCollection](#)

**Description:**

Returns a string representation of this Vector, containing the String representation of each element.

**Package [java.util](#)**  
**Class [Vector](#)****Public Method trimToSize**

```
void trimToSize()
```

**Description:**

Trims the capacity of this vector to be the vector's current size. If the capacity of this vector is larger than its current size, then the capacity is changed to equal the size by replacing its internal data array, kept in the field `elementData`, with a smaller one. An application can use this operation to minimize the storage of a vector.

## Package javax

### Classes

- ▣ [Application](#)
  - ▣ [MainThreadException](#)
- 

## Package javax

### Abstract Class Application

extends [Object](#)

Extend the abstract class *Application* in order to use touch or keyboard events. Deriving from *Application* creates a custom application class. Note that since *Application* is abstract it cannot be instantiated. Touch event handling requires the implementation of the *OnTouchListener* interface in the custom application class. To do that the method *onTouch()* has to be implemented. This method will be called from the *TouchEventDispatcher* class whenever a touch event occurs. The *run()* method of the *Application* class will make sure that *onTouch()* will be called within the main thread to avoid strange iLCD behavior due to thread synchronization problems. Note that the iLCD panel is a global resource and changing panel settings and attributes in different threads likely leads to non-deterministic behavior of the panel. It is generally not recommended to change iLCD panel properties outside the main thread.

#### See:

- ▣ [Application Class Design](#)

#### Public Constructors

- ▣ [Application](#)

#### Public Methods

- ▣ [onUpdate](#)
- ▣ [requestShutdown](#)
- ▣ [run](#)

#### Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
  - ▣ [toString](#)
  - ▣ [wait](#)
  - ▣ [hashCode](#)
  - ▣ [notify](#)
  - ▣ [notifyAll](#)
- 

## Package javax

### Class [Application](#)

#### Public Constructor Application

**Application** ()

**Description:**

The `Application()` constructor builds an instance of the `Application` class.

**Note**

☞ When there are multiple instances of `Application` classes only one at a time can execute its `run()` method. Call [requestShutdown\(\)](#) to stop the current instance and start the `run()` method of another `Application` class.

**See also:**

[EventManager  
Common](#)

---

Package [javax](#)

Class [Application](#)

**Public Method [onUpdate](#)**

```
abstract void onUpdate()
```

**Description:**

The `onUpdate()` method gets called periodically. Use this method to perform periodic tasks like reading measurement values or updating the screen.

**Note**

☞ The `onUpdate()` method has to be implemented by every class that extends the `Application` class.

**See also:**

[Application](#)

---

Package [javax](#)

Class [Application](#)

**Public Method [requestShutdown](#)**

```
void requestShutdown()
```

**Throws:**

☞ [MainThreadException](#)

**Description:**

The `requestShutdown()` method is used when a shutdown for the main thread is required.

**See also:**

[Application](#)

---

Package [javax](#)

**Class [Application](#)****Public Method run**

```
void run ()
```

**Throws:**

☐ [MainThreadException](#)

**Description:**

This is the main event loop of the Java program. It makes sure that event listeners and the `onUpdate()` method are called periodically. To stop the execution of this method call `requestShutdown()` of the same `Application` class instance.

**Example**

```
public static void main(String[] args)
{
    try
    {
        App app = new App();
        app.run();
    }
    catch (MainThreadException ex)
    {
        Logger.log(ex.getMessage());
    }
    catch (Exception ex)
    {
        Logger.log(ex.getMessage());
    }
}
```

The example above shows the typical use of the `run()` method.

**See also:**

[Application](#)



## Package [javax](#)

### Class [MainThreadException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

Typically thrown by the [Application.run\(\)](#) method to indicate that the method is not executed in the main thread.

#### Public Constructors

☐ [MainThreadException](#)

#### Methods inherited from [java.lang.Throwable](#)

- ☐ [getMessage](#)
- ☐ [getLocalizedMessage](#)
- ☐ [toString](#)
- ☐ [fillInStackTrace](#)
- ☐ [getStackTrace](#)

#### Methods inherited from [java.lang.Object](#)

- ☐ [equals](#)
  - ☐ [toString](#)
  - ☐ [wait](#)
  - ☐ [hashCode](#)
  - ☐ [notify](#)
  - ☐ [notifyAll](#)
- 

## Package [javax](#)

### Class [MainThreadException](#)

#### Public Constructor [MainThreadException](#)

```
MainThreadException()
```

```
MainThreadException(String s)
```

#### Description:

##### constructor [MainThreadException](#)():

Constructs a new *MainThreadException* class without additional information.

##### constructor [MainThreadException\(String s\)](#):

Constructs a new *MainThreadException* class with additional information.

---

## Package javax.events

Provides classes for event handling and the use of timers.

### Classes

- ▣ [AsyncEvent](#)
  - ▣ [AsyncEventHandler](#)
  - ▣ [EventLimitException](#)
  - ▣ [EventManagement](#)
  - ▣ [Events](#)
  - ▣ [HandlerLimitException](#)
  - ▣ [OneShotTimerHandler](#)
  - ▣ [PeriodicTimerHandler](#)
  - ▣ [Timer](#)
- 

## Package [javax.events](#)

### Class [AsyncEvent](#)

extends [Object](#)

This is the base class for all asynchronous events.

#### Public Constructors

- ▣ [AsyncEvent](#)

#### Public Methods

- ▣ [addHandler](#)
- ▣ [fire](#)
- ▣ [getHandler](#)
- ▣ [handledBy](#)
- ▣ [removeHandler](#)
- ▣ [setHandler](#)

#### Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
  - ▣ [toString](#)
  - ▣ [wait](#)
  - ▣ [hashCode](#)
  - ▣ [notify](#)
  - ▣ [notifyAll](#)
- 

## Package [javax.events](#)

### Class [AsyncEvent](#)

#### Public Constructor [AsyncEvent](#)

```
AsyncEvent ()
```

```
AsyncEvent (int count)
```

**Description:****AsyncEvent() method:**

*AsyncEvent* constructor that creates an array of handlers that can hold just one event handler. This is the default constructor for majority asynch events.

**AsyncEvent(int count) method:**

Parameter	Description
count	number of handlers

*AsyncEvent* constructor that creates an array of handlers that can hold specified number of event handlers. The parameter *count* represents the number of handlers that can be associated with this event.

**Example**

```
AsyncEvent event = new AsyncEvent();
```

This example creates a new *AsyncEvent* instance.

**See also:**

[TouchEventDispatcher](#)  
[AsyncEventHandler](#)

Package [javax.events](#)

Class [AsyncEvent](#)

**Public Method addHandler**

```
int addHandler(AsyncEventHandler handler)
```

**Throws:**

☞ [HandlerLimitException](#)

**Description:**

Parameter	Description
handler	reference to the asynchronous event

This method adds a new asynchronous event handler to the current list of handlers.

Returns	Description
id	handler ID

Minus one is returned when the handler was not successfully added. If there are no more times available a `HandlerLimitException` is thrown.

### Example

```
public final class TouchEventDispatcher extends javax.events.AsyncEvent
{
    private EventHandler handler;
    ...
    this.addHandler(this.handler);
}
```

A new `AsyncEventHandler` is added to the list of asynchronous event handlers.

### See also:

[TouchEventDispatcher](#)  
[AsyncEventHandler](#)

---

Package [javax.events](#)

Class [AsyncEvent](#)

### Public Method fire

```
void fire ()
```

### Description:

This method is called by the event dispatcher class. The body must be implemented by all async events registered in the `Event` class. The `handlerAsyncEvent ()` method must be called for each registered event handler.

