Peripheral Input/Output Services

ADDON.MI_ND

Release 670

Copyright

Icons in Body Text

Icon	Meaning
⚠	Caution
💬	Example
💡	Note
🧭	Recommendation
SYN	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
`Example text`	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Peripheral Input/Output Services

PIOS Release Notes

Purpose

Release Notes only contain a summary of the new features or changes. The corresponding documentation contains detailed information.

Peripheral Input/Output Services (New)

Technical Data

Function is	New
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 SP09
Assignment to Application Component	CA-ME-SER Server-side
Country Setting	Valid for all countries

Use

PIOS provides an abstraction layer between the application and the peripheral.

Document Generator (new)

This is a new component of the SAP NetWeaver Development Studio.

Technical Data

Function is	New
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 SP09
Assignment to Application Component	CA-ME-NTV Native / OS specific drivers
Country Setting	Valid for all countries

Use

The Document Generator is a new MDK component that enables MI developers to specify peripheral requirements within a mobile application project. These requirements are stored on a file called Driver Requirements Document (DRD). The DRD identifies the peripheral types, as well as, the attributes within those peripheral types required by the application. In this first release, the Document Generator supports the printer peripheral type.

Emulator (new)

This is a new component of the SAP NetWeaver Developer Studio.

Technical Data

Function is	New
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 SP09
Assignment to Application Component	CA-ME-SER Server-side
Country Setting	Valid for all countries

Use

The Peripheral I/O Emulator enables the developer to simulate the usage and functionality of peripheral types supported under the PIOS architecture. In this first release, it provides emulation for the printer peripheral type.

Driver Selection Tool (new)

Technical Data

Function is	New
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 SP09
Assignment to Application Component	CA-ME-SER Server-side
Country Setting	Valid for all countries

Use

With release MI 2.5 SP09 the Driver Selection Tool (DST) is a new technical component of the SAP MI Web Console. Administrators need to deploy the appropriate peripheral add-on for mobile applications. The selection process can be very complex due to multiple factors, such as peripheral attributes, operating system, virtual machine, processor, and available transports. These must be taken into consideration when selecting the appropriate driver.

The DST allows the system administrator to easily identify the available peripheral add-ons that match all these requirements.

For more information, see Driver Selection Tool [Page 150] (*SAP Mobile Engine* → *SAP ME for Administrators* → *SAP MI Web Console* → *Driver Selection Tool*).

Effects on System Administration

A button that launches the DST is enabled in the Mobile Component page of the SAP MI Web Console.

Support Package Stack 12

Scanner API (New)

Technical Data

Function is	New
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 Support Package Stack 12
Assignment to Application Component	BC-MJC-CLN-API Mobile Public Interface
Country Setting	Valid for all countries

Use

With release MI 2.5 Support Package Stack 12 the Scanner API is a new technical component of the SAP MI Client API. It provides an abstraction layer between the application and the scanner peripheral type. The developer can access the scanner features provided by PIOS through the MI Client API.

For more information, see PIOS Scanner API Features Description [Page 44].

Document Generator (Changed)

Technical Data

Function is	Changed
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 Support Package Stack 12
Assignment to Application Component	BC-MJC-CLN-NTV Mobile Native / OS specific drivers
Country Setting	Valid for all countries

Use

With Release MI 2.5 Support Package Stack 12 support for the scanner peripheral type has been added to the Document Generator.

For more information about the scanner peripheral type see PIOS Scanner API Features Description [Page 44].

Peripheral Input/Output Emulator (Changed)

Technical Data

Function is	Changed
Release	**Software Component** • Component: SAP_ME_SERVER_COMPONENT • Release: MI 2.5 Support Package Stack 12
Assignment to Application Component	BC-MJC-SER Mobile Server-side
Country Setting	Valid for all countries

Use

With Release MI 2.5 Support Package Stack 12 the ability to emulate the scanner peripheral type has been added to the Emulator.

For more information about the scanner peripheral type see PIOS Scanner API Features Description [Page 44].

Support Package Stack 14

RFID API (New)

Technical Data

Function is	New
Release	**Software Component** • Component: MI_MDK • Release: MI 2.5 Support Package Stack 14
Assignment to Application Component	BC-MOB-CLN-NTV-RFI Mobile RFID
Country Setting	Valid for all countries

Use

With release MI 2.5 Support Package Stack 14 the RFID API is a new technical component of the SAP MI Client API. It provides an abstraction layer between the application and the RFID peripheral type. The developer can access the RFID features provided by PIOS through the MI Client API.

For more information, see PIOS RFID API Features Description [Page 72].

Document Generator (Changed)

Technical Data

Function is	Changed
Release	**Software Component** - Component: MI_MDK - Release: MI 2.5 Support Package Stack 14
Assignment to Application Component	BC-MOB-IDE-MDK Mobile Applications Development Kit
Country Setting	Valid for all countries

Use

With Release MI 2.5 Support Package Stack 14 support for the RFID peripheral type has been added to the Document Generator.

For more information about the RFID peripheral type, see PIOS RFID API Features Description [Page 72].

Peripheral Input/Output Emulator (Changed)

Technical Data

Function is	Changed
Release	**Software Component** - Component: MI_MDK - Release: MI 2.5 Support Package Stack 14
Assignment to Application Component	BC-MOB-IDE-MDK Mobile Applications Development Kit
Country Setting	Valid for all

Use

With release MI 2.5 Support Package Stack 14 the ability to emulate the RFID peripheral type has been added to the Emulator.

For more information about the RFID peripheral type, see PIOS RFID API Features Description [Page 72].

Peripheral Input/Output Services Architecture

Purpose

Services provided by the SAP Mobile Infrastructure to provide peripheral access to mobile applications. PIOS provides an abstraction layer between the application and the peripheral. A developer using PIOS does not have to be concerned about the implementation details of each peripheral model supported. Instead the developer will target abstracted functionality provided by the peripheral and required by the application.

Integration

The PIOS architecture has design components, runtime components and an administration component. The design time components are integrated in the SAP NetWeaver Developer Studio. Runtime components and the administration component are integrated in the SAP Mobile Client and the SAP Web Console respectively.

Design-time components:

MDK Peripheral Support Actions - SAP NetWeaver Developer Studio toolbar components used to define and emulate the peripheral features required by an MI application.

- **Create/Modify Driver Requirements Document -** Launches a wizard to specify the mobile application peripheral requirements.
- **Set MDK Peripheral Emulation Mode -** Configures the SAP NetWeaver Developer Studio to run the application peripheral actions/requests in emulation mode.
- **Launch the Peripheral I/O Emulator -** Launches the Peripheral I/O Emulator (described below).

Peripheral Emulator - Emulates peripheral functionality according to settings determined by the developer.

Run-time components:

PIOS Classes in the SAP MI Client - Used to discover, connect, and use functionality available in supported peripheral types.

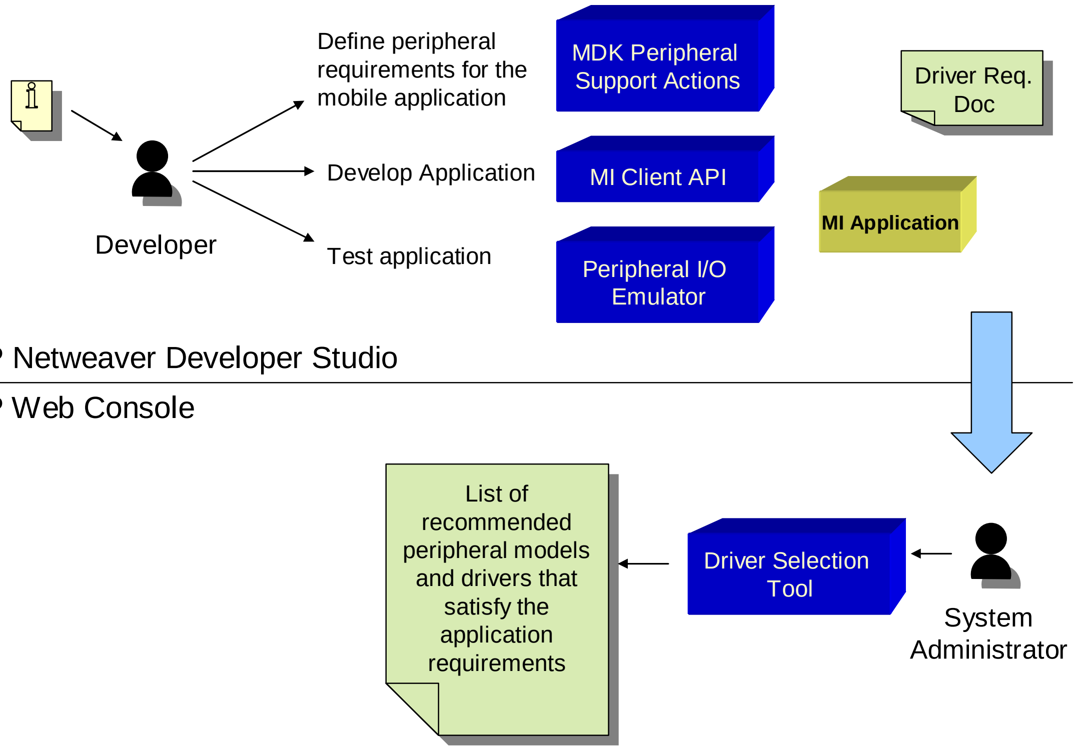
Driver Add-on - Component containing the software modules required to make use of a specific peripheral model on a client device.

Administrative component:

Driver Selection Tool in the SAP Web Console - Identifies available peripheral drivers that match the peripheral requirements of a mobile application (as specified in the Driver Requirements Document).

Example

The following diagram provides an overview of the PIOS architecture:


Define peripheral requirements for the mobile application
MDK Peripheral Support Actions
Driver Req. Doc
Developer
Develop Application
MI Client API
MI Application
Test application
Peripheral I/O Emulator
SAP Netweaver Developer Studio
SAP Web Console
List of recommended peripheral models and drivers that satisfy the application requirements
Driver Selection Tool
System Administrator

See Also

For more information about the Peripheral Input/Output Services Architecture and answers to frequently asked questions, refer to SAP Note 853397.

PIOS Getting Started

Use

The procedure explained below provides a quick walkthrough of the PIOS Architecture. The process will start by using the Client API to do a simple example, setting the development environment to test the application in the emulator, and finally the application will run in the emulator and the printer.

Prerequisites

- SAP Mobile Infrastructure installed.
- SAP NetWeaver Developer Studio installed and configured.
  - Complete MDK configuration.
    - MI login user name
    - MI login password
- MI Client installed and successfully synchronized.

Procedure

1. Open the SAP NetWeaver Developer Studio.
2. Create a new MI Project by pressing the *MDK: Create a new MI project* button. Select the *JSP Project* radio button and fill out all the required fields (*Project Name*, *Project root folder*, and so on). Leave the *JSP name* field with the dafult value (initial.jsp). Press *Finish*.

   If a browser windows with the XML shows up, close it.

3. Select the project you just created on the *MI Projects* list and then click on *Create/Modify Driver Requirements Document*.
4. Select *Printer* and then click *Add* → *Finish*. This will open the **Driver Requirements Document Editor.**
5. On the *Driver Requirements Document Editor* check the boxes for *Bitmapped fonts* and *Scalable fonts*. Click on *Save*.
6. Completely replace the code in the *initial.jsp*, whose tab is at the top of the DRD Editor, file with the code below:

```
<%@ page import="com.sap.ip.me.api.pios.connection.*" %>
<%@ page import="com.sap.ip.me.api.pios.printer.*" %>

<html>
<head>

<!--  You can place a image here to return to ME home page: 127.0.0.1
= localhost on the Desktop and also works on the PDA

<a href="http://127.0.0.1:4444/me" > <img src="mimes/myGIF.gif"
alt="ME Home" ></a>
 -->

</head>
<body>

  <jsp:useBean id="servletToJSPBean" scope="session"
                     class="miProjectPackage.bean.dataBean" />

<!-- For event handling we need a HTML "form" command -->
 <form method="post" action="start" id="myForm1" name="form">

<!-- Display title of the example -->
   <h4>Hello World Example</h4><br>

   Look at the Emulator.

 </form>

<%
      GraphicPrinter gP = null;
      try{

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
                conn.listDrivers(ConnectionType.PRINTER);

         PrinterParameters params =
                new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);

         gP = (GraphicPrinter) conn.open(params);

         String[] sFonts =
               gP.getFontConfigurationManager().listFontNames();

         PrinterFont pF = gP.getFont(sFonts[0]);

         gP.drawText(pF, 10, 10, "Hello World!", 0);

         gP.doPrint(1);

         gP.close();

      }
      catch (Throwable error){
         error.printStackTrace();
      }
      finally {
         try {
            gP.close();
         }
         catch (Exception ex){
         }
      }
%>
</body>
</html>
```

7. Save the file by clicking on *Save*.
8. Export the project by pressing the *MDK: Export a project into a MI archive (.war/.jar)*. Select the project from the *MI Projects* list and hit the *Next* button. Write `/initial.jsp` on *URL pattern*.
9. Upload the newly created .war file to the SAP Web Console. Be sure to assign the same name to the application as the name of the .war file itself.
10. Assign the .war file to your MI Client.
11. Start the MI Client and log on to it. Synchronize the MI Client to download the application (.war file). Restart the MI Client.
12. Set on the emulation mode by clicking on *Set MDK Peripheral Emulation Mode*.
13. Launch the Emulator by clicking on *Launch the Peripheral I/O Emulator*.
14. Set the emulator Offline by clicking *On/Offline*, and then click on Capability… In the *General* tab, make sure that *Page Measurements* is not checked. Select the *Media* tab and on *Type* select *Continuous*. When the *Setup Reminder* dialog box comes up, select *Yes*.

    If the Setup Reminder dialog box does not come up, click on *Setup...*

15. In the Printer Setup window, under the *Media* tab, select *Continuous_4* and click *OK*.
16. Run the application by clicking on *MDK: Export a project into a MI archive (.war/.jar)*. Then click on *Next* and select the checkboxes for *Run the mobile application* and *Uppercase for MI archive name*, in the *URL pattern* field write `/initial.jsp`.

    To run the application again, choose *Run* → *Run…* Select the project under *Configurations* and click on *Run*. (The *Package Explorer* tab must be selected.)

17. Click on *Finish*. The message “Hello World!” is displayed in the Emulator.

Alternate scenario: printing using the Windows driver:

1. Install the connector and piprmswin32 driver add-ons to the SAP Web Console.

- **For Windows 2000,** the files are
    - connector_w2k_x86_jvm_1_1_0
    - piprmswin32_w2k_x86_jvm_1_1_0
- **For Windows XP,** the files are
    - connector_wxp_x86_jvm_1_1_0
    - piprmswin32_wxp_x86_jvm_1_1_0

2. Perform a synchronization to install the Driver Add-Ons.
3. Set the emulation mode off by clicking on *Set MDK Peripheral Emulation Mode*.
4. Run the application by clicking on *MDK: Export a project into a MI archive (.war/.jar)*. Then click on *Next* and select the checkboxes for *Run the mobile application* and *Uppercase for MI archive name*, in the *URL pattern* field write `/initial.jsp`.
5. Click on *Finish*. “Hello Word” is printed in the default printer configured.

PIOS API Core

Purpose

The **PIOS API Core** includes classes used to discover, connect and use functionality available across different peripheral types. It provides an abstraction layer between the application and the peripheral. A developer using PIOS does not have to be concerned about the implementation details of each peripheral model supported. Instead the developer will target abstracted functionality provided by the peripheral and required by the application.

Implementation Considerations

The abstraction provided by the PIOS API also extends to provide peripheral support across several platforms (OS, JVMs, processors).