### Example

```
private AsyncEvent evList[] = new AsyncEvent[10];
...
evList[eventId].fire ();
```

The example above calls the `fire ()` method.

### See also:

[Events](#)  
[Timer](#)  
[AsyncEventHandler](#)

---

Package [javax.events](#)  
Class [AsyncEvent](#)

### Public Method `getHandler`

```
AsyncEventHandler getHandler(int index)
```

#### Description:

Parameter	Description
index	handler index

The *index* of the asynchronous event handler *AsyncEventHandler* to be retrieved.

Returns	Description
handler	asynchronous event handler

Returns the asynchronous event handler at the specified index. *null* is returned in case of an error.

#### Example

```
private AsyncEvent evList[] = new AsyncEvent[10];  
...  
evList[eventId].getHandler();
```

The example above returns the *AsyncEventHandler* with the specified event id.

#### See also:

[AsyncEventHandler](#)

---

Package [javax.events](#)  
Class [AsyncEvent](#)

### Public Method `handledBy`

```
boolean handledBy(AsyncEventHandler handler)
```

#### Description:

Parameter	Description
handler	asynchronous event handle

The specified asynchronous event handler which should be checked.

Returns	Range	Description
checked	true ... false	whether the handler was added or not

Whether the handler was successfully added to the list of asynchronous event handlers or not.

### Example

```
private AsyncEvent evList[] = new AsyncEvent[10];
...
if (evList[eventId].handledBy())
{ ...
```

The example above verifies if the specified asynchronous event handler was already added to the list of event handlers.

### See also:

[AsyncEventHandler](#)

---

Package [javax.events](#)

Class [AsyncEvent](#)

### Public Method `removeHandler`

```
void removeHandler(AsyncEventHandler handler)
```

### Description:

Parameter	Description
handler	asynchronous event handle

This method removes a specified handler from the list of asynchronous event handlers.

### Example

```
private AsyncEvent evList[] = new AsyncEvent[10];
...
evList[eventId].removeHandler();
```

The example above removes a specified asynchronous event handler from the list of event handlers.

### See also:

[AsyncEventHandler](#)

---

Package [javax.events](#)

## Class [AsyncEvent](#)

### Public Method [setHandler](#)

```
void setHandler(AsyncEventHandler handler)
```

#### Description:

Parameter	Description
handler	asynchronous event handle

This method removes all handlers from the handlers list and adds the current asynchronous handler to the handler list.

#### Example

```
private AsyncEvent evList[] = new AsyncEvent[10];  
...  
evList[eventId].setHandler();
```

The example above removes all asynchronous event handler from the event handler list and adds the specified asynchronous event handler to the event handler list.

#### See also:

[AsyncEventHandler](#)

---

Package [javax.events](#)

## Abstract Class [AsyncEventHandler](#)

extends [Object](#)

This is a abstract asynchronous class which must be inherited by all event handler classes. This approach provides a common base class for all event handler classes and to force them to implement the [handlerAsyncEvent\(\)](#) method.

### Public Constructors

⇒ [AsyncEventHandler](#)

### Public Methods

⇒ [handleAsyncEvent](#)

### Methods inherited from [java.lang.Object](#)

⇒ [equals](#)

⇒ [toString](#)

- ☐ [wait](#)
  - ☐ [hashCode](#)
  - ☐ [notify](#)
  - ☐ [notifyAll](#)
- 

Package [javax.events](#)

## Class [AsyncEventHandler](#)

### Public Constructor [AsyncEventHandler](#)

```
AsyncEventHandler ()
```

Package [javax.events](#)

## Class [AsyncEventHandler](#)

### Public Method [handleAsyncEvent](#)

```
abstract void handleAsyncEvent ()
```

#### Description:

The subclasses of the *AsyncEventHandler* must implement the actual event processing code.

#### Example

```
public class Timer extends AsyncEvent  
{  
    ...  
    if (evHandlers[i] != null)  
    {
```

The *Timer* class inherits from the *AsyncEvent* class and get access to the *evHandlers[]* reference.

#### See also:

[AsyncEvent](#)  
[OneShotTimerHandler](#)  
[PeriodicTimerHandler](#)  
[Timer](#)

---

Package [javax.events](#)

## Class [EventLimitException](#)

extends [Exception](#) → [Throwable](#) → [Object](#)

The *EventLimitException* is used to throw an exception when the application code is trying to add more events which are declared to the handler.



## Public Constructors

☞ [EventLimitException](#)

## Methods inherited from [java.lang.Throwable](#)

☞ [getMessage](#)  
☞ [getLocalizedMessage](#)  
☞ [toString](#)  
☞ [fillInStackTrace](#)  
☞ [getStackTrace](#)

## Methods inherited from [java.lang.Object](#)

☞ [equals](#)  
☞ [toString](#)  
☞ [wait](#)  
☞ [hashCode](#)  
☞ [notify](#)  
☞ [notifyAll](#)

---

## Package [javax.events](#)

## Class [EventLimitException](#)

### Public Constructor [EventLimitException](#)

```
EventLimitException()
```

```
EventLimitException(String s)
```

#### Description:

#### **EventLimitException()** method:

This method creates a new *EventLimitException* object.

#### **EventLimitException(String s)** method:

Parameter	Description
s	detailed message

This method creates a new *EventLimitException* object with a detailed message.

#### See also:

[Events](#)

---

## Package [javax.events](#)

### Class **EventManagement**

extends [Object](#)

With the class *EventManagement* it is possible to register several timer, touch or comm events.

Event management has to be started with [EventManagement.start\(\)](#) before any event can be registered. Note that when using touch events it's not necessary to call this method because the [Touch.setTouchFieldReportingEnabled\(boolean\)](#) method makes this call internally.

#### Note

- ☞ Please note that in order to receive touch and keyboard events the abstract class [Application](#) has to be extended (create a class derived from *Application*). This ensures that touch and keyboard events will be collected periodically and the corresponding handler methods will be executed in the main thread (for comm events this is not necessary). See e.g. the touch event samples ([iLCD Manager File Tab->New](#)) and [Application Class Design](#) on how to do that.
- ☞ Don't start the event thread ([EventManagement.start\(\)](#)) before *main ()*.

Event listeners (or handlers) can be added as follows:

- ☞ touch events: [EventManagement.getTouchEventDispatcher\(\).addListener\(OnTouchListenerObject\)](#)
- ☞ timer events: [EventManagement.getTimerEvent\(\).addListener\(AsyncEventHandlerObject\)](#)
- ☞ I2C events: [EventManagement.getI2CEventDispatcher\(\).addListener\(OnI2CListenerObject\)](#)
- ☞ SPI events: [EventManagement.getSPIEventDispatcher\(\).addListener\(OnSPIListenerObject\)](#)
- ☞ UART events: [EventManagement.getUARTEventDispatcher\(\).addListener\(OnUARTListenerObject\)](#)
- ☞ USB events: [EventManagement.getUSBEventDispatcher\(\).addListener\(OnUSBListenerObject\)](#)
- ☞ Ethernet events: [EventManagement.getEthernetEventDispatcher\(\).addListener\(OnEthernetListenerObject\)](#)