Integration

The PIOS API Core is part of the MI Client API.

Features

The PIOS API Core contains common functionality available across peripheral types. For example, it includes methods to:

- discover peripheral drivers installed on the device
- open a connection to a peripheral
- configure the driver

Refer to the MI Client API javadocs for detailed method description.

Constraints

- Peripheral models may offer additional functionality not supported by the PIOS API.

Printer API

Purpose

Printer support in the SAP Mobile Infrastructure is offered as part of the Peripheral Input/Output Services (PIOS). The Printer API provides an abstraction layer between the intricacies of different printer drivers and the application.

Integration

The Printer API is part of the MI Client API.

Features

- Refer to the Printer API javadocs for detailed method description.
- Refer to Features Description [Page 18].

Constraints

- Printer models may offer additional features not supported by this API.
- Supported functionality varies according to printer make and model.

PIOS Printer API Features Description

Definition

PIOS Printer API provides support for several features. These features vary depending on the printer make and model. The table below provides a description for these features:

Printable Objects

Printable Object	**Description**
Bitmap Image	Images in bitmap format.
PCX Image	Images in PCX format.
Scalable Fonts	Fonts based on Vector graphics and scale without degradation.
Bitmap Fonts	Fonts made of character images and may present some degradation when scaled up.
Barcode	Barcode printing.

Printer Features

Printer Feature	**Description**
Clear Error	Clear error condition from the printer.
Advance Forward	Advances the paper forward.
Advance Backward	Retracts the paper.
Dispose	Dispose of the commands sent to the printer. (No information is printed.)
Get Status	Queries the printer status.
Page measurements	Page length and width given in points.
Printer Head width	Obtain the printer head width (in points).
Text metrics	Obtain the text dimension (in points) based on the font.
Barcode metrics	Obtain the barcode dimension (in points).
Image metrics	Obtain the image dimension (in points).
Load Image	Upload an image into the printer's memory.
Delete Image	Delete an image from the printer's memory.
Get DPI (Dot per inch)	Returns the printer's resolution.
Transport Configuration	Modify and set the default transport (i.e. Serial, Bluetooth) options.
Font Configuration	Configure (add, delete, change) font information.
Send Raw Data	Sends raw bytes to the printer. This option sends the bytes (information) submitted directly to the printer, without any intervention from the API. ⚠ This feature should be used with caution. The printer may exhibit unexpected behavior if invalid data is sent to it.

Graphic Mode Printing

Graphic Mode Feature	**Description**
Barcode Rotation	Barcode rotation in 90º increments. Barcodes can be rotated to 90º, 180º, and 270º.
Image Rotation	Image rotation in 90º increments. Images can be rotated to 90º, 180º, and 270º.
Text Rotation	Text rotation in 90º increments. Text can be rotated to 90º, 180º, and 270º.
Line thickness	Set the line thickness. (in points)
Draw Text	Print text.
Page measurements	Page length and width given in points (for non-continues paper).
Draw Barcode	Print barcodes.
Draw Line	Print lines.
Draw Rectangle	Print rectangles.
Draw Image	Print images.

Line Mode Printing

Line Mode Feature	Description
Barcode Alignment	Align a barcode in a line (center, right, left).
Image Alignment	Align an image in a line (center, right, left).
Text Alignment	Align text in a line (center, right, left).
Print Text	Print text in a line.
Print Barcode	Print barcodes in a line.
Print Image	Print images in a line.
Set Line Spacing	Defines the space in points between two lines (in points).

Symbology

Symbology	Description
Codabar	Support for Codabar barcode symbology.
Code 39	Support for Code 39 barcode symbology.
Code 128	Support for Code 128 barcode symbology.
EAN-8	Support for EAN-8 barcode symbology.
EAN-13	Support for EAN-13 barcode symbology.
Interleaved 2 of 5	Support for Interleaved 2 of 5 barcode symbology.
PDF417	Support for PDF-417 (Portable Data Format) 2D barcode symbology.
UCC/EAN-128	Support for UCC/EAN barcode symbology.
UPC-A	Support for Universal Product Code (UPC) barcode symbology.

PIOS Printer API Guidelines

Definition

This document explains several guidelines for the PIOS Printer API of the SAP NetWeaver Developer Studio. Each of the guidelines is discussed below.

Use

These guidelines are intended for all the developers working on the SAP NetWeaver Developer Studio in a mobile application with printer peripheral requirements. It helps the developer get the most out of the PIOS infrastructure. The developer can access these features provided by PIOS through the MI Client API.

1. Printer Connection

Since multiple applications can be running in a mobile device at a given time it is recommended that mobile applications should open the connection to the printer when they are ready to send data to the printer and close the connection when they have finished. This prevents other applications from having to wait for the first one to close before they can open a connection. Otherwise if a mobile application opens the printer connection when it starts and does not close it until it finishes, no other application may use the printer until the first one finishes. Any other mobile application, that tries to open a connection to the scanner, will receive an error message.

2. **DPI Awareness**

DPI stands for “Dots Per Inch”, and it is the standard unit of measure used for printer resolution. Different printer models may have different resolutions. A developer should be aware of this fact while writing an application that works with more than one printer model. If the application prints an image, the size of the printed image may vary depending on the printer resolution. For instance, an image printed with a resolution of 300 dpi may be twice as large as an image printed with a 600 dpi resolution. Developers should write applications handling possible DPIs.

3. **Printer Head Width**

Printer head width information, returned in points, should be used by MI developers to avoid hard-coding this information into mobile applications. Doing so will limit the capabilities to run the application on multiple printer models.

4. **Page Size Awareness**

If an application is written for a printer that uses the *Page measurements* attribute (only supported for non-continuous paper printers), it can take advantage of this feature and position the fields to be printed according to the paper size. The term “fields” as used in this paragraph refers to text, lines, images, and barcodes.

5. **Use of Field Metrics**

Given the fact that printers vary in size, developers should make use of the field and page metrics to position images, barcodes, and graphics on the printed page.

6. Image Handling

Since loading images is time consuming, images should be loaded during the application installation cycle. In other words, the loading and deletion of images should not occur continuously throughout the execution of the mobile application.

7. Font Mapping

The API has virtual fonts that map to physical fonts in the printer. Virtual fonts can be added, removed, or modified. It is important to remember that if a font (virtual) is added in the API, then its counterpart, the physical font, must be added to the printer as well. Also a new virtual font can be set to be a copy of an existing font with modified attributes. The font name parameter must exist in the printer.

8. Scalable and Bitmapped Fonts

It is recommended to use scalable fonts over bitmapped fonts whenever possible (not all printers support both font types). Scalable fonts make better use of higher resolutions because they are based on vector graphics, and are more easily scaled than bitmapped. Bitmapped fonts may make their pixels evident when their size is enlarged.

9. Printing Copies

Printing several copies of the same data should be done by providing the desired number of copies to the doPrint() method. There is no need to go through another print cycle, sending the same data for a copy.

10. Un-buffered Methods

These are methods that are directly sent to the printer. They do not pass through the printer buffer and therefore cannot be cancelled. The un-buffered methods are clearError(), getStatus(), and advance().

> The advance method must be used only to position the print head to a specific position on the paper. To move forward in line mode developers should use the printText() method using an empty string.

11. Raw Bytes

A send raw bytes method exists but its use is not encouraged as it ties applications to specific printer models. The resulting behavior, for other printer models, cannot be predicted.

Printer API Examples

This section contains Printer API examples.

Print Text on the Left Paper

In this example the Printer is in **Line** mode and a line is printed with left alignment. The program performs the following steps:

1. Opens the connection to the printer.
2. Prints the sample text with left alignment.
3. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TextLeftLineMode {

   public static void main(String[] args) {

      LinePrinter lP = null;

      try{

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
               conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
               new PrinterParameters(driverInfo[0]);
         params.setPrinterMode(PrinterParameters.LINE_MODE);

         //----------------------(1)--------------------

         lP = (LinePrinter) conn.open(params);

         String[] sFonts =
              lP.getFontConfigurationManager().listFontNames();

         PrinterFont pF = lP.getFont(sFonts[0]);

         lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);

         //----------------------(2)--------------------

         lP.printText(pF,"Left side of the line.",
             LinePrinter.ALIGN_LEFT);

         lP.doPrint(1);

      }

      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(3)--------------------
      try {
         lP.close();
      } catch (Exception ex) {}
   }
      return;
   }
}
```

Print Text Center Alignment in Line Mode

This example demonstrates text aligned in the center of the line with the printer in **Line** mode. The program does the following:

1. Opens the connection to the printer.
2. Prints the text with center alignment.
3. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TextCenterLineMode {

   public static void main(String[] args) {

      LinePrinter lP = null;

      try{

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
             conn.listDrivers(ConnectionType.PRINTER);

         PrinterParameters params =
             new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.LINE_MODE);

         //----------------------(1)--------------------

         lP = (LinePrinter) conn.open(params);
         String[] sFonts =
             lP.getFontConfigurationManager().listFontNames();
         PrinterFont pF = lP.getFont(sFonts[0]);
         lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);

         //----------------------(2)--------------------

         lP.printText(pF, "Center of the line.",
                 LinePrinter.ALIGN_CENTER);
         lP.doPrint(1);

      }

      catch (Throwable error){
         error.printStackTrace();
      } finally {

         //----------------------(3)--------------------

         try {

            lP.close();

         } catch (Exception ex) {}

      }

      return;

   }

}
```

Print Text on the Right Side

In this example the printer is in **Line** mode and the text is printed on the right side of the page. This program follows the steps below:

1. Opens the connection to the printer.
2. Prints the text with right alignment.
3. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TextRightLineMode {

   public static void main(String[] args) {

      LinePrinter lP = null;

      try{

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
               conn.listDrivers(ConnectionType.PRINTER);

         PrinterParameters params =
               new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.LINE_MODE);

         //----------------------(1)--------------------

         lP = (LinePrinter) conn.open(params);

         String[] sFonts =
             lP.getFontConfigurationManager().listFontNames();

         PrinterFont pF = lP.getFont(sFonts[0]);

         lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);

         //----------------------(2)--------------------

         lP.printText(pF, "Right side of the line.  ",
                          LinePrinter.ALIGN_RIGHT);

         lP.doPrint(1);

      }
      catch (Throwable error){

         error.printStackTrace();

      } finally {

         //----------------------(3)--------------------

         try {
            lP.close();

         } catch (Exception ex) {}
   }
      return;
   }

}
```

Text Drawn on the Left Side of the Page

This example draws the text on the left of the page. The printer is in **Graphic** mode. This program follows the steps below:

1. Opens the connection to the printer.
2. Calculates the bottom of the page using the page and font metrics.
3. Draws the text on the bottom left corner of the page.
4. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TextLeftGraphicMode {

   public static void main(String[] args) {

      float xLeft = 0, yBottom = 0;
      String sText="Text in the left bottom corner of the page.";
      GraphicPrinter gP = null;

      try{

         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
            conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
            new PrinterParameters(driverInfo[0]);
         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);
         //----------------------(1)--------------------

         gP = (GraphicPrinter) conn.open(params);
         String[] sFonts =
            gP.getFontConfigurationManager().listFontNames();
         PrinterFont pF = gP.getFont(sFonts[0]);

         //----------------------(2)--------------------

         yBottom = gP.getPageMetrics().getHeight() -
                   pF.getMetrics(sText).getHeight() - 10;

         //----------------------(3)--------------------

         gP.drawText(pF, xLeft, yBottom, sText,
                     GraphicPrinter.NO_ROTATION);
         gP.doPrint(1);

      }
      catch (Throwable error){
         error.printStackTrace();

      } finally {

         //----------------------(4)--------------------
         try {

            gP.close();

         } catch (Exception ex) {}
      }
      return;
   }
}
```

Draw Text in the Center of the Page in Graphic Mode

This example demonstrates a line printed in the center of the page with the printer in **Graphic** mode. The program does as explained below:

1. Opens connection to the printer
2. Using the page and font metrics, calculates the coordinates for the text to be in the center of the page.
3. Draws the text on the center of the page.
4. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TextCenterGraphicMode {

   public static void main(String[] args) {

      float xCenter=0, yCenter=0;
      String sText="This text is on the center.";
      GraphicPrinter gP = null;
      try{

         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
                    conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
                    new PrinterParameters(driverInfo[0]);
         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);
         //----------------------(1)--------------------

         gP = (GraphicPrinter) conn.open(params);
         String[] sFonts =
             gP.getFontConfigurationManager().listFontNames();
         //----------------------(2)--------------------

         PrinterFont pF = gP.getFont(sFonts[0]);
         yCenter = (gP.getPageMetrics().getHeight()/2) -
                     pF.getMetrics(sText).getHeight()/2);
         Metrics metrics = pF.getMetrics(sText);
         float pageWidth = gP.getPageMetrics().getWidth();
         xCenter = (pageWidth/2) - (metrics.getWidth()/2);
         //----------------------(3)--------------------

         gP.drawText(pF, xCenter, yCenter, sText,
                      GraphicPrinter.NO_ROTATION);
         gP.doPrint(1);
      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(4)--------------------
         try {
            gP.close();
         } catch (Exception ex) {}

      }
      return;
   }
}
```

Text Drawn on the Right Side of the Page

In this example the code draws text on the right side of the page. The printer is in **Graphic** mode. The program does as follows:

1. Opens the connection to the printer.
2. Uses page and font metrics to calculate the coordinates for the text to print on the right.
3. Draws the text at the calculated coordinates.
4. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TextRightGraphicMode {

   public static void main(String[] args) {

      float xRight=0, y=15;
      String sText="This text is on the right.";

      GraphicPrinter gP = null;

      try{
         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
                      conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
                      new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);

         //----------------------(1)--------------------

         gP = (GraphicPrinter) conn.open(params);
         String[] sFonts =
               gP.getFontConfigurationManager().listFontNames();
         PrinterFont pF = gP.getFont(sFonts[0]);
         //----------------------(2)--------------------

         xRight = gP.getPageMetrics().getWidth() -
                  pF.getMetrics(sText).getWidth() - 10;
         //----------------------(3)--------------------

         gP.drawText(pF, xRight, y, sText,
                     GraphicPrinter.NO_ROTATION);

         gP.doPrint(1);
      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(4)--------------------
         try {
            gP.close();
         } catch (Exception ex) {}
   }

      return;

   }
}
```

Two Page Report

This small program prints a two page report. The printer is in **Graphic** mode. This program does the following:

1. Opens connection to the printer.
2. Draws the first page with a sample text.
3. Draws the second page with another sample text.
4. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class TwoPageReport {

   public static void main(String[] args) {

      int x=15, y=17;
      GraphicPrinter gP = null;

      try{

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
             conn.listDrivers(ConnectionType.PRINTER);

         PrinterParameters params =
             new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);

         //----------------------(1)--------------------

         gP = (GraphicPrinter) conn.open(params);

         String[] sFonts =
              gP.getFontConfigurationManager().listFontNames();

         PrinterFont pF = gP.getFont(sFonts[0]);

         //----------------------(2)--------------------

         gP.drawText(pF, x, y, "First Page", 0);
         gP.doPrint(1);

         //----------------------(3)--------------------

         gP.drawText(pF, x, y, "Second Page", 0);
         gP.doPrint(1);

      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {

         //----------------------(4)--------------------

         try {

            gP.close();

         } catch (Exception ex) {}
      }
      return;
   }
}
```

Image Printing with Printer in Line Mode

This example demonstrates how to print an image with the printer in **Line** mode. In this case there is a bitmap image file called “test.bmp”, which is stored in “C:\TEMP\”. This program does as explained below:

1. Lists the drivers.
2. Opens the image file.
3. Opens connection to the printer.
4. Loads the image into the printer.
5. Prints the image.
6. Closes connection to the printer.

```
package test;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class ImageLineMode {

   public static void main(String[] args) {

      LinePrinter lP = null;
      PrinterImage image = null;

      try {

         Connector conn = Connector.getInstance();

         //----------------------(1)--------------------

         DriverInfo[] driverInfo =
             conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
             new PrinterParameters(driverInfo[0]);
         params.setPrinterMode(PrinterParameters.LINE_MODE);
         try {

            //-------------------(2)--------------------

            File testImage = new File("C:\\TEMP\\test.bmp");
            if (testImage.exists()) {
               BufferedInputStream bfis = null;
               try {
                  bfis = new BufferedInputStream(
                              new FileInputStream(testImage));
                  int bufferSize = (int)testImage.length();
                  byte[] imageBytes = new byte[bufferSize];
                  int i = bfis.read(imageBytes, 0, bufferSize);

                  //-------------(3)--------------------

                  lP = (LinePrinter) conn.open(params);

                  if (i > 0) {

                     image = lP.createImage("test",
                            PrinterImage.IMAGE_BMP, imageBytes);
                     //----------(4)--------------------
                     lP.loadImage(image);
                  }
               }
               finally {
                  if (bfis != null) bfis.close();
               }
            }
         }
         catch (Throwable tFile) {
            tFile.printStackTrace();
         }
         String[] sFonts =
              lP.getFontConfigurationManager().listFontNames();
         PrinterFont pF = lP.getFont(sFonts[0]);
         lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);
         //----------------------(5)--------------------
         lP.printImage(image.getName(), LinePrinter.ALIGN_CENTER);
         lP.doPrint(1);

      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(6)--------------------
         try {
            lP.close();
         } catch (Exception ex) {}
      }
      return;
   }
}
```

Printing a Barcode with Printer in Line Mode

In this code a barcode is printed using Code 39 symbology. The "Full ASCII" and "Check Digit mod43" options are set for the symbology. This code performs the following:

1. Opens connection to the printer.
2. Creates a barcode with Human Readable Below.
3. Sets the density for the barcode to the default.
4. Sets the height of the barcode to 10.
5. Sets the Scale of the barcode to Double.
6. Encodes the data used, 10101, using ASCII.
7. Sends barcode to the printer.
8. Closes connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;
import com.sap.ip.me.api.pios.symbology.*;

public class BarcodeLineMode {

   public static void main(String[] args) {

     LinePrinter lP = null;

     try{

         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
                    conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
                    new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.LINE_MODE);

         Symbology symbology = new Code39();

         symbology.setOptions(Code39.FULLASCII |
                              Code39.CHECK_DIGIT_MOD43);

         //----------------------(1)--------------------

         lP = (LinePrinter) conn.open(params);

         String[] sFonts =
                  lP.getFontConfigurationManager().listFontNames();

         PrinterFont pF = lP.getFont(sFonts[0]);

         //----------------------(2)--------------------

         PrinterBarcode barcode = lP.createBarcode(symbology,
                    PrinterBarcode.HUMAN_READABLE_BELOW);

         //----------------------(3)--------------------

         barcode.setDensity(PrinterBarcode.DENSITY_DEFAULT);

         //----------------------(4)--------------------

         barcode.setHeight(10);
         //----------------------(5)--------------------
         barcode.setScaleFactor(PrinterBarcode.SCALE_DOUBLE);
         //----------------------(6)--------------------
         byte[] data = "10101".getBytes("ASCII");
         //----------------------(7)--------------------
         lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);
         lP.printBarcode(barcode, data, LinePrinter.ALIGN_CENTER);

         lP.doPrint(1);
      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(8)--------------------
         try {
            lP.close();
         } catch (Exception ex) {}
      }
      return;
   }
}
```

Text Rotation

This example demonstrates text rotation. The printer is in **Graphic** mode. This program follows the steps below:

1. Opens the connection to the printer.
2. Gets the text metrics, width and height.
3. Draws the text with a rotation of 180 degrees.
4. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class RotateText {

   public static void main(String[] args) {

      float x=0, y=0;
      String sText="This is a line of text.";
      GraphicPrinter gP = null;

      try{

         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
                      conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
                      new PrinterParameters(driverInfo[0]);
         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);

         //----------------------(1)--------------------

         gP = (GraphicPrinter) conn.open(params);
         String[] sFonts =
             gP.getFontConfigurationManager().listFontNames();
         PrinterFont pF = gP.getFont(sFonts[0]);

         //----------------------(2)--------------------

         x = pF.getMetrics(sText).getWidth() + 10;
         y = pF.getHeight() + 15;

         //----------------------(3)--------------------

         gP.drawText(pF, x, y,
                     sText,GraphicPrinter.ROTATE_180_DEGREES);

         gP.doPrint(1);

      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(4)--------------------
         try {
            gP.close();
         } catch (Exception ex) {}
   }
      return;
   }
}
```

Rotate an Image

This example below shows demonstrates rotating an image using the PIOS Client API. In this case there is a bitmap image file called “test.bmp”, which is stored in “C:\TEMP\”. The printer is **Graphic** mode. This program follows the steps below:

1. Opens the image file.
2. Opens the connection to the printer.
3. Loads the image to the printer's memory.
4. Gets the image height.
5. Draws the image with a 90 degree rotation.
6. Closes the connection to the printer.

```
package test;

import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;

public class RotateImage {

   public static void main(String[] args) {

      GraphicPrinter gP = null;
      PrinterImage image = null;
      float x = 0, y = 15;

      try {

         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
              conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
              new PrinterParameters(driverInfo[0]);
         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);

         try {

            //-------------------(1)--------------------
            File testImage = new File("C:\\TEMP\\test.bmp");
            if (testImage.exists()) {
               BufferedInputStream bfis = null;

               try {
                  bfis = new BufferedInputStream
                     (new FileInputStream(testImage));
                  int bufferSize = (int)testImage.length();
                  byte[] imageBytes = new byte[bufferSize];
                  int i = bfis.read(imageBytes, 0, bufferSize);

                  //-------------(2)--------------------

                  gP = (GraphicPrinter) conn.open(params);

                  if (i > 0) {
                     image = gP.createImage("test",
                            PrinterImage.IMAGE_BMP, imageBytes);
                     //----------(3)--------------------
                     gP.loadImage(image);
                  }
               }
               finally {
                  if (bfis != null) bfis.close();
               }
            }
         }
         catch (Throwable tFile) {
            tFile.printStackTrace();
         }

         //----------------------(4)--------------------
         x = image.getMetrics().getHeight() + 10;

         //----------------------(5)--------------------
         gP.drawImage(image.getName(), x, y,
                      GraphicPrinter.ROTATE_90_DEGREES);
         gP.doPrint(1);
      }
      catch (Throwable e){
         e.printStackTrace();
      } finally {
         //----------------------(6)--------------------
         try {
            gP.close();
         } catch (Exception ex) {}
      }
      return;
   }
}
```

Rotating a Barcode

This example demonstrates how to rotate a barcode. Rotation of a barcode is only possible with the printer in **Graphic** mode. This program does the following:

1. Sets the symbology options for Full ASCII and Check Digit Mod43.
2. Opens the connection to the printer.
3. Sets for the barcode: Human Readable Below, Default Density, Double Scale.
4. Encodes the data, 32123, using ASCII.
5. Gets the metrics for the barcode.
6. Draws the barcode rotating it 270 degrees.
7. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;
import com.sap.ip.me.api.pios.symbology.*;

public class RotateBarcode {

   public static void main(String[] args) {

      float x=0, y=0;
      GraphicPrinter gP = null;
      try{

         Connector conn = Connector.getInstance();
         DriverInfo[] driverInfo =
              conn.listDrivers(ConnectionType.PRINTER);

         PrinterParameters params =
              new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.GRAPHIC_MODE);

         Symbology symbology = new Code39();

         //----------------------(1)--------------------

         symbology.setOptions(Code39.FULLASCII |
                              Code39.CHECK_DIGIT_MOD43);

         //----------------------(2)--------------------

         gP = (GraphicPrinter) conn.open(params);

         //----------------------(3)--------------------

         PrinterBarcode barcode = gP.createBarcode(symbology,
                 PrinterBarcode.HUMAN_READABLE_BELOW);

         barcode.setDensity(PrinterBarcode.DENSITY_DEFAULT);
         barcode.setHeight(10);
         barcode.setScaleFactor(PrinterBarcode.SCALE_DOUBLE);

         //----------------------(4)--------------------

         byte[] data = "32123".getBytes("ASCII");
         //----------------------(5)--------------------
         x = barcode.getHeight() + 10;
         y = barcode.getMetrics(data).getWidth() + 15;
                  //----------------------(6)--------------------
         gP.drawBarcode(barcode, x, y,
                        data,GraphicPrinter.ROTATE_270_DEGREES);

         gP.doPrint(1);
      }
      catch (Throwable error){
         error.printStackTrace();
      } finally {
         //----------------------(7)--------------------
         try {
            gP.close();
         } catch (Exception ex) {}
      }
      return;
   }
}
```

Configure the Printer Parameters

This example demonstrates how to configure a printer parameter in the piprmswin32 driver. The parameter to display a printer selection pop-up window is set to true, used, and then set to false. The printer is in **Line** mode. This program follows the steps mentioned below:

1. Opens connection to the printer.
2. Gets the configuration manager for the driver.
3. Using the configuration manager gets the parameter of the driver.
4. Sets the value of the parameter to true.
5. Closes and reopens connection to the printer for the change to the parameter value to take effect.
6. Prints some sample text.
7. Finally, sets the parameter back to false.
8. Closes the connection to the printer.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;
import com.sap.ip.me.api.pios.configuration.*;

public class ConfigurePrinterParameters {

   public static void main(String[] args) {

       LinePrinter lP = null;

       try {

         //----------------------(1)--------------------

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
                conn.listDrivers(ConnectionType.PRINTER);

         PrinterParameters params =
                new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.LINE_MODE);

         lP = (LinePrinter) conn.open(params);

         //----------------------(2)--------------------

         DriverConfigurationManager manager =
           lP.getParameters().getDriver().getConfigurationManager();

         //----------------------(3)--------------------

         Configuration config1 =
                  manager.getConfiguration("DriverParameters");

         //----------------------(4)--------------------

         String promptUser = config1.getParameters()[0];

         config1.setParameterValue(promptUser, "true");

         manager.save();
         //----------------------(5)--------------------
         lP.close();
         lP = (LinePrinter) conn.open(params);
         //-------------------(6)--------------------
         String[] sFonts =
             lP.getFontConfigurationManager().listFontNames();
         PrinterFont pF = null;
         if (sFonts != null) {
            pF = lP.getFont(sFonts[0]);
            lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);
            lP.printText(pF, " This is a test string.",
                         LinePrinter.NO_ALIGNMENT);
            pF = lP.getFont(sFonts[1]);
            lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);
            lP.printText(pF, " This is a test string.",
                         LinePrinter.NO_ALIGNMENT);
            lP.doPrint(1);
         }
         //----------------------(7)--------------------
         manager =
           lP.getParameters().getDriver().getConfigurationManager();
         config1 = manager.getConfiguration("DriverParameters");
         promptUser = config1.getParameters()[0];
         config1.setParameterValue(promptUser, "false");
         manager.save();
      }
      catch (Throwable error){
         error.printStackTrace();
      }
      finally {
         //----------------------(8)--------------------
         try {
            lP.close();
         } catch (Exception ex) {}
      }
         return;
   }
}
```

Adding/Remove a Font Using the Client API

This example shows how to add and remove a font configuration to the PIOS infrastructure using the Client API. In this example the “Batang” font configuration is added, some text is printed with this font configuration, and then it is deleted. The font configuration added must map to an existing font in the printer. The printer is in **Line** mode. This code does is as described below:

1. Opens connection to the printer.
2. Adds the font configuration.
3. Sets the parameters for the font configuration and saves the changes.
4. Closes and reopens the connection to the printer for the font configuration to take effect.
5. Prints a sample text.
6. Deletes the font configuration and saves the changes.
7. Closes the connection to the printer.

Remember to configure the Emulator [Page 111] or printer accordingly.

```
package test;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.printer.*;
import com.sap.ip.me.api.pios.configuration.*;

public class AddFont {

   public static void main(String[] args) {

      LinePrinter lP = null;

      try{

         Connector conn = Connector.getInstance();

         DriverInfo[] driverInfo =
              conn.listDrivers(ConnectionType.PRINTER);
         PrinterParameters params =
              new PrinterParameters(driverInfo[0]);

         params.setPrinterMode(PrinterParameters.LINE_MODE);

         //----------------------(1)--------------------

         lP = (LinePrinter) conn.open(params);

         //----------------------(2)--------------------

         String sFont = "Batang";

         FontConfigurationManager fCM =
               lP.getFontConfigurationManager();

         fCM.addFontConfiguration(sFont);

         Configuration conf = fCM.getFontConfiguration(sFont);


         //----------------------(3)--------------------
         conf.setParameterValue("Size", "14");
         conf.setParameterValue("Options", "0");
         conf.setParameterValue("Name", "Batang");
         conf.setParameterValue("Description", "customfont");
         conf.setParameterValue("FontType", "2");
         fCM.save();

         //----------------------(4)--------------------

         lP.close();
         lP = (LinePrinter) conn.open(params);

         String[] sFonts =
             lP.getFontConfigurationManager().listFontNames();

         //----------------------(5)--------------------

         PrinterFont pF = lP.getFont(sFont);
         lP.printText(pF, "", LinePrinter.NO_ALIGNMENT);

         lP.printText(pF, " This is a Batang test string.",
                      LinePrinter.NO_ALIGNMENT);

         pF = lP.getFont(sFonts[1]);

         lP.printText(pF, " This is a regular test string.",
                      LinePrinter.NO_ALIGNMENT);

         lP.doPrint(1);

         //----------------------(6)--------------------

         fCM.deleteFontConfiguration(sFont);

         fCM.save();


      } catch (Throwable error){

         error.printStackTrace();

      } finally {

         //----------------------(7)--------------------

         try {

            lP.close();

         } catch (Exception ex) {}

      }

      return;