#### Public Methods

- ☞ [getEthernetEventDispatcher](#)
- ☞ [getI2CEventDispatcher](#)
- ☞ [getKeyboardEventDispatcher](#)
- ☞ [getRotaryEncoderEventDispatcher](#)
- ☞ [getSPIEventDispatcher](#)
- ☞ [getTimerEvent](#)
- ☞ [getTouchEventDispatcher](#)
- ☞ [getUARTEventDispatcher](#)
- ☞ [getUSBEventDispatcher](#)
- ☞ [start](#)
- ☞ [stop](#)

#### Methods inherited from [java.lang.Object](#)

- ☞ [equals](#)
- ☞ [toString](#)
- ☞ [wait](#)
- ☞ [hashCode](#)
- ☞ [notify](#)
- ☞ [notifyAll](#)

#### Public Fields

- ☞ static final int EV\_Ethernet
- ☞ static final int EV\_I2C
- ☞ static final int EV\_KEYBOARD
- ☞ static final int EV\_ROTARY
- ☞ static final int EV\_SPI

- ☐ static final int EV\_SWTIMERS
  - ☐ static final int EV\_TOUCH
  - ☐ static final int EV\_UART
  - ☐ static final int EV\_USB
- 

Package [javax.events](#)

## Class [EventManager](#)

### Public Method [getEthernetEventDispatcher](#)

```
static CommEventDispatcher getEthernetEventDispatcher()
```

Returns a Ethernet event dispatcher object for adding Ethernet listeners.

Returns	Description
<a href="#">EthernetEventDispatcher</a>	EthernetEventDispatcher object

---

Package [javax.events](#)

## Class [EventManager](#)

### Public Method [getI2CEventDispatcher](#)

```
static CommEventDispatcher getI2CEventDispatcher()
```

#### Description:

Returns an I2C event dispatcher object for adding I2C event listeners.

Returns	Description
<a href="#">I2CEventDispatcher</a>	I2CEventDispatcher object

---

Package [javax.events](#)

## Class [EventManager](#)

### Public Method [getKeyboardEventDispatcher](#)

```
static KeyboardEventDispatcher getKeyboardEventDispatcher()
```

Package [javax.events](#)

## Class [EventManager](#)

### Public Method [getSPIEventDispatcher](#)

```
static CommEventDispatcher getSPIEventDispatcher()
```

Returns a SPI event dispatcher object for adding SPI listeners.

Returns	Description
<a href="#">SPIEventDispatcher</a>	SPIEventDispatcher object

Package [javax.events](#)

## Class [EventManagement](#)

### Public Method [getUSBEventDispatcher](#)

```
static CommEventDispatcher getUSBEventDispatcher ()
```

Returns a USB event dispatcher object for adding USB listeners.

Returns	Description
<a href="#">USBEventDispatcher</a>	USBEventDispatcher object

Package [javax.events](#)

## Class [EventManagement](#)

### Public Method [getTimerEvent](#)

```
static Timer getTimerEvent ()
```

#### Description:

Returns	Description
<a href="#">Timer</a>	Timer object

The `getTimerEvent()` method returns a *Timer* object, and provides timer specific methods. Note that the *Timer* class is extended with the *AsyncEvent* class.

#### Example

```
// The TimerIntervalTest class must be implemented by the user,
// and extended by the PeriodicTimerHandler.
// A timer sample is available in the iLCD Manager XE,
// and is named as "Java Timer Sample".
TimerIntervalTest handler = new TimerIntervalTest();
EventManagement.getTimerEvent().addHandler(handler);
```

#### See also:

[Timer](#)  
[AsyncEvent](#)

---

Package [javax.events](#)

## Class [EventManager](#)

### Public Method [getTouchEventDispatcher](#)

```
static TouchEventDispatcher getTouchEventDispatcher()
```

#### Description:

Returns	Description
<a href="#">TouchEventDispatcher</a>	TouchEventDispatcher object

The *getTouchEventDispatcher()* method returns a *TouchEventDispatcher* object, and provides touch event specific methods. Note that the *TouchEventDispatcher* class is extended with the *AsyncEvent* class.

#### Example

```
// The class which adds a listener must implement
// the OnTouchListener interface.
// A timer sample is available in the iLCD Manager XE,
// and is named as "Java Timer Sample".
EventManager.getTouchEventDispatcher().addListener(this);
// When the OnTouchListener interface is used,
// the onTouch(TouchEvent) method must be implemented.
public void onTouch(TouchEvent event)
{ ...
```

#### See also:

[TouchEventDispatcher](#)

[AsyncEvent](#)

---

Package [javax.events](#)

## Class [EventManager](#)

### Public Method [getUARTEventDispatcher](#)

```
static CommEventDispatcher getUARTEventDispatcher()
```

#### Description:

Returns an UART event dispatcher object for adding UART event listeners.

---

Returns	Description
<a href="#">UARTEventDispatcher</a>	UARTEventDispatcher object

---

Package [javax.events](#)

## Class [EventManagement](#)

### Public Method [start](#)

```
static void start()
```

#### Description:

With the `start()` method a new `Event` thread is started. This thread handles all types of events: touch field, keyboard, comm events.

#### Example

```
EventManagement.start();
```

#### Note

☞ Note that for comm events (SPI, UART, USB) any corresponding receive buffers will be cleared.

#### See also:

[Timer](#)

[Common](#)

[setTouchFieldReportingEnabled](#)

[setKeyboardReportingEnabled](#)

---

Package [javax.events](#)

## Class [EventManagement](#)

### Public Method [stop](#)

```
static void stop()
```

#### Description:

The `stop()` method will stop the `Event` thread. The `EventManagement` can be started again with the `start()` method. Please consider, once the `Event` thread is stopped also any timer will be stopped regardless whether the `Event` thread is started again.

#### Example

```
EventManagement.stop();
```

**See also:**

[Timer](#)  
[Common](#)  
[EventManager](#)

---

**Package [javax.events](#)****Class Events**

extends [Thread](#) → [Object](#)

implements [Runnable](#)

This class implements the event dispatcher. The `run()` method of this class implements an infinite loop in which this thread waits (suspended) for an event to occur. When an event is recognised the VM thread scheduler immediately resumes this thread so it can call `fire()` methods of registered AsyncEvent handler. If there are several events pending then they are processed in turn until all fire counters are decremented to zero.

**Note**

☞ There can be only one instance of this thread. If there are more instances of this class then only the last created instance will receive event notifications.

**Public Constructors**

☞ [Events](#)

**Public Methods**

☞ [addEvent](#)  
☞ [clearPendingFireCount](#)  
☞ [removeAll](#)  
☞ [run](#)

**Methods inherited from [java.lang.Thread](#)**

☞ [sleep](#)  
☞ [interrupt](#)  
☞ [interrupted](#)  
☞ [isInterrupted](#)  
☞ [yield](#)  
☞ [start](#)  
☞ [stop](#)  
☞ [run](#)  
☞ [setDaemon](#)  
☞ [isDaemon](#)  
☞ [join](#)  
☞ [isAlive](#)  
☞ [suspend](#)  
☞ [resume](#)  
☞ [currentThread](#)  
☞ [currentThreadId](#)  
☞ [getMaxThreadsCount](#)

## Methods inherited from [java.lang.Object](#)

- ☐ [equals](#)
- ☐ [toString](#)
- ☐ [wait](#)
- ☐ [hashCode](#)
- ☐ [notify](#)
- ☐ [notifyAll](#)

## Public Fields

- ☐ static final int ALL\_EVENTS
  - ☐ static int MAX\_EVENTS
- 

## Package [javax.events](#)

## Class [Events](#)

### Public Constructor Events

**Events** ()

#### Description:

Creates an instance of the *Events* class.