   }

}
```

Scanner API

Purpose

Scanner support in the SAP Mobile Infrastructure is offered as part of the Peripheral Input/Output Services (PIOS). The Scanner API provides an abstraction layer between the intricacies of different scanner drivers and the application.

Integration

The Scanner API is part of the MI Client API.

Features

- Refer to the Scanner API javadocs for detailed method description.
- Refer to Features Description [Page 44].

Constraints

- Scanner models may offer additional features not supported by this API.
- Supported functionality varies according to scanner make and model.

PIOS Scanner API Features Description

Definition

PIOS Scanner API provides support for several features. These features vary depending on the scanner make and model. The table below provides a description for these features:

Scannable Objects

Scannable Object	**Description**
Linear Barcode	An automatic identification technology that encodes information into a linear array of adjacent varying width parallel rectangular bars and spaces.
Two-Dimensional Barcode	An automatic identification technology that encodes information in two dimensions. Small geometric shapes represent the data in these barcodes. Two-dimensional barcodes store more information than linear barcodes in the same space.

Scanner Features

Scanner Feature	**Description**
Soft Trigger	Allows the scan engine laser beam to be triggered from software. It is one shot mode, the laser turns off when a barcode is scanned.
Beep on Fail	Hardware beeps on failed scanning.
Beep on Read	Hardware beeps on successful scan.
Beep off	Turns off the scanner’s hardware beep.
Scan Aware Mode	The application handles the scanned data through events. The scanning process generates events on data received and on error.
Wedge Mode	When using this mode, scanned data is sent as if it was typed from the keyboard. It is sent to the window that has the focus.

Symbologies and Supported Options

Symbology / Option	Description
Codabar	Enables the Codabar barcode symbology.
CLSI Optional	If a CLSI barcode is scanned, the CLSI formatting is applied to the returned data. Otherwise, the API uses any other options selected on the scanned data.
CLSI Required	Scanned barcode must be CLSI compliant. ⚠ No other option can be selected.
NOTIS	NOTIS formatting is applied to the scanned data.
Check Digit MOD16	Enables check digit validation in the scanner engine. The check digit is not transmitted.
Check Digit Transmit	Sends the check digit in a barcode as part of the data. Requires the check digit MOD16 to be enabled.
Code 39	Enables the Code 39 barcode symbology.
Standard	The scanner engine interprets the barcode as standard data. This is the default character set, if no character set is specified. ⚠ Only one character set can be enabled at any given time. The character set can be either the standard or the full ASCII, but not both.
Full ASCII	Support for the whole ASCII character set. ⚠ Only one character set can be enabled at any given time. The character set can be either the standard or the full ASCII, but not both.
Check Digit MOD43	Support for Module 43 check digit. The check digit is not transmitted. ⚠ Only one check digit option can be enabled. The check digit option can be either MOD43 or the French CIP, but not both.

Check Digit French CIP	Support for French CIP (Club Inter Pharmaceutique) check digit. The check digit is not transmitted. ⚠ Only one check digit option can be enabled. The check digit option can be either MOD43 or the French CIP, but not both.
Check Digit Transmit	Sends the check digit in a barcode as part of the data. Requires a check digit option to be enabled.
Code 128	Enables the Code 128 barcode symbology.
EAN-8	Enables the EAN-8 barcode symbology.
Check Digit Transmit	Sends the check digit in a barcode as part of the data.
EAN-13	Enables the EAN-13 barcode symbology.
Check Digit Transmit	Sends the check digit in a barcode as part of the data.
Interleaved 2 of 5	Enables the Interleaved 2 of 5 barcode symbology.
Check Digit MOD10 USS	Support for Module 10 USS (Uniform Symbology Specification) check digit. The check digit is not transmitted. ⚠ Only one check digit option can be enabled. The check digit option can be either MOD10 USS or MOD10 OPCC, but not both.
Check Digit MOD10 OPCC	Support for Module 10 OPCC (Optical Product Code Council) check digit. The check digit is not transmitted. ⚠ Only one check digit option can be enabled. The check digit option can be either MOD10 USS or MOD10 OPCC, but not both.
Check Digit Transmit	Sends the check digit in a barcode as part of the data. Requires a check digit option to be enabled.
PDF417	Enables the PDF-417 (Portable Data Format) 2D barcode symbology.
UCC/EAN-128	Enables the UCC/EAN barcode symbology.
UPC-A	Enables the Universal Product Code (UPC) barcode symbology.
Check Digit Transmit	Sends the check digit in a barcode as part of the data.

Global Options for Symbologies

Option	Description
UPC/EAN Five Digit Add-On	Transmits the two digit add-on as part of the data, if the add-on is present in the barcode. This parameter affects UPC-A, EAN-8, and EAN-13 symbologies.
UPC/EAN Two Digit Add-On	Transmits the five digit add-on as part of the data, if the add-on is present in the barcode. This parameter affects UPC-A, EAN-8, and EAN-13 symbologies.
UPC/EAN Add-On Digits Required	Forces the add-on digits to be present in the barcode. If an add-on is not present, the barcode will not be read. One or both of the add-on options have to be enabled.

PIOS Scanner API Guidelines

Definition

This document explains several guidelines for the PIOS Scanner API, part of the MI Client API. Each of the guidelines is discussed below.

Use

These guidelines are intended for all the developers working on the SAP NetWeaver Developer Studio in a mobile application with scanner peripheral requirements. It helps the developer get the most out of the PIOS infrastructure. The developer can access these features provided by PIOS through the MI Client API.

8. Establishing a connection to the scanner

Since multiple applications can be running in a mobile device at a given time, it is recommended that mobile applications should open the connection to the scanner when they are ready to scan data and close the connection when they have finished receiving the data. This prevents other applications from having to wait for the first one to close before they can open a connection. Otherwise if a mobile application opens the scanner connection when it starts and does not close it until it finishes, no other application may use the scanner until the first one finishes. Any other mobile application that tries to open a connection to the scanner will receive an error message.

9. Start Read and End Read

Use Start Read method only when you need to scan a barcode, and use End Read as soon as the scanning is done. The Start Read method starts the scanner engine thus consuming more processor time and battery power, as opposed to when the engine is stopped.

10. When to use the Scanner Connection, Start Read, and End Read

If an application has several windows (that require scanner support), it is recommended to establish one connection (one open and one close) to the scanner per window. After the connection has been opened, the application may call as many Start/End Reads as necessary.

Any modification (set or remove options) to the Scanner Connection must be made before the start read. If the application tries to set an option after the start read, an error will be generated.

11. Scan Aware Mode and Wedge Mode

It is recommended to use scan aware mode whenever is possible. This mode is more flexible than its counterpart because it allows data manipulation and does not restrict the incoming data to the screen with the focus on it. If a developer is creating an application from scratch, it is recommended that he/she uses the scan aware mode.

Wedge mode should be limited to a situation where an application has already been developed and scanning functionality must be added to the application. Wedge mode requires the least amount of changes to the existing application because it returns data as if it was entered from the keyboard.

12. Symbologies

Use as few symbologies as possible. Limit the active symbologies to the ones that are going to be used by the application. The more active symbologies the scanner has, the longer it takes to initialize.

13. Soft Trigger

The Scanner API provides the ability to activate the laser beam from the application. This feature is particularly important when the only way to activate the laser beam is from the application because there is no physical trigger button. This may be the case with a scanner that attaches to device, for instance.

14. Preamble and Postamble

These two methods allow you to add data before and after the scanned data respectively. This could prove useful when using a tab, or something similar, as postamble.

An application could scan several barcodes and send the scanned data to a spreadsheet. If a tab is set as the postamble, the data is sent to a cell and the focus will skip to the next cell, ready to receive the next barcode.

Scanner API Examples

This section contains Scanner API examples.

Add / Remove Symbologies

In this example the scanner uses Scanner Aware Mode and three barcodes are scanned after Code 39, Code 128, and Codabar symbologies are added to the scanner. The program performs the following steps:

⚠

This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the constructor for the class. It opens a connection to the scanner in scan aware mode and sets an event listener.
2. Implements the onError method from ScannerListener.
3. Implements the onDataReceived method from ScannerListener.
4. Declares and implements a method called scanBarcode. This method starts the scanning engine, waits for the application to read a barcode, and stops the scanning engine.
5. Declares and implements the openScanAwareConnection method.
6. Declares and implements the method called activateSymbologies. This method adds Code 39, Code 128, and Codabar symbologies to the scanner.
7. Declares and implements the method called deactivateSymbologies. This method removes Code 39 and Code 128 symbologies from the scanner.
8. Declares and implements the close method. It closes the connection to the scanner.
9. Creates an instance of the class and calls the activateSymbologies method.
10. Scans three barcodes. The scanned barcodes must have their symbology activated; otherwise the scanner will not read them.
11. Calls the deactivateSymbologies method. Then scans one barcode and closes the connection to the scanner.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;

public class AddingSymbologies implements ScannerListener {

   private ScannerConnection scannerConnection = null;
   private boolean barcodeScanned = false;
   private Exception lastException = null;

   //----------------------(1)----------------------

   public AddingSymbologies() throws PIOSException {

      openScanAwareConnection();
      scannerConnection.setEventListener(this);
   }
```

```
//----------------------(2)----------------------
public void onError(ScannerException e) {
   lastException = e;
   barcodeScanned = true;
}
//----------------------(3)----------------------
public void onDataReceived(ScannerData scannerData) {
   try {
      System.out.println(
         "Data = " + new String(scannerData.toByteArray(),
                 "ASCII"));
   } catch (Exception ex) {
      lastException = ex;
   }
   barcodeScanned = true;
}

//----------------------(4)----------------------
public void scanBarcode() throws Exception {
   scannerConnection.startRead();

   while (!barcodeScanned) {
      Thread.sleep(500);
   }
   if (lastException != null) {
      throw lastException;
   }
   scannerConnection.endRead();
   barcodeScanned = false;
}
//----------------------(5)----------------------
private void openScanAwareConnection() throws PIOSException {
   Connector connector = Connector.getInstance();
   DriverInfo[] scanners =
       connector.listDrivers(ConnectionType.SCANNER);
   ScannerParameters parameters =
       new ScannerParameters(scanners[0]);
   parameters.setMode(ScannerParameters.SCAN_AWARE);
   scannerConnection =
      (ScannerConnection) connector.open(parameters);
}
```

```
   //----------------------(6)----------------------

   public void activateSymbologies() throws PIOSException {

      scannerConnection.addSymbology(new Code39(Code39.FULLASCII));
      scannerConnection.addSymbology(new Code128());
      scannerConnection.addSymbology(
         new Codabar(
            Codabar.CHECK_DIGIT_MOD16
               | Codabar.CHECK_DIGIT_TRANSMIT));
   }

   //----------------------(7)----------------------

   public void deactivateSymbologies() throws PIOSException {

      scannerConnection.removeSymbology(SymbologyType.CODE128);
      scannerConnection.removeSymbology(SymbologyType.CODE39);

   }

   //----------------------(8)----------------------

   public void close() {

      try {
         scannerConnection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }

   public static void main(String[] args) {

      AddingSymbologies addSymbologiesExample = null;

      try {
         //----------------------(9)----------------------

         addSymbologiesExample = new AddingSymbologies();
         addSymbologiesExample.activateSymbologies();

         //----------------------(10)----------------------

         for (int i = 0; i < 3; i++) {
            addSymbologiesExample.scanBarcode();
         }

         //----------------------(11)----------------------

         addSymbologiesExample.deactivateSymbologies();
         addSymbologiesExample.scanBarcode();

      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {

         addSymbologiesExample.close();

      }

   }
}
```

Beep Options

This example deals with the scanner's beep options. It shows how to turn on beeping scanning, for both successful reading and a failed scan. The program executes as explained below:

This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the constructor for the class. It opens a connection to the scanner in scan aware mode, activates the Code 39 symbology, and sets an event listener.
2. Implements the onError method from ScannerListener.
3. Implements the onDataReceived method from ScannerListener.
4. Declares and implements a method called scanBarcode. This method starts the scanning engine, waits for the application to read a barcode, and stops the scanning engine.
5. Declares and implements the openScanAwareConnection.
6. Declares and implements the setBeepOptions method. This method receives and sets the beep options for the scanner.
7. Declares and implements the close method. It closes the connection to the scanner.
8. Opens a connection to the scanner. Turns on the "Beep on Read" and the "Beep on Fail" options for the scanner. Then scans a barcode.
9. Turns off the beeping options for the scanner. Then scans a barcode.
10. Closes the connection to the scanner.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;


public class BeepOptions implements ScannerListener {

   private ScannerConnection scannerConnection = null;

   private boolean barcodeScanned = false;

   private Exception lastException = null;


   //-----------------------(1)-----------------------

   public BeepOptions() throws PIOSException {

      openScanAwareConnection();

      scannerConnection.addSymbology(new Code39(Code39.FULLASCII));

      scannerConnection.setEventListener(this);

   }


   //-----------------------(2)-----------------------
```

```
public void onError(ScannerException e) {
   lastException = e;
   barcodeScanned = true;
}

//----------------------(3)----------------------
public void onDataReceived(ScannerData scannerData) {
   try {
      System.out.println(
         "Data = " + new String(scannerData.toByteArray(),
         "ASCII"));
   } catch (Exception ex) {
      lastException = ex;
   }
   barcodeScanned = true;
}
//----------------------(4)----------------------
public void scanBarcode() throws Exception {
   scannerConnection.startRead();

   while (!barcodeScanned) {
      Thread.sleep(500);
   }

   if (lastException != null) {
      throw lastException;
   }

   scannerConnection.endRead();
   barcodeScanned = false;
}
//----------------------(5)----------------------
private void openScanAwareConnection() throws PIOSException {
   Connector connector = Connector.getInstance();
   DriverInfo[] scanners =
        connector.listDrivers(ConnectionType.SCANNER);
   ScannerParameters parameters =
        new ScannerParameters(scanners[0]);
        parameters.setMode(ScannerParameters.SCAN_AWARE);
        scannerConnection =
           (ScannerConnection) connector.open(parameters);
}
//----------------------(6)----------------------
```

```
   public void setBeepOptions(long options) throws PIOSException {
      scannerConnection.setOptions(options);
   }
   //-----------------------(7)-----------------------
   public void close() {
      try {
         scannerConnection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
   public static void main(String[] args) {
      BeepOptions beepOptionsExample = null;

      try {

         beepOptionsExample = new BeepOptions();

         //-----------------------(8)-----------------------
         beepOptionsExample.setBeepOptions(
            ScannerConnection.BEEP_ON_FAIL
               | ScannerConnection.BEEP_ON_READ);
         beepOptionsExample.scanBarcode();

         //-----------------------(9)-----------------------

   beepOptionsExample.setBeepOptions(ScannerConnection.BEEP_OFF);
         beepOptionsExample.scanBarcode();

      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {
         //-----------------------(10)-----------------------
         beepOptionsExample.close();
      }
   }
}
```

Determine Symbology

This example shows how to determine the symbology of a scanned barcode.

- This feature is available only if the scanner supports it. Some scanners makes and models may not provide this functionality.
- This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the constructor for the class. It opens a connection to the scanner in scan aware mode and sets an event listener.
2. Implements the onError method from ScannerListener.
3. Implements the onDataReceived method from ScannerListener.
4. Declares and implements a method called scanBarcode. This method starts the scanning engine, waits for the application to read a barcode, and stops the scanning engine.
5. Declares and implements the method called activateSymbologies. This method adds Code 39, Code 128, and Codabar symbologies to the scanner.
6. Declares and implements the getSymbologyName method. It returns the symbology name determined during the onDataReceived event.
7. Declares and implements the close method. It closes the connection to the scanner.
8. Creates a new instance of the class and activates symbologies for the scanner.
9. Scans a barcode and displays the symbology of the barcode.
10. Closes the connection to the scanner.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;

public class DetermineSymbology implements ScannerListener {

   private ScannerConnection scannerConnection = null;
   private boolean barcodeScanned = false;
   private String symbologyName = null;
   private Exception lastException = null;

   //----------------------(1)----------------------

   public DetermineSymbology() throws PIOSException {

      Connector connector = Connector.getInstance();

      DriverInfo[] scanners =
          connector.listDrivers(ConnectionType.SCANNER);

      ScannerParameters parameters =
          new ScannerParameters(scanners[0]);

      parameters.setMode(ScannerParameters.SCAN_AWARE);
      scannerConnection =
           (ScannerConnection) connector.open(parameters);
      scannerConnection.setEventListener(this);
   }
   //-----------------------(2)-----------------------
   public void onError(ScannerException e) {
      lastException = e;
      barcodeScanned = true;
   }

   //-----------------------(3)-----------------------
   public void onDataReceived(ScannerData scannerData) {
      symbologyName = scannerData.getSymbology().getName();
      barcodeScanned = true;
   }
   //-----------------------(4)-----------------------
   public void scanBarcode() throws Exception {
      scannerConnection.startRead();

      while (!barcodeScanned) {
         Thread.sleep(500);
      }
      if (lastException != null) {
         throw lastException;
      }
      scannerConnection.endRead();
      barcodeScanned = false;
   }
   //-----------------------(5)-----------------------
   public void activateSymbologies() throws PIOSException {
      scannerConnection.addSymbology(new Code39(Code39.FULLASCII));
      scannerConnection.addSymbology(new Code128());
      scannerConnection.addSymbology(
         new Codabar(
            Codabar.CHECK_DIGIT_MOD16
               | Codabar.CHECK_DIGIT_TRANSMIT));
   }
   //----------------------(6)----------------------
   public String getSymbologyName() {
      return this.symbologyName;
   }

   //----------------------(7)----------------------
   public void close() {
      try {
         scannerConnection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }

   public static void main(String[] args) {
      DetermineSymbology determineSymbologyExample = null;
      try {
         //----------------------(8)----------------------
         determineSymbologyExample = new DetermineSymbology();
         determineSymbologyExample.activateSymbologies();

         //----------------------(9)----------------------
         determineSymbologyExample.scanBarcode();
         System.out.println("Symbology = " +
determineSymbologyExample.getSymbologyName());

      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {
         //----------------------(10)----------------------
         determineSymbologyExample.close();
      }
   }
}
```

Preamble and Postamble

This example shows how to set a preamble and a postamble to scan data. The program performs the following steps:

> ⚠
>
> This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the constructor for the class. It opens a connection to the scanner in scan aware mode, activates the Code 39 symbology, and sets an event listener.
2. Implements the onError method from ScannerListener.
3. Implements the onDataReceived method from ScannerListener.
4. Declares and implements a method called scanBarcode. This method starts the scanning engine, waits for the application to read a barcode, and stops the scanning engine.
5. Declares and implements the setPreamble method. It sets the preamble for the barcode data.
6. Declares and implements the setPostamble method. It sets the postamble for the barcode data.
7. Declares and implements the close method. It closes the connection to the scanner.
8. Creates an instance of the class. Then sets the preamble and postamble.
9. Scans a barcode and displays the scanned data.
10. Closes the connection to the scanner.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;

public class PreAmblePostAmble implements ScannerListener {

   private ScannerConnection scannerConnection = null;
   private String barcodeData = null;
   private boolean barcodeScanned = false;
   private Exception lastException = null;

   //-----------------------(1)-----------------------

   public PreAmblePostAmble() throws PIOSException {

      Connector connector = Connector.getInstance();

      DriverInfo[] scanners =
             connector.listDrivers(ConnectionType.SCANNER);

      ScannerParameters parameters =
             new ScannerParameters(scanners[0]);

      parameters.setMode(ScannerParameters.SCAN_AWARE);

      scannerConnection =
              (ScannerConnection) connector.open(parameters);

      scannerConnection.addSymbology(new Code39(Code39.FULLASCII));
      scannerConnection.setEventListener(this);
   }
   //-----------------------(2)-----------------------
   public void onError(ScannerException e) {
      lastException = e;
      barcodeScanned = true;
   }
   //-----------------------(3)-----------------------
   public void onDataReceived(ScannerData scannerData) {
      try {
         barcodeData = new String(scannerData.toByteArray(),
                       "ASCII");
      } catch (Exception ex) {
         lastException = ex;
      }
      barcodeScanned = true;
   }

   //-----------------------(4)-----------------------
   public void scanBarcode() throws Exception {
      scannerConnection.startRead();

      while (!barcodeScanned) {
         Thread.sleep(500);
      }

      if (lastException != null) {
         throw lastException;
      }

      scannerConnection.endRead();
      barcodeScanned = false;
   }

   //-----------------------(5)-----------------------
   public void setPreamble(String preamble) throws PIOSException {
      scannerConnection.setPreamble(preamble.getBytes());
   }
```

```
    //----------------------(6)----------------------
    public void setPostamble(String postamble) throws PIOSException {
       scannerConnection.setPostamble(postamble.getBytes());
    }

    //----------------------(7)----------------------
    public void close() {
       try {
          scannerConnection.close();
       } catch (Exception ex) {
          ex.printStackTrace();
       }
    }

    public static void main(String[] args) {
       PreAmblePostAmble preamblePostambleExample = null;

       try {
          //----------------------(8)----------------------
          preamblePostambleExample = new PreAmblePostAmble();

          preamblePostambleExample.setPreamble("<-Preamble->");
          preamblePostambleExample.setPostamble("<-Postamble->\t");

          //----------------------(9)----------------------
          preamblePostambleExample.scanBarcode();
          System.out.println("The data with preamble and postamble
is:\n" + preamblePostambleExample.barcodeData);

       } catch (Exception ex) {
          ex.printStackTrace();
       } finally {
          //----------------------(10)----------------------
          preamblePostambleExample.close();
       }
    }
}
```

Soft Trigger

This example shows how to use and activate the soft trigger. The program does as explained below:

This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the constructor for the class. It opens a connection to the scanner in scan aware mode, activates the Code 39 symbology, and sets an event listener.
2. Implements the onError method from ScannerListener.
3. Implements the onDataReceived method from ScannerListener.
4. Declares and implements a method called scanBarcode. This method starts the scanning engine, turns on the soft trigger, waits for the application to read a barcode, and stops the scanning engine.
5. Declares and implements the close method. It closes the connection to the scanner.
6. Creates an instance of the class.
7. Scans a barcode.
8. Closes the connection to the scanner.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;

public class SoftTrigger implements ScannerListener {

   private ScannerConnection scannerConnection = null;
   private boolean barcodeScanned = false;
   private Exception lastException = null;
   public static final boolean ON = true;


   //-----------------------(1)-----------------------

   public SoftTrigger() throws PIOSException {

      Connector connector = Connector.getInstance();

      DriverInfo[] scanners =
              connector.listDrivers(ConnectionType.SCANNER);

      ScannerParameters parameters =
              new ScannerParameters(scanners[0]);

      parameters.setMode(ScannerParameters.SCAN_AWARE);

      scannerConnection =
                 (ScannerConnection) connector.open(parameters);

      scannerConnection.addSymbology(new Code39(Code39.FULLASCII));

      scannerConnection.setEventListener(this);

   }
```

```
//-----------------------(2)-----------------------
public void onError(ScannerException e) {
   lastException = e;
   barcodeScanned = true;
}

//-----------------------(3)-----------------------
public void onDataReceived(ScannerData scannerData) {
   try {
      System.out.println(
         "Data = " + new String(scannerData.toByteArray(),
         "ASCII"));
   } catch (Exception ex) {
      lastException = ex;
   }
   barcodeScanned = true;
}
//-----------------------(4)-----------------------
public void scanBarcode() throws Exception {
   scannerConnection.startRead();
   scannerConnection.setSoftTrigger(ON);

   while (!barcodeScanned) {
      Thread.sleep(500);
   }

   if (lastException != null) {
      throw lastException;
   }

   scannerConnection.endRead();
   barcodeScanned = false;
}
//-----------------------(5)-----------------------
public void close() {
   try {
      scannerConnection.close();
   } catch (Exception ex) {
      ex.printStackTrace();
   }
}
```

```
   public static void main(String[] args) {
      SoftTrigger softTriggerExample = null;

      try {
         //-----------------------(6)-----------------------
         softTriggerExample = new SoftTrigger();

         //-----------------------(7)-----------------------
         softTriggerExample.scanBarcode();

      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {
         //-----------------------(8)-----------------------
         softTriggerExample.close();
      }
   }
}
```

Wedge Mode

This example shows how to use the wedge mode. The program does the following:

This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the class constructor. It opens a connection to the scanner in wedge mode, starts the scanning engine, and launches a window that receives the scanned data.
2. Declares and implements the cancel method. It calls the closeConnection method and closes the window.
3. Declares and implements the closeConnection method. It stops the scanning engine and closes the connection to the printer.
4. Calls the class constructor and displays the window.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;
import java.awt.*;
import java.awt.event.*;

public class WedgeMode extends Frame {

   private ScannerConnection scannerConnection = null;

   //-----------------------(1)-----------------------

   public WedgeMode() throws Exception {

      Connector connector = Connector.getInstance();

      DriverInfo[] scanners =
         connector.listDrivers(ConnectionType.SCANNER);

      ScannerParameters parameters =
         new ScannerParameters(scanners[0]);

      parameters.setMode(ScannerParameters.WEDGE);

      scannerConnection =
        (ScannerConnection) connector.open(parameters);

      scannerConnection.addSymbology(new Code39(Code39.FULLASCII));
      scannerConnection.setPostamble("\t".getBytes());

      scannerConnection.startRead();

      this.setTitle("Wedge Window");
      this.setLayout(new BorderLayout());
      this.setSize(new Dimension(240, 265));
      this.add(new TextArea());

      Button cancel = new Button("Cancel");

      cancel.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {

            cancel();

         }

      });
      Panel p = new Panel(new FlowLayout());
      p.add(cancel);
      this.add(p, BorderLayout.SOUTH);
      cancel.setBounds(new Rectangle(15, 15));
      cancel.repaint();
      this.pack();

      Dimension screenSize =
           Toolkit.getDefaultToolkit().getScreenSize();

      Dimension frameSize = this.getSize();

      if (frameSize.height > screenSize.height) {

         frameSize.height = screenSize.height;

      }

      if (frameSize.width > screenSize.width) {

         frameSize.width = screenSize.width;

      }

      this.setLocation(
         (screenSize.width - frameSize.width) / 2,
         (screenSize.height - frameSize.height) / 2);
   }

   //----------------------(2)----------------------

   public void cancel() {

      this.closeConnection();
      this.dispose();
   }

   //----------------------(3)----------------------

   private void closeConnection () {

      try {

         scannerConnection.endRead();
         scannerConnection.close();

      } catch (Exception ex) {
         ex.printStackTrace();

      }

   }

   public static void main(String[] args) {

      try {

         //----------------------(4)----------------------

         WedgeMode t = new WedgeMode();
         t.setVisible(true);

      } catch (Exception ex) {

         ex.printStackTrace();

      }

   }
}
```

Using the Scanner Attributes

In this example the scanner uses Scanner Aware Mode and scans a barcode after adding (and verifying) all the symbologies supported by the scanner. The program performs the following steps:

This example assumes that at least one driver has been installed. It also assumes this driver supports all the attributes that are being used.

1. Implements the constructor for the class. It opens a connection to the scanner in scan aware mode, if this mode is supported, and sets an event listener. If the scan aware mode is not supported, this method generates an Exception.
2. Implements the onError method from ScannerListener.
3. Implements the onDataReceived method from ScannerListener.
4. Declares and implements a method called scanBarcode. This method starts the scanning engine, waits for the application to read a barcode, and stops the scanning engine.
5. Declares and implements the method called activateAllSymbologies. This method tries to add all the symbologies supported by the API to the scanner. If a symbology cannot be added to the scanner a message is sent to the default output window.
6. Declares and implements the close method. It closes the connection to the scanner.
7. Creates an instance of the class and calls the activateAllSymbologies method.
8. Scans one barcode. The scanned barcode must have their symbology activated; otherwise the scanner will not read them.
9. Closes the connection to the scanner.

```
package scanner_api_examples;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.scanner.*;
import com.sap.ip.me.api.pios.symbology.*;

public class UsingAttributes implements ScannerListener {

   private ScannerConnection scannerConnection = null;
   private boolean barcodeScanned = false;
   private Exception lastException = null;
   private DriverInfo scanner = null;

   //----------------------(1)----------------------

   public UsingAttributes() throws PIOSException {

      Connector connector = Connector.getInstance();

      DriverInfo[] scanners =
             connector.listDrivers(ConnectionType.SCANNER);

      this.scanner = scanners[0];

      if (scanner.isAttributeSupported(

            ScannerConnection.Attributes.SCAN_AWARE_MODE)) {

      ScannerParameters parameters = new ScannerParameters(scanner);

         parameters.setMode(ScannerParameters.SCAN_AWARE);
         scannerConnection =
              (ScannerConnection) connector.open(parameters);
         scannerConnection.setEventListener(this);
      } else {
         throw new PIOSException("Unable to open connection to the
scanner because Scan Aware Mode is not supported.");
      }
   }
   //----------------------(2)----------------------
   public void onError(ScannerException e) {
      lastException = e;
      barcodeScanned = true;
   }

   //----------------------(3)----------------------
   public void onDataReceived(ScannerData scannerData) {
      try {
         System.out.println("************");
         System.out.println(
      "* Data = " + new String(scannerData.toByteArray(), "ASCII"));
         System.out.println("************");
      } catch (Exception ex) {
         lastException = ex;
      }
      barcodeScanned = true;
   }
   //----------------------(4)----------------------
   public void scanBarcode() throws Exception {
      scannerConnection.startRead();

      while (!barcodeScanned) {
         Thread.sleep(500);
      }
      if (lastException != null) {
         throw lastException;
      }
      scannerConnection.endRead();
      barcodeScanned = false;
   }

   //----------------------(5)----------------------
   public void activateAllSymbologies() throws PIOSException {
      if (scanner
         .isAttributeSupported(ScannerConnection.Attributes.CODABAR))
{
         scannerConnection.addSymbology(new Codabar());
      } else {
         System.out.println(
            "Codabar symbology is not supported by the scanner.");
      }
      if (scanner
         .isAttributeSupported(ScannerConnection.Attributes.CODE39))
{
         scannerConnection.addSymbology(new Code39());
      } else {
         System.out.println(
            "Code 39 symbology is not supported by the scanner.");
      }
      if (scanner
         .isAttributeSupported(ScannerConnection.Attributes.CODE128))
      {
         scannerConnection.addSymbology(new Code128());
      } else {
         System.out.println(
            "Code 128 symbology is not supported by the scanner.");
      }
      if (scanner.
          isAttributeSupported(ScannerConnection.Attributes.EAN13)) {
         scannerConnection.addSymbology(new EAN13());
      } else {
         System.out.println(
            "EAN-13 symbology is not supported by the scanner.");
      }
      if (scanner.
          isAttributeSupported(ScannerConnection.Attributes.EAN8)) {
         scannerConnection.addSymbology(new EAN8());
      } else {
         System.out.println(
            "EAN-8 symbology is not supported by the scanner.");
      }
      if (scanner.isAttributeSupported
         (ScannerConnection.Attributes.INTERLEAVED2OF5)) {
         scannerConnection.addSymbology(new Interleaved2Of5());
      } else {
         System.out.println(
         "Interleaved 2 of 5 symbology not supported by scanner.");
      }
      if (scanner .isAttributeSupported
         (ScannerConnection.Attributes.PDF417))
      {
         scannerConnection.addSymbology(new PDF417());
      } else {
         System.out.println(
            "PDF-417 symbology is not supported by the scanner.");
      }
      if (scanner.isAttributeSupported
            (ScannerConnection.Attributes.UCCEAN128)) {
         scannerConnection.addSymbology(new UCCEAN128());
      } else {
         System.out.println(
         "UCC/EAN 128 symbology is not supported by the scanner.");
      }
      if (scanner.isAttributeSupported
         (ScannerConnection.Attributes.UPC_A)) {
         scannerConnection.addSymbology(new UPC_A());
      } else {
         System.out.println(
            "UPC-A symbology is not supported by the scanner.");
      }
   }
   //-----------------------(6)-----------------------
   public void close() {
      try {
         scannerConnection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
```

```
public static void main(String[] args) {
      UsingAttributes usingAttributesExamples = null;

      try {
         //-----------------------(7)-----------------------
         usingAttributesExamples = new UsingAttributes();
         usingAttributesExamples.activateAllSymbologies();

         //-----------------------(8)-----------------------
         usingAttributesExamples.scanBarcode();