#### Example

```
Events events = new Events();
```

This example creates a new *events* instance.

#### See also:

[EventManager](#)

---

## Package [javax.events](#)

## Class [Events](#)

### Public Method addEvent

```
void addEvent(AsyncEvent event, int index)
```

#### Throws:

- ☐ [EventLimitException](#)

#### Description:

Paramete	Description
----------	-------------



r	
event	event to be added
index	index to the event

The `addEvent()` method adds a new event into the list of registered events. Particular events must be placed at indexes in the event list that match the order of fire counters implemented in native code. Note that the Timer event must be always at index zero.

### Example

```
private Events events = new Events();
...
private void registerEvent(AsyncEvent event, int queue)
{ ...
    this.events.addEvent(event, queue);
}
```

This example creates a new `events` instance and adds a new asynchronous event to the event queue.

### See also:

[EventManager](#)  
[Events](#)

## Package [javax.events](#)

### Class [Events](#)

#### Public Method `clearPendingFireCount`

```
boolean clearPendingFireCount(AsyncEvent event)
static void clearPendingFireCount(int eventId)
```

#### Description:

#### `clearPendingFireCount(AsyncEvent event)` method:

Parameter	Description
event	async event for which fire counter should be cleared

There is a counter associated with each event. It's possible that the counter is incremented faster than the corresponding event handler can process requests. This method will clear the fire counter for the specified event.

Returns	Range	Description
---------	-------	-------------

cleared	true ... false	whether a pending event is cleared or not
---------	----------------	---

#### clearPendingFireCount(int eventId) method:

Parameter	Description
eventId	eventId for which fire counter should be cleared

This method will clear the fire counter for the event specified by its ID. If ALL\_EVENTS is specified then all fire counters are cleared.

#### Example

```
private Events events = new Events();
...
this.events.clearPendingFireCount(event);
```

This example creates a new *events* instance and clears the pending event.

#### Note

☞ Note that for comm events (SPI, UART, USB) any corresponding receive buffers will be cleared.

Package [javax.events](#)

Class [Events](#)

#### Public Method removeAll

```
void removeAll()
```

#### Description:

Removes all events from the list of registered events.

#### Example

```
private Events events = new Events();
...
this.events.removeAll();
```

This example creates a new *events* instance and removes all events from the list of registered events.

**Package [javax.events](#)****Class [Events](#)****Public Method [run](#)**

```
void run ()
```

**Overrides:**

⇒ [run](#) in class [Thread](#)

**Description:**

The `run()` method implements an infinite loop. Once all events are processed this thread enters a suspend mode. The execution is resumed when an event is recognized.

**Example**

```
private Events events = new Events();  
...  
this.events.run();
```

This example creates a new `events` instance and calls the `run()` method to start a new thread.

---

**Package [javax.events](#)****Class [HandlerLimitException](#)**

extends [Exception](#) → [Throwable](#) → [Object](#)

The `HandlerLimitException` is used to throw an exception when the application code is trying to add more events which are declared to the handler.

**Public Constructors**

⇒ [HandlerLimitException](#)

**Methods inherited from [java.lang.Throwable](#)**

- ⇒ [getMessage](#)
- ⇒ [getLocalizedMessage](#)
- ⇒ [toString](#)
- ⇒ [fillInStackTrace](#)
- ⇒ [getStackTrace](#)

**Methods inherited from [java.lang.Object](#)**

- ⇒ [equals](#)
  - ⇒ [toString](#)
  - ⇒ [wait](#)
  - ⇒ [hashCode](#)
  - ⇒ [notify](#)
  - ⇒ [notifyAll](#)
-

Package [javax.events](#)

## Class [HandlerLimitException](#)

### Public Constructor [HandlerLimitException](#)

```
HandlerLimitException()
```

```
HandlerLimitException(String s)
```

#### Description:

#### [HandlerLimitException\(\)](#) method:

This method creates a new *HandlerLimitException* object.

#### [HandlerLimitException\(String s\)](#) method:

Parameter	Description
s	detailed message

This method creates a new *HandlerLimitException* object with a detailed message.

#### See also:

[AsyncEvent](#)

---

Package [javax.events](#)

## Abstract Class [OneShotTimerHandler](#)

extends [AsyncEventHandler](#) → [Object](#)

This class is a base class for handlers that responds to single shot timer events. Derived classes must implement the body of the [handleAsyncEvent\(\)](#) method.

### Public Constructors

▣ [OneShotTimerHandler](#)

### Methods inherited from [javax.events.AsyncEventHandler](#)

▣ [handleAsyncEvent](#)

### Methods inherited from [java.lang.Object](#)

▣ [equals](#)

▣ [toString](#)

▣ [wait](#)

▣ [hashCode](#)

▣ [notify](#)

☰ [notifyAll](#)

---

Package [javax.events](#)

## Class [OneShotTimerHandler](#)

### Public Constructor [OneShotTimerHandler](#)

```
OneShotTimerHandler ()
```

---

Package [javax.events](#)

## Abstract Class [PeriodicTimerHandler](#)

extends [AsyncEventHandler](#) → [Object](#)

This class is a base class for handlers that responds to single shot timer events. Derived classes must implement the body of the [handleAsyncEvent\(\)](#) method.

### Public Constructors

☰ [PeriodicTimerHandler](#)

### Methods inherited from [javax.events.AsyncEventHandler](#)

☰ [handleAsyncEvent](#)

### Methods inherited from [java.lang.Object](#)

☰ [equals](#)  
☰ [toString](#)  
☰ [wait](#)  
☰ [hashCode](#)  
☰ [notify](#)  
☰ [notifyAll](#)

---

Package [javax.events](#)

## Class [PeriodicTimerHandler](#)

### Public Constructor [PeriodicTimerHandler](#)

```
PeriodicTimerHandler ()
```

---

## Package [javax.events](#)

### Class **Timer**

extends [AsyncEvent](#) → [Object](#)

The Timer class is a special type of asynchronous events that requires additional native support. The maximum supported timeout value is 655635 milliseconds. Two asynchronous timer are supported:

- ▣ [OneShotTimerHandler](#)
- ▣ [PeriodicTimerHandler](#)

#### Note

- ▣ There can be only one handler associated with each timer.
- ▣ Timer event handler must be always present at index zero when registered with `Events.addEvent` method.

#### Public Constructors

- ▣ [Timer](#)

#### Public Methods

- ▣ [disable](#)
- ▣ [enable](#)
- ▣ [fire](#)
- ▣ [start](#)

#### Methods inherited from [javax.events.AsyncEvent](#)

- ▣ [addHandler](#)
- ▣ [getHandler](#)
- ▣ [handledBy](#)
- ▣ [removeHandler](#)
- ▣ [setHandler](#)
- ▣ [fire](#)

#### Methods inherited from [java.lang.Object](#)

- ▣ [equals](#)
- ▣ [toString](#)
- ▣ [wait](#)
- ▣ [hashCode](#)
- ▣ [notify](#)
- ▣ [notifyAll](#)

#### Public Fields

- ▣ static int MAX\_TIMERS
- 

## Package [javax.events](#)

### Class [Timer](#)

#### Public Constructor [Timer](#)

```
Timer ()
```

**Description:**

This method constructs a new *Timer* object.

**Example**

```
Timer timer = new Timer();
```

This example creates a new *Timer* instance.

**See also:**

[EventManagerment](#)

---

Package [javax.events](#)

Class [Timer](#)

**Public Method disable**

```
void disable(AsyncEventHandler handler)
```

**Description:**

Parameter	Description
handler	reference to the asynchronous event

This method disable a specific asynchronous event handler.