      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {
         //-----------------------(9)-----------------------
         usingAttributesExamples.close();
      }
   }
}
```

RFID API

Purpose

Radio Frequency Identification (RFID) support in the SAP Mobile Infrastructure is offered as part of the Peripheral Input/Output Services (PIOS). The RFID API provides an abstraction layer between the intricacies of different RFID readers' drivers and the application.

Integration

The RFID API is part of the MI Client API.

Features

- Refer to the RFID API Javadocs for detailed method description.
- Refer to RFID Features Description [Page 72].

Constraints

- RFID readers' models may offer additional features not supported by this API.
- Supported functionality varies according to RFID reader make and model.

PIOS RFID API Features Description

Definition

PIOS RFID API provides support for several features. These features vary depending on the RFID reader make and model. The table below provides a description for these features:

RFID Reader Features

RFID Reader Feature	Description
Program Tag ID	Assigns a new Tag ID to all the tags inside the RFID range. To avoid ID duplicates, make sure that only one tag is in the reader's area of coverage before performing this operation.
Identify All	The RFID reader has the ability to return all the tags available within its area of coverage. The tag type must have been previously defined in the tag configuration file.
Identify by Tag Type	The RFID reader returns all the available tags, within its area of coverage, that match the specified tag type. The tag type must be defined in the tag configuration file and supported by the RFID reader hardware.
Read a Single Tag	The RFID peripheral reads bytes from a specific tag.
Write a Single Tag	The RFID peripheral writes bytes to a specific tag.
Lock Tag ID with Password	This feature locks (with a password) a tag ID so it cannot be changed. Once locked, the tag must be reset in order to be reprogrammed with a new the tag ID. All tags inside the RFID range will be locked.
Reset Tag ID with Password	Resets a locked tag ID, making it programmable again. This feature unlocks and erases the tag ID. To reset the tag ID, you need the password used to lock the tag ID in the first place.

PIOS RFID API Guidelines

Definition

This document explains several guidelines for the PIOS RFID API, part of the MI Client API. Each of the guidelines is discussed below.

Use

These guidelines are intended for all the developers working on the SAP NetWeaver Developer Studio in a mobile application with RFID peripheral requirements. It helps the developer get the most out of the PIOS architecture. The developer can access these features provided by PIOS through the MI Client API.

1. **Tag Configuration File**

The RFID API programs, reads, and/or writes tag types that have been predefined in the tag configuration file. The tags can be edited, added to, or removed from the file using the tag configuration manager, part of the RFID API.

The RFID reader can only work (read, write, and so on) those tag types defined in the tag configuration file and supported by the hardware. Supported tag types depend on the RFID reader make and model.

⚠

Tags that are not defined in the tag configuration file will be ignored by the API even if they are supported by the RFID reader.

2. **List Tag Types method**

This method returns all the tag types that have been configured in the API. An array of tag types is returned. The configured tag types are returned in an array. This method must be called before an *identify by tag type* is called, this assures using a valid tag type when calling the *identify*.

3. **Tag configuration manager**

To receive a list of all the configured tag types for a specific RFID reader, use the *listTagTypes* method. To edit the list by adding or removing a tag type, or to edit a particular tag type use the *TagConfigurationManager* class.

4. **Identify method**

To get a list of all the tags (that are defined in the API) inside the RFID reader's range, use the *identify* method with no parameters. To get a list of all the tags in range of a specific tag type, pass the tag type as a parameter to the *identify* method.

5. **Read and write operations**

The API can read from or write data to one tag at any given time. The tag to be accessed must be inside the RFID reader's range. To accomplish this, call the *identify* method before invoking either the read or the write. This guarantees the use of a valid (in range) tag.

6. **Tag structure awareness**

An RFID tag may be divided in several areas. These areas can be one the following:

   a. Reserved - This data is written by the tag manufacturer. The data in this area may be used internally by the tag.
   b. Read only - A portion of the data that was both written and locked, or was written in a single-use tag. The information in this area is accessible but cannot be changed.
   c. Writable - This area may be written and read using the API.

It is important to be aware of the different areas and to avoid trying to write to reserved or read-only areas. Otherwise an exception will be thrown.

7. **Writing without exceptions**

The following procedure is recommended to write to a tag without raising an exception:

   a. Identify the tags in the RFID reader's range, either with a tag type as a parameter or without a tag type. It depends on what needs to be done. A list of available tags is returned.
   b. Select the tag that will be written to from the list and get its tag type.
   c. Use the tag type writable areas for the tag to determine what section or sections of the tag can be written to.

By doing this, any exception should be avoided.

RFID API Examples

This section contains RFID API examples.

Identify All

In this example the RFID reader identifies all the tags, independently of the tag type, in the reader's range. The program performs the following steps:

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Checks if the driver supports the "identify all" attribute and displays how many tags are in range.
3. Displays the tag ID and tag type for each of the detected RFID tags.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;
public class ExampleIdentifyAll {
   public static void main(String[] args) {
      RfidConnection rfidConnection = null;
      try {
         Connector connector = Connector.getInstance();
         //----------------------(1)--------------------
         DriverInfo[] rfidDrivers =
            connector.listDrivers(ConnectionType.RFID);
         if (rfidDrivers.length > 0) {
            RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);
            rfidConnection = (RfidConnection) connector.open(rfidParams);
            //----------------------(2)--------------------
            if (rfidDrivers[0].isAttributeSupported
                       (RfidConnection.Attributes.IDENTIFY_ALL)) {
               RfidTag[] tagList = rfidConnection.identify();
               if (tagList.length <= 0) {
                  System.out.println("There are no tags in range.");
               } else {
                  System.out.println(
                     "Total tags: " + tagList.length + ".");
               }
               //----------------------(3)--------------------
               for (int i = 0; i < tagList.length; i++) {
                  System.out.print("TagID: ");
                  for (int j = 0;
                     j < tagList[i].getTagID().length;
                     j++) {
```

```
                     System.out.print(tagList[i].getTagID()[j] + " ");
                  }
                  System.out.println(
                     "\t  tag type: "
                        + tagList[i].getTagType().getName());
               }
            } else {
               System.out.println(
                  "Required driver attribute is not supported.");
            }
         } else {
            System.out.println("There are no RFID drivers.");
         }
      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {
         try {
            rfidConnection.close();
         } catch (PIOSException pex) {
            pex.printStackTrace();
         }
      }
   }
}
```

Identify by Tag Type

In this example the RFID reader perform an *identify* by tag type. The program does as explained below:

> This example assumes that the RFID reader supports and has configured the "TAGIT_HF_TYPE1" tag type. Supported tag types vary depending on RFID make and model.

1. Gets the RFID parameters from the first driver and opens a connection to it.
2. Checks if the driver supports the "identify by tag type" attribute and checks if the "TAGIT_HF_TYPE1" tag type is supported.
3. If the tag type is supported, it performs an identify for this tag type and displays how many tags were found in the reader's range.
4. Displays the RFID tags found, their ID, and their type.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleIdentifyTagType {

   public static void main(String[] args) {

      RfidConnection rfidConnection = null;

      try {

         //----------------------(1)--------------------

         Connector connector = Connector.getInstance();

         DriverInfo[] rfidDrivers =

            connector.listDrivers(ConnectionType.RFID);

         if (rfidDrivers.length > 0) {

            RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);

            rfidConnection = (RfidConnection) connector.open(rfidParams);

            //----------------------(2)--------------------

            if (rfidDrivers[0].isAttributeSupported
                  (RfidConnection.Attributes.IDENTIFY_BY_TAG_TYPE)) {

               RfidTagType[] supportedTags = rfidConnection.listTagTypes();

               RfidTagType particularTagType = null;

               for (int i = 0; i < supportedTags.length; i++) {

                  if ("TAGIT_HF_TYPE1".equals(supportedTags[i].getName()))
                  {

                     particularTagType = supportedTags[i];

                  }

               }
```

```
                //----------------------(3)--------------------

                    if (particularTagType != null) {
                        RfidTag[] tagList =
                            rfidConnection.identify(particularTagType);

                        if (tagList.length <= 0) {
                            System.out.println("There are no tags in range.");

                        } else {

                            System.out.println(
                                "Total tags: " + tagList.length + ".");

                        }

                        //----------------------(4)--------------------

                        for (int j = 0; j < tagList.length; j++) {

                            System.out.print("TagID: ");

                            for (int k = 0;

                                k < tagList[j].getTagID().length;

                                k++) {

                                System.out.print(
                                    tagList[j].getTagID()[k] + " ");

                            }

                            System.out.println(
                                "\t  tag type: "
                                    + tagList[j].getTagType().getName());

                        }

                    } else {

                        System.out.println(
                            "The tag type \"TAGIT_HF_TYPE1\" is not supported.");

                    }

                } else {

                    System.out.println(

                        "Required driver attribute is not supported.");

                }

            } else {

                System.out.println("There are no RFID drivers.");

            }

        } catch (Exception ex) {

            ex.printStackTrace();

        } finally {

            try {

                rfidConnection.close();

            } catch (PIOSException pex) {

                pex.printStackTrace();

            }

        }

    }

}
```

Read

In this example the RFID reader reads data from a tag. The program does as described below:

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Checks if the "identify all" and the "read single" attributes are supported by the driver.
3. Checks if there are RFID tags within range and for the first of those tags that has readable areas.
4. Reads the tag readable areas and displays the read data.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleReadTag {

   public static void main(String[] args) {

       RfidConnection rfidConnection = null;

       try {

          //----------------------(1)--------------------

          Connector connector = Connector.getInstance();

          DriverInfo[] rfidDrivers =
             connector.listDrivers(ConnectionType.RFID);

          if (rfidDrivers.length > 0) {
             RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);
             rfidConnection = (RfidConnection) connector.open(rfidParams);
             //----------------------(2)--------------------

             if (rfidDrivers[0].isAttributeSupported
                 (RfidConnection.Attributes.IDENTIFY_ALL)
                && rfidDrivers[0].isAttributeSupported(
                   RfidConnection.Attributes.READ_SINGLE)) {

                //----------------------(3)--------------------

                RfidTag[] tagList = rfidConnection.identify();
                if (tagList.length > 0) {
                   boolean readableAreas = false;
                   RfidTag sampleTag = null;
                   for (int i = 0;
                      (i < tagList.length) && !readableAreas; i++) {
                      RfidTagUserArea[] areas =
                         tagList[i].getTagType().getUserReadableAreas();

                      if (areas.length > 0) {
                         readableAreas = true;
                         sampleTag = tagList[i];

                      }
                   }
                   //----------------------(4)--------------------
                   if (sampleTag != null) {
                      RfidTagUserArea[] areas =
                          sampleTag.getTagType().getUserReadableAreas();
                      for (int j = 0; j < areas.length; j++) {
                          int readableAreaLength =
                          areas[j].getEndPos()- areas[j].getStartPos() + 1;
                          RfidTagData data =
                           rfidConnection.read(
                              sampleTag,
                              areas[j].getStartPos(),
                              readableAreaLength);
                        if (data != null) {
                           System.out.println(
                              "Tag data for area #" + (j + 1) + " is: ");
                           for (int k = 0;
                              k < data.getTagData().length; k++) {
                              System.out.print(
                                 " " + data.getTagData()[k]);
                              if (((k % 20) == 0) && (k > 0)) {
                                 System.out.println();
                              }
                           }
                        }
                     }
                  } else {
                     System.out.println("No readable tags in range.");
                  }
               } else {
                  System.out.println("No tags in range.");
               }
            } else {
               System.out.println(
         "The required attributes are not supported by this driver.");
            }
         } else {
            System.out.println("There are no RFID drivers.");
         }
      } catch (Exception ex) {
         ex.printStackTrace();
      } finally {
         try {
            rfidConnection.close();
         } catch (PIOSException pex) {
            pex.printStackTrace();
         }
      }
   }
}
```

Write

In this example the RFID reader writes data to a tag. The program does the following:

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Checks if the driver supports the "identify all", the "write single", and the "read single" attributes.
3. Looks for the first tag with writable areas.
4. Creates a byte array, with random values, of the size of each of the writable area.
5. Writes the data above to the tag and reads the data back for corroboration.
6. Checks, byte per byte, if the written and read data are the same.

```
import java.util.*;

import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleWriteTag {

   public static void main(String[] args) {

       RfidConnection rfidConnection = null;

       try {

          //----------------------(1)--------------------

          Connector connector = Connector.getInstance();

          DriverInfo[] rfidDrivers =
             connector.listDrivers(ConnectionType.RFID);

          if (rfidDrivers.length > 0) {

             RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);

             rfidConnection = (RfidConnection) connector.open(rfidParams);

             //----------------------(2)--------------------

             if (rfidDrivers[0].isAttributeSupported(
                    RfidConnection.Attributes.IDENTIFY_ALL)
                 && rfidDrivers[0].isAttributeSupported
                     (RfidConnection.Attributes.WRITE_SINGLE)
                 && rfidDrivers[0].isAttributeSupported
                     (RfidConnection.Attributes.READ_SINGLE)) {

                 RfidTag[] tagList = rfidConnection.identify();

                 if (tagList.length > 0) {

                    boolean writableAreas = false;
                    RfidTag sampleTag = null;

                    //----------------------(3)--------------------

                    for (int i = 0;

                       (i < tagList.length) && !writableAreas; i++) {

                       RfidTagUserArea[] areas =

                          tagList[i].getTagType().getUserWritableAreas();
```

```
      if (areas.length > 0) {
         writableAreas = true;
         sampleTag = tagList[i];
      }
   }
   if (sampleTag != null) {
      RfidTagUserArea[] areas =
         sampleTag.getTagType().getUserWritableAreas();
      //----------------------(4)--------------------
      for (int j = 0; j < areas.length; j++) {
         int writableSize =
         areas[j].getEndPos()-areas[j].getStartPos()+1;
         byte[] dataBytes = new byte[writableSize];
         new Random().nextBytes(dataBytes);
         //----------------------(5)--------------------
         rfidConnection.write(
            sampleTag,
            areas[j].getStartPos(),
            dataBytes);
         RfidTagData data =
            rfidConnection.read(
               sampleTag,
               areas[j].getStartPos(),
               writableSize);
         //----------------------(6)--------------------
         if (data != null) {
            System.out.println
              ("Check tag data #" + (j + 1) + ": ");
            if (isEqual(data.getTagData(),
               dataBytes)) {
               System.out.println(
            "The written and read data are the same.");
            } else {
               System.out.println(
            "The written and read data are different.");
            }
         }
      }
   } else {
      System.out.println("No writable tags in range.");
   }
} else {
   System.out.println("No tags in range.");


}
```

```
                } else {
                    System.out.println(
                "The required attributes are not supported by this driver.");
                }
            } else {
                System.out.println("There are no RFID drivers.");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                rfidConnection.close();
            } catch (PIOSException pex) {
                pex.printStackTrace();
            }
        }
    }

    public static boolean isEqual(byte[] a, byte[] b) {
        if (a.length != b.length) {
            return false;
        }
        for (int i = 0; i < a.length; i++) {
            if (a[i] != b[i]) {
                return false;
            }
        }
        return true;
    }
}
```

List Tag Types

In this example the program gets a list of supported tag types and the RFID identifies the tags that are within its range for supported tag types. The program does the following:

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Identifies RFID tags by tag type and displays the list of tags, inside the reader's range, for each tag type.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleListTagType {

   public static void main(String[] args) {

      RfidConnection rfidConnection = null;

      try {

         //----------------------(1)--------------------

         Connector connector = Connector.getInstance();

         DriverInfo[] rfidDrivers =
            connector.listDrivers(ConnectionType.RFID);

         if (rfidDrivers.length > 0) {

            RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);
            rfidConnection = (RfidConnection) connector.open(rfidParams);
            RfidTagType[] tagTypeList = rfidConnection.listTagTypes();

            //----------------------(2)--------------------

            for (int pos = 0; pos < tagTypeList.length; pos++) {

               System.out.println(

                  "****** Identify Tags: " + tagTypeList[pos].getName());