**Example**

```
Timer timer = new Timer();  
...  
this.timer.disable(event);
```

This example creates a new *Timer* instance and disables this timer.

**See also:**

[EventManagerment](#)

---

**Package [javax.events](#)****Class [Timer](#)****Public Method [enable](#)**

```
void enable(AsyncEventHandler handler)
```

**Description:**

Parameter	Description
handler	reference to the asynchronous event

This method enables a specific asynchronous event handler.

**Example**

```
Timer timer = new Timer();  
...  
this.timer.enable(event);
```

This example creates a new *Timer* instance and enables this timer.

**See also:**

[EventManager](#)

---

**Package [javax.events](#)****Class [Timer](#)****Public Method [fire](#)**

```
void fire()
```

**Overrides:**

☐ [fire](#) in class [AsyncEvent](#)

**Description:**

The *fire()* method calls the handler methods for each registered timer event handler. Only those timer handlers are called for which timeout has expired.

**Example**

```
timer.fire();
```

**See also:**

[EventManager](#)

---



**Package [javax.events](#)****Class [Timer](#)****Public Method [start](#)**

```
void start(AsyncEventHandler handler, int timeout)
```

**Description:**

Parameter	Range	Description
handler	-	asynchronous event handle
timeout	0 ... 65535	specified time

This method starts the specified timer and sets up the timers timeout value.

**Example**

```
timer.start(event);
```

**See also:**

[EventManager](#)  
[Touch](#)

**Firmware Version History**

Version	Release Date	Description
6.15	September 19, 2022	<ul style="list-style-type: none"> <li>☞ Fix: Rounding error during PWM signal calculation.</li> <li>☞ Rotary Encoder implemented for DPM5050.</li> </ul>
6.14	June 15, 2022	<ul style="list-style-type: none"> <li>☞ Support for Goodix touch controller added.</li> </ul>
6.13	June 15, 2022	<ul style="list-style-type: none"> <li>☞ DPP-F102 added.</li> </ul>
6.12	March 1, 2022	<ul style="list-style-type: none"> <li>☞ Various smaller improvements.</li> </ul>
6.11	March 1, 2022	<ul style="list-style-type: none"> <li>☞ DPP-FHC70 added.</li> </ul>
6.10	February 8, 2022	<ul style="list-style-type: none"> <li>☞ Fix: Java binary did not return to Macro execution after stop instruction.</li> </ul>
6.09	January 18, 2022	<ul style="list-style-type: none"> <li>☞ Various bugfixes.</li> </ul>
6.08	December 21, 2021	<ul style="list-style-type: none"> <li>☞ Improvements to SD card read/write functions.</li> </ul>

6.07	October 10, 2021	<ul style="list-style-type: none"> <li>☞ Added new display types with sizes 4.3" and 5.0".</li> </ul>
6.06	September 27, 2021	<ul style="list-style-type: none"> <li>☞ New command: <a href="#">Get PCAP controller information</a> for Macros and Java.</li> <li>☞ Simultaneous function of both USB ports enabled.</li> </ul>
6.05	August 31, 2021	<ul style="list-style-type: none"> <li>☞ Blink function added to I/O ports.</li> <li>☞ New command: <code>GetDate()</code>.</li> <li>☞ Fix: Alert pin functionality.</li> </ul>
6.04	June 30, 2021	<ul style="list-style-type: none"> <li>☞ Improvements to internal timing mechanism.</li> </ul>
6.03	June 28, 2021	<ul style="list-style-type: none"> <li>☞ Fixed issue with Flash Write.</li> </ul>
6.02	June 21, 2021	<ul style="list-style-type: none"> <li>☞ Implemented Software Watchdog.</li> </ul>
6.01	June 2, 2021	<ul style="list-style-type: none"> <li>☞ Improvements to I<sup>2</sup>C functions.</li> </ul>
6.00	April 29, 2021	<ul style="list-style-type: none"> <li>☞ First Firmware and Bootloader release for new fast iLCD series based on the controller DPM5050. Note: Older DPC3090 models stay at version 5.08.</li> </ul>
5.08	July 28, 2020	<ul style="list-style-type: none"> <li>☞ Rotary encoder implementation improved.</li> <li>☞ Fix: I2C not working after Java VM exit.</li> </ul>
5.07	May 2, 2019	<ul style="list-style-type: none"> <li>☞ Rotary encoder support added.</li> <li>☞ New command: <a href="#">Format MicroSD Card</a>.</li> <li>☞ DAC feature added: <a href="#">Set DAC Value</a>.</li> <li>☞ String cache changed to hash table.</li> </ul>
5.06	April 26, 2018	<ul style="list-style-type: none"> <li>☞ USB support for Java added.</li> <li>☞ Ethernet support for Java added.</li> </ul>
5.05	March 28, 2018	<ul style="list-style-type: none"> <li>☞ Fix: UART bug when receiving single byte in Java.</li> </ul>
5.04	March 19, 2018	<ul style="list-style-type: none"> <li>☞ Java event handling added for SPI, I2C, UART, Keyboard events.</li> <li>☞ Fix: Ethernet bug, communication problems and panel reboots when using Ethernet.</li> <li>☞ Fix: NAND Flash issues when relocating blocks many times.</li> </ul>
5.03	September 5, 2017	<ul style="list-style-type: none"> <li>☞ Fix: SD Card file handling in Java. Number of bytes read was not correct when EOF was hit.</li> </ul>
5.02	July 12, 2017	<ul style="list-style-type: none"> <li>☞ SPI and SD Card support for Java added.</li> </ul>
5.01	April 28, 2017	<ul style="list-style-type: none"> <li>☞ Fix: some minor bugs.</li> </ul>
5.00	March 12, 2017	<ul style="list-style-type: none"> <li>☞ Java VM integrated in DPC3090</li> </ul>
4.71	March 22, 2016	<ul style="list-style-type: none"> <li>☞ Added command <a href="#">Write Scan Line and Advance</a></li> </ul>
4.70	January 22, 2016	<ul style="list-style-type: none"> <li>☞ Fix: Touch field reported false make- and brake events during calibration.</li> </ul>
4.69	Dezember 28, 2015	<ul style="list-style-type: none"> <li>☞ Fix: More often relocate Nand Flash pages because of read disturb</li> </ul>
4.68	Dezember 16, 2015	<ul style="list-style-type: none"> <li>☞ Fix: Command <a href="#">Execute Touch Make Macro</a> and <a href="#">Execute Touch Break Macro</a> with field_idx set to 0xFF use now the correct macro</li> </ul>
4.67	November 2, 2015	<ul style="list-style-type: none"> <li>☞ Fix: Problems with first time NAND-flash initialization (during production)</li> </ul>

4.65	June 25, 2015 (not released)	<ul style="list-style-type: none"> <li>☐ Info screen is shown if flash memory empty</li> <li>☐ Fix: Capacitive touch screen missed events when under full load</li> <li>☐ Fix: <a href="#">Write File</a> via WinUSB at full speed</li> <li>☐ Fix: RTC was initialized a few milliseconds after startup (refer to <a href="#">Time/Date Related Commands</a>)</li> </ul>
4.64	February 17, 2015	<ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Retrieve Last Touch Screen Event</a> didn't include move events if asynchronous movement reporting (refer to <a href="#">Enable/Disable Reporting Movements</a>) was turned off (since v4.50)</li> </ul>
4.63	January 29, 2015 (not released)	<ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Restore Cursor &amp; Attributes from Memory</a> with same font as set before could use wrong colors for anti-aliased characters</li> </ul>
4.62	January 23, 2015 (not released)	<ul style="list-style-type: none"> <li>☐ Fix: Graphics on MicroSD card not found when path had no trailing '/'</li> <li>☐ Fix: System messages now ignore scale factor</li> </ul>
4.61	January 21, 2015 (not released)	<ul style="list-style-type: none"> <li>☐ Fix: Font spacing ignored font scale factor when not modified via <a href="#">Set Font Spacing</a></li> </ul>
4.60	January 16, 2015 (not released)	<ul style="list-style-type: none"> <li>☐ Now supporting anti-aliased fonts</li> <li>☐ Patch: <a href="#">Set Touch Field Width</a> and <a href="#">Set Touch Field Height</a> with parameter 0 means full width/height now</li> <li>☐ Patch: Modify cursor position after auto-scrolling (due to vertical text wrapping) to ensure next line can be displayed non-cropped</li> <li>☐ Fix: DPP-CT1060 "inverted" resistive touch always failed calibration (since v4.50)</li> <li>☐ Fix: Scaled, not-transparent fonts had gaps between characters</li> <li>☐ Fix: Last character in line before horizontal text wrapping could be printed over last one if narrow</li> <li>☐ Fix: Scaled fonts on DPC3050 showed solid lines</li> </ul>
4.51	December 16, 2014	<ul style="list-style-type: none"> <li>☐ Fix: Improved flash writing speed for iLCDs with DPC3090</li> </ul>
4.50	December 12, 2014	<ul style="list-style-type: none"> <li>☐ Now supporting projected capacitive (PCAP) touch screens</li> <li>☐ Added new touch commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Get Touch Screen Type</a></li> <li>☐ <a href="#">Set Number of Touch Fingers</a></li> <li>☐ <a href="#">Set Threshold for Movement Reporting</a></li> <li>☐ <a href="#">Set Cursor to Touch Field</a></li> <li>☐ <a href="#">Set PCAP Configuration to Factory Default</a></li> <li>☐ <a href="#">Enable/Disable Touch Field Queue</a></li> <li>☐ <a href="#">Start Demonstration Mode</a></li> </ul> </li> <li>☐ Added new input/output commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Disable Communication-Ports</a></li> <li>☐ <a href="#">Get Enabled Communication-Ports</a></li> </ul> </li> <li>☐ Now supporting new 7" iLCD Panel DPP-x70</li> </ul>
4.31	September 17, 2014	<ul style="list-style-type: none"> <li>☐ Now supporting: <ul style="list-style-type: none"> <li>☐ new 4.3" iLCD Panel DPP-x43</li> <li>☐ new 5.7" iLCD Panel DPP-x57</li> </ul> </li> </ul>
4.30	July 22, 2014	<ul style="list-style-type: none"> <li>☐ Fix: First called macro didn't return ACK if no startup macro set (since v4.25)</li> </ul>
4.29	June 23, 2014	<ul style="list-style-type: none"> <li>☐ Patch: Prevent longer startup after lots of block relocates in Nand flash (since v4.16)</li> </ul>
4.28	June 16, 2014	<ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Get Text Message Extent</a> didn't take text message offset, prefix or suffix into account</li> </ul>

4.27	May 30, 2014	<ul style="list-style-type: none"> <li>☐ Fix: Erase jumper on iLCDs with Nand flash only invalidated flash until reset (since v4.16)</li> </ul>
4.26	May 28, 2014	<ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Screen Memory Related Commands</a> could crash firmware</li> </ul>
4.25	May 27, 2014	<ul style="list-style-type: none"> <li>☐ Added command <a href="#">Set Relative Cursor Position</a></li> <li>☐ Added new shape drawing commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Draw Dot</a></li> <li>☐ <a href="#">Draw Dot at X/Y</a></li> <li>☐ <a href="#">Draw Styled Circle</a></li> <li>☐ <a href="#">Draw Ellipse</a></li> </ul> </li> <li>☐ Added advanced design attributes: <ul style="list-style-type: none"> <li>☐ <a href="#">Set Shadow Offset</a></li> <li>☐ <a href="#">Set Rectangle Corner Radius</a></li> </ul> </li> <li>☐ Patch: New property flags for <a href="#">Draw Rectangle</a></li> <li>☐ Patch: Shape shadows are now drawn as a complete shape instead of single lines</li> <li>☐ Fix: Commands could be interpreted before startup macro starts executing</li> <li>☐ Fix: Touch fields covering a complete rotated viewport didn't work</li> <li>☐ Fix: Touch calibrate and verify were executed in current viewport</li> <li>☐ Fix: <a href="#">Get # of Screens</a> could return wrong values on DPP-Cx1060A</li> <li>☐ Fix: <a href="#">Get Graphic Info</a> always returned index 0</li> <li>☐ Now supporting: <ul style="list-style-type: none"> <li>☐ 5" iLCD Panel DPP-Hx50</li> <li>☐ new 7" iLCD Panel DPP-x70</li> </ul> </li> </ul>
4.21	November 22, 2013	<ul style="list-style-type: none"> <li>☐ Patch: Optimized contrast settings for DPC3050 models</li> </ul>
4.20	November 4, 2013	<ul style="list-style-type: none"> <li>☐ Added filling commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Fill Display</a></li> <li>☐ <a href="#">Fill Display Area</a></li> <li>☐ <a href="#">Set Filling Color</a></li> <li>☐ <a href="#">Set Filling Gradient</a></li> <li>☐ <a href="#">Set Filling Tile</a></li> </ul> </li> <li>☐ Added adjustment commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Adjust Display</a></li> <li>☐ <a href="#">Adjust Display Area</a></li> <li>☐ <a href="#">Set Adjustment for Graphics</a></li> <li>☐ <a href="#">Set Brightness Adjustment</a></li> <li>☐ <a href="#">Set Contrast Adjustment</a></li> <li>☐ <a href="#">Set Hue Adjustment</a></li> <li>☐ <a href="#">Set Saturation Adjustment</a></li> </ul> </li> <li>☐ Added command <a href="#">Set Alpha</a></li> <li>☐ Patch: New property flags for <a href="#">Draw Rectangle</a> command (filling, no frame) <ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Erase Display</a> deactivated bold mode</li> <li>☐ Fix: Text output beyond the screen border could result in invalid cursor position when horizontal wrapping disabled</li> <li>☐ Fix: <a href="#">Paint Screen From</a> in orientations other than source was offset by one pixel</li> </ul> </li> </ul>
4.17	October 2, 2013	<ul style="list-style-type: none"> <li>☐ Added command <a href="#">Execute Protected Macro</a></li> <li>☐ Patch: <a href="#">Remove Touch Field</a> with parameter 0xFE removes all touch field in current viewport</li> </ul>
4.16	August 5, 2013	<ul style="list-style-type: none"> <li>☐ Patch: Nand Flash pages are relocated before read disturb issues could occur</li> </ul>

4.15	July 16, 2013	<ul style="list-style-type: none"> <li>☐ Fix: ADC ports 0 and 1 didn't work with active touch screen on some models</li> </ul>
4.14	June 13, 2013	<ul style="list-style-type: none"> <li>☐ Fix: 2D-encoded scanlines sent with wrong length byte in rare cases</li> </ul>
4.13	May 21, 2013	<ul style="list-style-type: none"> <li>☐ Now supporting new 7" iLCD Panel DPP-Cx8048A</li> <li>☐ Fix: Some Unicode characters were interpreted as escape sequences</li> </ul>
4.12	May 2, 2013	<ul style="list-style-type: none"> <li>☐ Fix: Unicode messages were output with double length</li> <li>☐ Fix: Characters wider than 32 pixels were not shown correctly</li> </ul>
4.11	April 30, 2013	<ul style="list-style-type: none"> <li>☐ Now supporting 2D run-length encoding</li> <li>☐ Added commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Write 1D/2D Run-Length Encoded Scan Line</a></li> <li>☐ <a href="#">Read 1D/2D Run-Length Encoded Scan Line</a></li> </ul> </li> </ul>
4.10	April 8, 2013	<ul style="list-style-type: none"> <li>☐ Faster graphic drawing due to reorganization of drawing routines</li> <li>☐ Now supporting incremental rotary encoder</li> <li>☐ Added command <a href="#">Enable/Disable Rotary Encoder Reporting</a></li> </ul>
4.05	March 19, 2013	<ul style="list-style-type: none"> <li>☐ Added command <a href="#">Enable/Disable Touch Field Reporting</a></li> <li>☐ Reduced power consumption by improving sleep mode behaviour when idling</li> </ul>
4.04	March 13, 2013	<ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Draw Touch Field Text Message</a> only worked with reported touch fields (key other than 0)</li> </ul>
4.02	February 7, 2013	<ul style="list-style-type: none"> <li>☐ Fix: <a href="#">Write Application Data to Flash</a> with the "no reboot" bit set crashed on DPC3090</li> </ul>
4.01	January 16, 2013	<ul style="list-style-type: none"> <li>☐ Fix: Preliminary models of DPP-CxP3224A had old hardware revision</li> </ul>
4.0	December 17, 2012	<ul style="list-style-type: none"> <li>☐ Now supporting <a href="#">Unicode fonts</a></li> <li>☐ Added commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Write Unicode Text</a></li> <li>☐ <a href="#">Get Unicode Text Extent</a></li> </ul> </li> <li>☐ Fix: Transparent animations showed flashes on DPP-CxS2440</li> </ul>
3.05	November 26, 2012	<ul style="list-style-type: none"> <li>☐ Added commands: <ul style="list-style-type: none"> <li>☐ <a href="#">Get Project Info</a></li> <li>☐ <a href="#">Get Graphic Info</a></li> <li>☐ <a href="#">Set Graphic Alignment</a></li> <li>☐ <a href="#">Display Graphic Area</a></li> <li>☐ <a href="#">Set Animation Background Color</a></li> <li>☐ <a href="#">Set Animation Background Frame</a></li> <li>☐ <a href="#">Set Animation Background Graphic</a></li> <li>☐ <a href="#">Set Animation Background Screen</a></li> <li>☐ <a href="#">Remove Animation Background</a></li> <li>☐ <a href="#">Move Animation To Frame</a></li> <li>☐ <a href="#">Enable/Disable Automatic Touch Macro Executing</a></li> </ul> </li> <li>☐ Don't reset offset, prefix and suffix when "Extras on Reset" option in project is set to "Keep"</li> <li>☐ Patch: Animation frame removal method "remove to background" now interpreted and displayed correctly</li> <li>☐ Fix: Scaled transparent graphics stretched background horizontally on DPC3050 controller</li> </ul>
3.03	October 12, 2012	<ul style="list-style-type: none"> <li>☐ Now supporting: <ul style="list-style-type: none"> <li>☐ new 10.2" iLCD Panel DPP-Cx1060A</li> <li>☐ new 3.5" iLCD Panel DPP-Cx3224A</li> </ul> </li> </ul>
3.01	July 4, 2012	<ul style="list-style-type: none"> <li>☐ Fix: All prefix and suffix weren't reset by <a href="#">Reset All</a></li> </ul>

3.00	July 2, 2012	<ul style="list-style-type: none"> <li>☞ Added new Composite USB Driver (HID / WinUSB)</li> </ul>
2.40	May 14, 2012	<ul style="list-style-type: none"> <li>☞ Now supporting:</li> <li>☞ new DPC3090 iLCD Controller</li> <li>☞ new 128 MB Flash Memory</li> </ul>
2.37	May 11, 2012	<ul style="list-style-type: none"> <li>☞ Added command <a href="#">Get Input Buffer Size</a></li> <li>☞ Modified USB product ID and string descriptor to distinguish between WinUSB and HID devices</li> </ul>
2.36	April 20, 2012	<ul style="list-style-type: none"> <li>☞ Fix: Inverting didn't work in rotated viewports</li> </ul>
2.35	April 18, 2012	<ul style="list-style-type: none"> <li>☞ Fix: DPC3050 keyboard reports on single column were always true</li> </ul>
2.34	April 11, 2012	<ul style="list-style-type: none"> <li>☞ Fix: Parameters for prefix and suffix commands were word values</li> </ul>
2.33	April 3, 2012	<ul style="list-style-type: none"> <li>☞ Fix: Macro Timer was not reset by <a href="#">Reset All</a></li> <li>☞ Fix: Macro call with subsequent commands didn't return anything if macro timer set</li> </ul>
2.32	March 27, 2012	<ul style="list-style-type: none"> <li>☞ Improved transmission speed by block-wise communication from iLCD to application</li> </ul>
2.31	March 23, 2012	<ul style="list-style-type: none"> <li>☞ Fix: Inverting didn't work correctly for DPP-CT2440</li> <li>☞ Fix: Font Spacing wasn't reset by <a href="#">Reset All</a></li> <li>☞ Fix: Bold attribute was persistent in Viewport structure</li> </ul>
2.30	January 18, 2012	<ul style="list-style-type: none"> <li>☞ Fix: Parameters for time/date commands were word values</li> </ul>
2.25	January 17, 2012	<ul style="list-style-type: none"> <li>☞ Fix: Baud rates &gt; 230400 lead to framing errors</li> </ul>
2.24	November 10, 2011	<ul style="list-style-type: none"> <li>☞ Added commands:</li> <li>☞ <a href="#">Set Backlight Intensity (High-Res)</a></li> <li>☞ <a href="#">Get Backlight Intensity (High-Res)</a></li> </ul>
2.23	October 24, 2011	<ul style="list-style-type: none"> <li>☞ Fix: Startup graphic wasn't displayed if in MicroSD card</li> </ul>
2.22	September 29, 2011	<ul style="list-style-type: none"> <li>☞ Fix: Ethernet lost connection while verifying after writing to flash memory</li> </ul>
2.21	August 30, 2011	<ul style="list-style-type: none"> <li>☞ Fix: SPI read replaced last byte with NULL</li> </ul>
2.20	June 14, 2011	<ul style="list-style-type: none"> <li>☞ Fix: Selecting viewport stored wrong attributes while animation running</li> </ul>
2.19	May 9, 2011	<ul style="list-style-type: none"> <li>☞ Patch: Offset, prefix and suffix for macros and messages assigned to touch fields now interpreted when activated, not when assigned</li> <li>☞ Fix: Executing touch macros via command didn't send response</li> </ul>
2.18	April 14, 2011	<ul style="list-style-type: none"> <li>☞ Transmission speed of TCP/IP increased</li> </ul>
2.17	March 24, 2011	<ul style="list-style-type: none"> <li>☞ Fix: System messages were drawn with current font and line style</li> </ul>
2.16	March 23, 2011	<ul style="list-style-type: none"> <li>☞ Fix: Rotated rectangles with line thickness &gt; 2 were drawn without rotation on DPC3050 controller</li> </ul>
2.15	March 18, 2011	<ul style="list-style-type: none"> <li>☞ Fix: Changing display orientation now relative to default orientation set in iLCD Manager XE</li> <li>☞ Fix: SPI lost one byte when quoting</li> </ul>
2.14	March 17, 2011	<ul style="list-style-type: none"> <li>☞ Fix: Touch fields on invisible screens could overlay a visible (=active) one</li> </ul>

2.13	March 16, 2011	<ul style="list-style-type: none"> <li>☞ Patch: Extended structure returned by <a href="#">Get Network Status</a></li> <li>☞ Fix: Writing to flash crashed for projects &gt; ~6.4 MB</li> <li>☞ Fix: Animations stored manually on MicroSD card weren't displayed correctly</li> </ul>
2.12	February 9, 2011	<ul style="list-style-type: none"> <li>☞ Added command <a href="#">Get Network Status</a></li> <li>☞ Fix: Error code 1035 was used twice, E_IO_TIMESET is now 1210</li> <li>☞ Fix: Range for hour parameter in time/date commands changed from 1..24 to 0..23</li> </ul>
2.11	December 29, 2010	<ul style="list-style-type: none"> <li>☞ Patch: Date and time now initialized to valid values for MicroSD card timestamps</li> </ul>
2.10	November 30, 2010	<ul style="list-style-type: none"> <li>☞ Added MicroSD card support for DPC3080 controller</li> </ul>
2.06	November 7, 2010	<ul style="list-style-type: none"> <li>☞ Now supporting new DPC3050 iLCD Controller</li> <li>☞ Faster initialization of TCP/IP</li> <li>☞ Improved transmission speed of screendumps</li> <li>☞ Patch: Copied screen areas now scaled with coordinate scale factors</li> <li>☞ Fix: Scale factors weren't reset by <a href="#">Reset All</a></li> </ul>
2.05	October 15, 2010	<ul style="list-style-type: none"> <li>☞ Fix: Getting free queue space on SPI returned 0 if space &gt; 256</li> </ul>
2.04	October 14, 2010	<ul style="list-style-type: none"> <li>☞ Patch: Free Queue Space on SPI was maximum 128 byte</li> </ul>
2.03	October 13, 2010	<ul style="list-style-type: none"> <li>☞ Improved transmission speed when writing to flash</li> </ul>

## iLCD Manager XE Version History

Version	Release Date	Description
4.0.9.5	April 29, 2021	<ul style="list-style-type: none"> <li>☞ First iLCD Manager XE release for supporting the new fast iLCD series based on the controller DPM5050.</li> <li>☞ Bugfix: iLCD Manager XE blocked when there is a lot of Console.print() activity.</li> </ul>
4.0.9.4	January 8, 2021	<ul style="list-style-type: none"> <li>☞ Bugfix: Project download fail for DPC3050.</li> </ul>
4.0.9.3	July 28, 2020	<ul style="list-style-type: none"> <li>☞ Bugfix: Conversion.floatToString.</li> <li>☞ Bugfix: iLCD Manager XE not responsive when no iLCD connected.</li> <li>☞ New Firmware Version 5.08.</li> </ul>
4.0.9.2	June 11, 2019	<ul style="list-style-type: none"> <li>☞ Java Debugger: Watch inspection improved.</li> <li>☞ Bugfix: Older version project load error.</li> <li>☞ Bugfix: Command and Completion List Popup crash.</li> </ul>
4.0.9.1	May 8, 2019	<ul style="list-style-type: none"> <li>☞ Taskbar icon bug removed.</li> </ul>
4.0.9.0	May 2, 2019	<ul style="list-style-type: none"> <li>☞ Rotary Encoder support added.</li> <li>☞ New command: <a href="#">Format MicroSD Card</a>.</li> <li>☞ DAC feature added: <a href="#">Set DAC Value</a>.</li> <li>☞ Help updated.</li> <li>☞ Fixed File Tab bugs.</li> <li>☞ New Firmware Version 5.07.</li> <li>☞ <a href="#">Release Note</a>.</li> </ul>

4.0.8.0	April 26, 2018	<ul style="list-style-type: none"> <li>☞ Event Handling added for USB and Ethernet.</li> <li>☞ Help updated.</li> <li>☞ New Firmware Version 5.06.</li> </ul>
4.0.7.0	March 19, 2018	<ul style="list-style-type: none"> <li>☞ Event Handling added for Comm interfaces (I2C, SPI, UART) and Keyboard.</li> <li>☞ Java API changed! See <a href="#">Release Note</a>.</li> <li>☞ Auto insert try-catch clause (activate with SPACE).</li> <li>☞ Show number of matches and wrap around when searching.</li> <li>☞ New Firmware Version 5.04.</li> </ul>
4.0.6.0	October 2, 2017	<ul style="list-style-type: none"> <li>☞ Description of Java Commands added to Help.</li> <li>☞ PDF Help updated (Java included).</li> <li>☞ Fixed minor bugs.</li> </ul>
4.0.5.0	September 5, 2017	<ul style="list-style-type: none"> <li>☞ Java Help finished (java.io package).</li> <li>☞ Show Watch Tooltips with Unicode.</li> <li>☞ Show Exception line in ExceptionWindow.</li> </ul>
4.0.4.0	July 12, 2017	<ul style="list-style-type: none"> <li>☞ Java Help Update:</li> <li>☞ java.lang, java.util, java.io.File, java.io.FileInputStream, java.io.FileOutputStream, java.io.RandomAccessFile</li> <li>☞ SPI communication (comm.SPI) feature added.</li> <li>☞ File IO classes File, FileInputStream, FileOutputStream RandomAccessFile implemented to allow access to Micro SD Card.</li> </ul>
4.0.3.0	May 24, 2017	<ul style="list-style-type: none"> <li>☞ "Find in Files" option added to find dialog (open with Ctrl+Shift+F).</li> <li>☞ Java Help updated.</li> <li>☞ "Add new empty Java App" menu item added to Java App toolbar (create without templates).</li> </ul>
4.0.2.0	April 28, 2017	<ul style="list-style-type: none"> <li>☞ Java Help updated.</li> <li>☞ Quick Search feature added: Double click a word and use F3 (Shift+F3).</li> <li>☞ New firmware version with minor bug fixes.</li> </ul>
4.0.1.0	April 4, 2017	<ul style="list-style-type: none"> <li>☞ Added console window to Java IDE.</li> <li>☞ Fixed minor bugs, improved usability of Java IDE.</li> </ul>
4.0.0.0	March 12, 2017	<ul style="list-style-type: none"> <li>☞ Now supporting creation and debugging of Java Apps.</li> <li>☞ New DPC3090 firmware V5.0 with integrated Java VM.</li> </ul>

**Package [ilcd](#)****Class [Touch](#)****Public Method [getPcapControllerInformation](#)**

```
String getPcapControllerInformation()
```

**Description:**

Returns	Description
PcapControllerInformation	PCAP information

The `getPcapControllerInformation()` method returns information regarding the PCAP of the module as a string.



Example string which is returned by this method: *Manuf.: FocalTech, ChipID: 85, FW: 3, FW-lib.: 48.3*

### Example

```
Draw.writeText(Touch.getPcapControllerInfo());
```

### See also:

[General.getDeviceInfo\(\)](#)

---

Copyright © demmel products gmbh. All rights reserved