               RfidTag[] tags = rfidConnection.identify(tagTypeList[pos]);

               if (tags.length > 0) {

                  for (int count = 0; count < tags.length; count++) {

                     System.out.print(
                     "Type : " + tagTypeList[pos].getName()+ "   ID: ");
                     byte[] tagID = tags[0].getTagID();
                     for (int i = 0; i < tagID.length; i++) {
                        System.out.print(tagID[i] + " ");

                     }
                     System.out.println();

                  }

                  System.out.println("Total: " + tags.length + "\n");

               } else {
                  System.out.println(
                      "No tags in range for type "
                      + tagTypeList[pos].getName());

               }

            }
            } else {
                System.out.println("There are no RFID drivers.");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            try {
                rfidConnection.close();
            } catch (PIOSException pex) {
                pex.printStackTrace();
            }
        }
    }
}
```

Tag Configuration Manager

In this example a new tag type configuration is added. The program performs the following steps:

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Adds the "NewTagType" configuration and sets all the tag parameters to "NEW VALUE". Displays each of the modified tag parameters.
3. Saves the configuration.

The value, "NEW VALUE", is used to illustrate the use of the tag configuration manager and not intended to be used in reality.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.configuration.Configuration;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleTagConfigurationManager {

    public static void main(String[] args) {
        RfidConnection rfid = null;
        try {
            //----------------------(1)--------------------
            Connector connector = Connector.getInstance();
            DriverInfo[] rfidDrivers =
                connector.listDrivers(ConnectionType.RFID);
            if (rfidDrivers.length > 0) {
                RfidParameters parameters = new RfidParameters(rfidDrivers[0]);
                rfid = (RfidConnection) connector.open(parameters);
                TagConfigurationManager manager =
                        rfid.getTagConfigurationManager();
                //----------------------(2)--------------------
                Configuration tagConfiguration =
                    manager.addTagConfiguration("NewTagType");

                String[] tagParameters = tagConfiguration.getParameters();

                for (int i = 0; i < tagParameters.length; i++) {
                    tagConfiguration.setParameterValue(
                        tagParameters[i], "NEW VALUE");
                    System.out.println(tagParameters[i]);

                }
                //----------------------(3)--------------------
                manager.save();
            } else {
                System.out.println("There are no RFID drivers.");
            }
        } catch (Exception ex) {
            ex.printStackTrace();

        } finally {
            try {
                rfid.close();


            } catch (PIOSException pex) {
                pex.printStackTrace();
            }
        }
    }
}
```

Program and Lock a Tag ID

In this example the RFID reader assigns a new tag ID to any programmable tag inside its range and locks the tag with a password. The program does as explained below:

For this example, it is assumed that tag ID is 12-bytes long and the password has a length of one character.

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Checks if the driver supports the "program tag ID" and "lock tag ID with password" attributes.
3. Assigns the new tag ID to any tag inside the reader's range and locks it.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleProgramTagID {

   public static void main(String[] args) {

       RfidConnection rfidConnection = null;
       try {
          //----------------------(1)--------------------
          Connector connector = Connector.getInstance();
          DriverInfo[] rfidDrivers =
             connector.listDrivers(ConnectionType.RFID);

          if (rfidDrivers.length > 0) {
             RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);

             rfidConnection = (RfidConnection) connector.open(rfidParams);

             //----------------------(2)--------------------

             if (rfidDrivers[0].isAttributeSupported
                  (RfidConnection.Attributes.PROGRAM_TAG_ID)
                  && rfidDrivers[0].isAttributeSupported
                  (RfidConnection.Attributes.LOCK_TAG_ID_WITH_PSWD)) {
              //----------------------(3)--------------------
                rfidConnection.programTagID(
                       new byte[] { -1,2,-3,4,15,26,47,-88,9,10,11,-50 });
             byte[] pwd = "z".getBytes();
                rfidConnection.lockTagID(pwd);
                rfidConnection.close();
             } else {
                System.out.println(
             "The required attributes are not supported by this driver.");
          }
          } else {
             System.out.println("There are no RFID drivers.");        }
   } catch (Exception ex) {
          ex.printStackTrace();
       } finally {
          try {
             rfidConnection.close();
          } catch (PIOSException pex) {
             pex.printStackTrace();
          }
       }
   }
}
```

Reset a Tag ID

In this example the RFID reader resets the tag ID of each tag inside its range, using the password used to lock the tags (see Program and Lock a Tag ID [Page 87]). The program does the following:

For this example, it is assumed that tag ID is 12-bytes long and the password has a length of one character.

1. Gets the RFID parameters from the first driver and, using them, opens the connection to the RFID reader.
2. Checks if the driver supports the "identify all" and the "reset tag ID with password" attributes.
3. Identifies all the tags within the reader's range and tries to reset each of them.

```
import com.sap.ip.me.api.pios.PIOSException;
import com.sap.ip.me.api.pios.connection.*;
import com.sap.ip.me.api.pios.rfid.*;

public class ExampleResetTagID {

    public static void main(String[] args) {

        RfidConnection rfidConnection = null;

        try {
            //----------------------(1)--------------------

            Connector connector = Connector.getInstance();

            DriverInfo[] rfidDrivers =
                connector.listDrivers(ConnectionType.RFID);

            if (rfidDrivers.length > 0) {
                RfidParameters rfidParams = new RfidParameters(rfidDrivers[0]);
                rfidConnection = (RfidConnection) connector.open(rfidParams);

                //----------------------(2)--------------------

                if (rfidDrivers[0].isAttributeSupported
                    (RfidConnection.Attributes.IDENTIFY_ALL)
                      && rfidDrivers[0].isAttributeSupported
                    (RfidConnection.Attributes.RESET_TAG_ID_WITH_PSWD)) {

                    RfidTag[] tagList = rfidConnection.identify();

                    if (tagList.length > 0) {
                        byte[] pwd = "z".getBytes();

                        //---------------------(3)--------------------

                        for (int i = 0; i < tagList.length; i++) {
                            try {
                                rfidConnection.resetTagID(tagList[i], pwd);
                            } catch (Exception exReset) {
                                System.out.println(
                                   "There was an error resetting tag #"
                                    + i + " (tag type: "
                                    + tagList[i].getTagType().getName()  + ").");
                            }
                        }
                    }
                } else {
                  System.out.println(
                "The required attributes are not supported by this driver.");
                }
            } else {
                System.out.println("There are no RFID drivers.");
            }
        } catch (Exception ex) {
            ex.printStackTrace();

        } finally {
            try {
                rfidConnection.close();
            } catch (PIOSException pex) {
                pex.printStackTrace();
            }
        }
    }
}
```

PIOS Add-on Drivers

The following chapter provides information about the PIOS add-on drivers.

For more information about driver known issues and limitations, driver attributes and configuration options refer to SAP Note 761833.

Installing a Driver Add-on

Purpose

This process is intended for system administrators deploying a mobile application with peripheral requirements. The system administrator deploys drivers and connector add-ons that meet the requirements of the mobile application.

Prerequisites

There is a connector add-on (it can be deployed when the driver is deployed).

Process Flow

1. A mobile application with peripheral requirements is uploaded to the SAP MI Web Console.
2. The system administrator uses the driver selection tool [Page 150] to find the driver add-on that matches the target platform.
3. The administrator checks if the connector add-on for the target platform is uploaded to the SAP MI Web Console. If the matching driver and/or connector are not loaded to the SAP MI Web Console, they can be obtained from the *SAP Service Marketplace*.
4. After the connector and driver add-ons have been uploaded to the SAP MI Web Console, the administrator deploys them.

   If the connector is not installed on the target device, both the connector and the driver add-ons can be deployed at the same time as the application. If there is a connector in the target mobile device, only the driver and application need to be deployed.

Result

The mobile application and the required driver add-on are deployed to the target mobile device.

Driver Configuration

Purpose

Peripheral input output services (PIOS) drivers can be configured by modifying several parameters. Parameters are defined for different configurations, and configurations are divided into configuration types. The system administrator can modify configuration parameters with the Mobile Infrastructure configuration system.

Implementation Considerations

Parameters of driver add-ons are formed with four tokens. The first token is the *driver name*. The second token is the *configuration type*. The third and fourth tokens are *configuration name* and *parameter* respectively. Modification of driver parameters should follow the naming convention presented below:

<driver name>.<configuration type>.<configuration name>.<parameter>=value

> ⚠
>
> Tokens and values are case-sensitive. The correct name must be entered in the MI configuration system to change a parameter value.

Integration

- Driver configurations are handled by the MI configuration system.
- Drivers and driver configurations are assigned using the SAP MI Web Console.

Features

Drivers have several parameters that can be used to change options for a driver. These parameters are defined for configurations that are separated into configuration types. PIOS drivers use the driver configuration type (cfg) to store parameters that modify how drivers connect to peripherals. This configuration type is also used to store parameters that are specific from driver to driver. A different configuration type is used to store the font configuration parameters (fntcfg) for peripheral type “Printer”. This configuration type (fntcfg) is used to configure fonts supported by the driver.

System administrators can modify driver configuration and font configuration parameters with the MI configuration system.

Constraints

- Adding a font with the font configuration parameters does not install the font on the physical printer. Printer fonts must be installed manually on the printer and should match the configured parameters.
- Configuration values are applied to drivers without validation. Unexpected behavior may be detected if a driver is not configured properly.

Example

Examples for parameter configuration and font configuration parameters are given below:

- In this example, a driver configuration parameter for the piprsymmf4t (Symbol microFlash 4t) printer driver is configured. This line sets the serial port baud rate to 9600 bits per second:

  ```
  piprsymmf4t.cfg.Serial.BaudRate=9600
  ```

- This example modifies a font configuration parameter for the piprmswin32 (Microsoft Windows 32-bit) printer driver. This line sets the “bitmapped bold italic” font options to bold and italic:

  ```
  piprmswin32.fntcfg.BitmappedBoldItalic.Options=bold,italic
  ```

See Also

You can find a list of all available parameters in a SAP note that is created for each driver. For a list of available drivers see the collective SAP Note **761833**.

MDK Peripheral Support Actions

Purpose

The MDK Peripheral Support Actions is part of the SAP Mobile Infrastructure perspective of the SAP NetWeaver Developer Studio. It enables the developer to:

1. Create/Edit Driver Requirements Document (DRD)
2. Set MDK Peripheral Emulation Mode
3. Launch the Peripheral I/O Emulator.

Implementation Considerations

The DRD is required for those mobile applications that require some peripheral support. It identifies the peripheral types as well as the attributes within those peripheral types required by an MI application. It is used by both the developer and the system administrator. The developer specifies the peripheral requirements. The system administrator later uses it in the Driver Selection Tool (DST) of the Web Console to find drivers that match those peripheral requirements.

An option to set the MDK in peripheral emulation mode is also provided and enables the developer to automatically switch the output destination from the peripheral to the Emulator and vice versa. Launching the Peripheral Input/Output Emulator from SAP Mobile Infrastructure perspective is also possible.

Integration

The MDK Peripheral Support Actions is part of the Mobile Infrastructure perspective of the SAP NetWeaver Developer Studio.

Features

The MDK Peripheral Support Actions has functions to:

- Add a new DRD to an SAP MI project.
- Edit an existing DRD.
    - Add/Remove peripheral types from an existing DRD.
    - Select/Unselect required attributes for each peripheral type.
- Set MDK Peripheral Emulation Mode.
- Launch the Peripheral I/O Emulator.

Driver Requirements Document Editor

Definition

The Driver Requirements Document Editor allows the developer to select the peripheral types and their respective options required for the mobile application.

Use

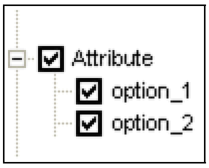
The Driver Requirements Document Editor is displayed:

- After adding a new DRD to an MI project.
- By double clicking an existing DRD in the project.
- After adding or removing peripheral types from the current DRD.
-

Structure

The DRD Editor has three components:

1. The **attributes tree window**. Displays all available attributes of a peripheral type. It is used to select the required attributes.
    - **Attribute:** Refers to the parent node. When checked, it implies support for all corresponding options under the parent node.
    - **Options:** Refers to each node under an attribute.



☑ Attribute
  ☑ option_1
  ☑ option_2

Attributes and options available are those supported by the:

- Printer API [Page 18]
- Scanner API [Page 44]
- RFID API [Page 72]

2. The **Description Panel:** displays a short description of the selected attribute. It is the panel located at the bottom of the Driver Requirements Document Editor.
3. **Peripheral Type tab:** Used to select the peripheral attribute tree window from the available peripheral types.

Using the Driver Requirements Document Editor

Use

The editor presents the developer with the attributes tree window. It is important to select all peripheral attributes and options required by the application in order for the Driver Selection Tool to accurately select peripheral model drivers that match those requirements. Attributes and options that are not selected may or may not be supported by the selected peripheral model.

Features

Two scenarios are supported to select attributes and options in the attributes tree:

1. Select only the parent **-** When checked, all corresponding attributes options under the root parent node are also checked.
2. Select one or more options **-** It means required support for all the selected attributes options under the root parent node. They follow a logical AND behavior, each selected option becomes a requirement that must be met during the peripheral add-on selection process.

Printer head width in the Printer peripheral type is the exception to this rule. In this case, when more than one option is selected at least one of them will be matched. In other words, the *Printer head width* attribute follows a logical OR behavior, where the requirement is met when at least one of the selected options is met.

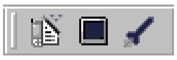
Activities

The developer must check every peripheral option required by the mobile application. Any option that has not been selected is ignored by the Driver Selection Tool. In other words, peripheral drivers displayed in the Driver Selection Tool as matched drivers, may or may not provide support for unchecked options.

MDK Peripheral Support Actions Toolbar

Use

The MDK Peripheral Support Actions Toolbar provides quick access to the Peripheral Support Actions available in the Mobile Infrastructure perspective:



Each of the buttons is further explained below.

Prerequisites

- The SAP NetWeaver Developer Studio is installed.
- An Eclipse project is available and selected.
- The Mobile Infrastructure perspective is selected.

Features

Toolbar Functions

Button	Function Name	Meaning
📝	*Create/Modify Driver Requirements Document*	Opens the wizard to create a new Driver Requirements Document. If a DRD is present in the project, opens the wizard to modify it.
■	*Set MDK Peripheral Emulation Mode*	Toggles the Emulation Mode On/Off. Configures the MI Client to use the Peripheral Input / Output emulator instead of the physical peripheral.
🖌	*Launch Peripheral I/O Emulator*	Launches the Peripheral Input / Output Emulator.

Creating a Driver Requirements Document

Use

This procedure describes the steps required to create a Driver Requirements Document.

Prerequisites

- Eclipse must be in the Mobile Infrastructure perspective.
- An existing project has been selected and the project does not have an existing Driver Requirements Document.

Procedure

3. Click on the *Create/Modify Driver Requirements Document* button. The *New Driver Requirements Document* wizard opens.
4.
5. You can also access the wizard by choosing *New* → *Other...* → *MDK Development Tools* on the *File* menu and selecting *Driver Requirements Document*.
6. Select the peripherals required by the mobile application from the *Peripherals Available* list. Click *Add >*.
7. After all the required peripheral types have been selected, click *Finish*. The *Driver Requirements Document Editor* appears as part of the selected project.

Result

A Driver Requirements Document is created.

Modifying the Driver Requirements Document

Use

Modify an existing Driver Requirements Document (DRD). It allows additions or deletions of peripheral types from an existing DRD.

Prerequisites

- The Mobile Infrastructure perspective is selected.
- A project containing a Driver Requirements Document is selected.

Procedure

1. Click on the *Create/Modify Driver Requirements Document* button. The *Modify existing Driver Requirements Document* wizard opens.
2. 💡
3. You can also access the wizard by choosing *New* → *Other...* → *MDK Development Tools* on the *File* menu and selecting the Driver Requirements Document. If a DRD is present in the project, the *Modify existing Driver Requirements Document* wizard will open.
4. Add or remove peripheral types as desired.
5. Click Finish. The *Driver Requirements Document Editor* opens.
6. Select the peripheral attributes required by the application.
    a. Unselect those that are not required.

Result

The Driver Requirements Document is modified.

Launching the Peripheral I/O Emulator

Use

Launch the Peripheral Input/Output Emulator from the Mobile Infrastructure perspective.

Prerequisites

- Eclipse must be in the Mobile Infrastructure perspective.

Procedure

1. Click on the *Launch the Peripheral I/O Emulator* button. The Emulator is launched.
    - You can also launch the Emulator by selecting *Launch the Peripheral I/O Emulator* on the *Edit* menu.
    - Refer to the Peripheral Input/Output Emulator [Page 100] for further details on the use of the Emulator.

Result

The Emulator is launched.

Set MDK Peripheral Emulation Mode

Use

This mode enables the developer to automatically switch use of the peripheral to the emulator and vice versa. When the developer turns on the Emulation Mode, the application, while running inside the MI Client, uses peripheral emulation.

Prerequisites

- The SAP NetWeaver Developer Studio is in the Mobile Infrastructure perspective.

Procedure

1. Turn on the Emulation Mode by clicking on the *Set MDK Peripheral Emulation Mode* ▣ button.

   You can also set the MDK in emulation mode by selecting *Set MDK Peripheral Emulation Mode* from the *Edit* menu.

2. Restart the MI Client for the settings to take effect.
3. Run your application.

Result

Your application will use the Peripheral I/O emulator.

Peripheral Input/Output Emulator

Purpose

The Peripheral Input/Output Emulator emulates peripheral types supported by PIOS. It helps a developer to test and/or debug a mobile application that requires peripheral support without using the physical peripheral.

Implementation Considerations

- You can test your application without having to connect any peripheral to the system you are working on.
- The Emulator is supported in the Windows XP and 2000 platforms.

Integration

The Peripheral I/O Emulator is part of the Mobile Infrastructure perspective of the SAP NetWeaver Developer Studio.

Components

The emulator main window has several areas:

- **Hierarchical menus** - Group related program functions within each menu category. The categories are:
    - File - Related to project's data. It saves, opens, and creates a new project.
    - Edit - This menu option is peripheral specific. Refer to the appropriate peripheral type for the menu description.
    - View - Sets the focus of the application to the project tree or peripheral panel. It also allows the user to hide or show the project tree.
    - Peripheral Options - This menu option is peripheral specific. Refer to the appropriate peripheral type for the menu description.
    - Help - Displays version information of the Emulator.
- **Toolbars** - Each toolbar provides access to frequently used functions in the menus. The toolbars are:
    - Project Toolbar -Provides shortcuts to functions inside the File menu.
    - Peripheral Data Toolbar - Provides shortcuts to functions in the Edit menu. (It is disabled for the printer.)
    - Peripheral Toolbar - This toolbar is peripheral specific. Refer to the appropriate peripheral type for the menu description.
- **Project Tree** - Contains an entry for each installed Peripheral Type. It is located on the left side of the screen, below the toolbars.
- **Peripheral Panel** - Information on this panel depends on the selected peripheral type. It is located on the right side of the screen, to the right of the Project Tree.

Accelerators have been provided for some menu items. Accelerators are key combinations (Ctrl + AnyKey) which substitute a mouse command.

Each selectable menu item and prompt has a mnemonic. These mnemonics are shown by pressing the ALT key.

If you are using Microsoft Windows XP or Windows 2000 you can enable the mnemonics by going to Control Panel - Display settings and un-checking, in Windows XP, “Hide underlined letters for keyboard navigation until I press the Alt key” or, in Windows 2000, “Hide keyboard navigation indicators until I use the Alt key”.

Constraints

- The Emulator restricts emulated functions to those supported by the PIOS Client API. Actual peripheral hardware may have more functions not included in the Emulator.
- Settings in the Emulator are stored by user. If a developer makes changes in the Emulator for a project, the changes are in effect the next time the Emulator runs, even if it is in a different project.
- Image sizes vary according to the printer resolution. As the resolution goes higher the image size reduces.
- Differences in the screen resolution and the resolution being emulated of the emulator may result in skew appearance of text, barcode and images.

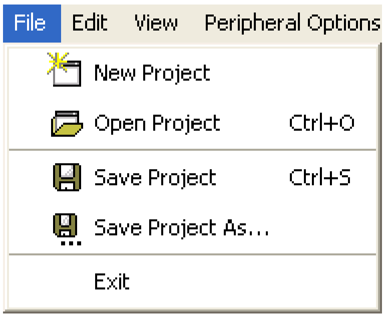
PIOS Emulator Menu Options

This section describes the PIOS Emulator Menu options.

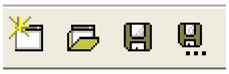
File Menu

Definition

The File menu is used to create, open, and save a project. It also has the *Exit* function to close the Emulator.



The Project Toolbar also provides access to these functions:



Use

File Menu Functions

Function	Accelerator	Description
New Project		Creates a new project.
Open Project	CTRL + O	Opens an existing project file.
Save Project	CTRL + S	Saves the current project to a file named as the project.
Save Project As...		Saves the current project to a file under a name defined by the user.
Exit		Closes and exits the emulator.

Edit Menu

Definition

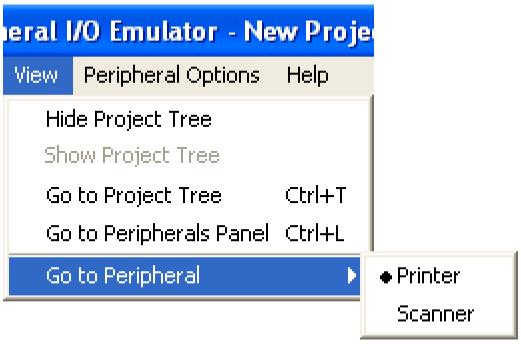
The Edit menu depends on the selected peripheral type. Refer to the appropriate peripheral:

- Edit Menu for Printer Peripheral Type [Page 106]
- Edit Menu for Scanner Peripheral Type [Page 130]
- Edit Menu for the RFID Peripheral Type [Page 141]

View Menu

Definition

The view menu allows setting the focus to the different panels of the Peripheral I/O Emulator. The focus can be changed quickly to the project tree, the peripheral panel, or you can select a different peripheral type from the available peripheral types, thus changing the peripheral panel and options.



Use

View Menu functions

Function	Accelerator	Description
Hide Project Tree		Hides the project tree.
Show Project Tree		If hidden, it shows the project tree.
Go to Project Tree	CTRL + T	Sets the focus on the Project Tree.
Go to Peripherals Panel	CTRL + L	Sets the focus on the Peripherals Panel.
Go to Peripheral		Sets the focus on the Peripherals Panel and changes the current peripheral to the selected one.

Peripheral Options Menu

Definition

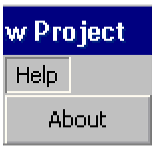
The Peripheral Options menu depends on the selected peripheral type. Refer to the appropriate peripheral:

- Peripheral Options Menu for Printer Peripheral Type [Page 106]
- Peripheral Options Menu for Scanner Peripheral Type [Page 131]
- Peripheral Options Menu for RFID Peripheral Type [Page 142]

Help Menu

Definition

The Help menu provides version information of the Emulator.



Use

Help Menu functions

Function	Description
About	Displays version information about the Emulator and available peripheral types.

Printer Peripheral Panel

Definition

The printer peripheral panel contains the virtual printout area, printer buffer, image buffer and the state buffer. These are further discussed below.

Structure

- **Virtual Printout Area** - Emulates the virtual media. It displays images, text, and/or barcodes that are printed. This area is located on the left side of the Print Buffer, the Image Buffer, and the State Buffer.
    - **Measuring String -** used to measure the distance between two points inside the Virtual Printout Area. (Refer to Using the Measuring String [Page 127] for details)

  - **Unit button -** used to toggle the reference rulers found on the top and left-hand side of the virtual printout area from inches, points and centimeters. It is located at the top left corner of the printer peripheral panel.

    - Any barcodes displayed in the virtual printout are for visual representation purposes only and are not valid barcode representation. Only barcode dimensions and position are emulated
    - Barcode size and font size are an approximation.
    - Images are displayed as black boxes which approximate image size.

- **Print buffer** - Displays the queue of commands sent to the printer. It also keeps a history of commands already printed. This buffer is located at the top right corner of the printer peripheral panel.
- **Image buffer** - Displays images loaded in the emulated image buffer. Located below the Print Buffer.
- **State Buffer** - Displays current emulated printer settings and status: Located below the Image Buffer.
  - Status - Online or Offline.
  - Connection - Open or Closed.
  - Mode - Line or Graphic
  - Resolution - Current emulated resolution in dot per inch (DPI).
  - Print Head - Displays the current position of the printer head.
  - Line Space - Displays the space between lines in points. This field is only relevant in Line Mode.
  - Paper Count - Displays the paper count for non-continuous media. For continuous media this field is blank.
  - Paper Length - Displays the paper length for continuous media. For non-continuous media this field is blank.
  - Media - Displays the media type in use by the emulator.
  - Default Font - Displays the default font type, and size.

Edit Menu for Printer Peripheral Type

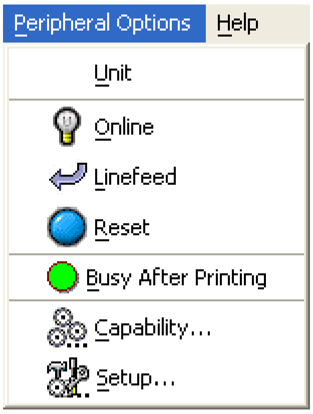
Definition

The Edit menu option is not available for the printer peripheral type.

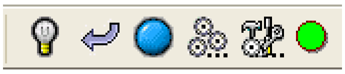
Peripheral Options Menu for Printer Peripheral Type

Definition

This menu provides access to functions that set and modify the printer emulation behavior. Each function is explained below.



There is also a toolbar, the Peripheral Options Toolbar, which relates directly to this menu:



The relationship between each button and the menu item is further explained in the next table.

Use

There are two buttons that are status indicators. These indicate if the printer:

- is Online or Offline
- returns a Busy or Ready status

The status indicators are explained below and then the function each of the buttons perform.

Status Indicators

Icon	Status Description
💡	The printer is online. It is equivalent to having a printer turned on, and connected to the device.
💡	The printer is offline. It emulates a printer that is turned off or/and disconnected from the device.
●	If queried, the printer will return a "ready" status after a print job. Another job can be sent to the printer.
⏳	If queried, the printer will return a "busy" status after a print job. If another job is sent to the printer, an exception will be thrown.

Peripheral Options menu functions for the printer

Menu option	Button	Description
Unit	Unit	Toggles the ruler unit of measure in the peripheral panel from points to centimeters, to inches, to points again.
Online	💡	Turns the printer status to Online. After clicking the button, its icon will change to the icon in the next line.
Offline	💡	Turns the printer status to Offline. After clicking the button, its icon will change to the icon in the previous line.
Linefeed	↩	Moves the printer cursor forward by one line (10 points). 💡 This function is only available for the continuous media type.
Reset	●	Resets the printer. Clears the printer buffer, image buffer, virtual printout area, and closes the connection. Sets the *Print Head* property value in the *State Buffer* to (0.0,0.0).
Capability...	⚙	This function is only available while the printer is Offline. It provides access to printer configuration attributes in each of the available categories. These categories are explained in the structure section below. 💡 The PIOS Emulator emulates attributes supported by the Printer API [Page 18].
Setup...	⚙	This function is only available while the printer is Offline. It allows set up of the media, font, and printer resolution.
Busy After Printing	●	Turns on the "Busy" status after the next print job. The button icon will change to the icon in the next line. If an application requests the printer status after this button has been pressed, it will receive a busy status. If a print job is sent to the printer, an exception will be thrown.
Recover From Being Busy	⌛	Turns off the "Busy" status after the next print job; meaning the printer will recover and return a "ready" status. The button icon will change to the icon in the previous line. If an application requests the printer status after this button has been pressed, it will receive a "printer ready" message.

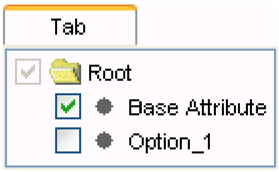
Structure

When you click on *Capability…* a window with several tabs is displayed. Each tab is explained in the next table:

Capability... Categories

Category	Description
General	Contains general printer attributes. The state of each attribute can be modified by clicking on the attribute node selection checkbox or selecting the node and pressing the space bar.
Barcode	Contains a list of the printer barcodes capabilities and barcode attributes. The state of each attribute can be modified by either clicking on the attribute node selection checkbox or selecting the node and pressing the space bar.
Font	Displays the available fonts for emulation. Emulation support for bitmapped, scalable fonts, or both can be selected.
Media	Displays the list of supported paper media. Selection of the type of media, *Continuous* or *Non-Continuous*, and the *Print Head Width* is possible. This automatically selects the media that match these two settings.

- Some attributes displayed in the *General* and *Barcode* tabs are grouped in folders. The first entry inside these folders is referred to as the **base attribute**. This base attribute is automatically selected if any of the other options within the folder is selected. If the base attribute is unselected any other selected entries within the folder are cleared as well.


Tab

☑ Root
  ☑ ● Base Attribute
  ☐ ● Option_1

Under the *Barcode* tab, the Code 39 symbology allows to uncheck *Standard*. However, this is not a real case because whenever a printer supports Code 39, the Standard codeset is supported.

- Selecting a folder will select all attributes in that folder.

Setup... Options

When you click on Setup… in the peripheral options menu or toolbar a window with several tabs is displayed, each tab is explained below:

Option	Description
Media	Select the media (paper size) that the virtual printer will use. Only the ones selected in the Capability Media category will be available for selection in this window. **Paper Length / Count -** For continuous media, the paper length can be specified in either inches or centimeters. For non-continuous media, the paper count can be specified. **Size -** Displays the size (width and length) of the media in inches. Note: For continuous paper, the length is represented as a tilde (~). **Margin -** Displays the top, left, bottom and right margins in inches. **Pre-print -** Displays the width, length, horizontal, and vertical separation of pre-printed media. This is often used to represent labels on a paper backing. *Margin* and *Pre-print* fields are only showed when there are pre-defined areas in the media.
Default Font	Select default font type, style, and size for the virtual printer. **Type -** Displays the type (Bitmapped, Scalable) of the default font selected. **Style -** Select the default font style (PLAIN, BOLD, ITALIC, BOLD+ITALIC) **Size -** Select the default font size from the list of available sizes.
Resolution	Select the desired resolution for the virtual printer from the list.

Integration

The peripheral options menu contains options specific to an individual peripheral type. Menu options will vary per peripheral type.

Emulator Configuration

This section describes the PIOS Emulator configuration options.

Property Files for Printer Peripheral Type

Purpose

The emulator property files offer the option to add or remove available fonts and media from the emulator.

Each property file may have two versions, a default and a user specific version. The default file has a DFT extension, for example "*filename.dft*". The second file has an extension of ".username", that is “*filename.username*”. The extension "username" refers to the logon name of the user.

The default versions are created the first time the Emulator runs. The user specific versions are created when the property files are modified through the *Capability…* and/or *Setup…* windows.

Features

All property files follow the Java Property file format:

- The line starting with “#” is a comment line.
- Other lines follow the “Key=Value” format.

  If a Key has a space as part of the name, for example “Courier New”, a backslash (\) must be inserted before the space. The Key in the file will look like:

  ```
  Courier\ New
  ```

- When a list is assigned to a Key the values are separated by a coma and no spaces are allowed between the coma and the value. For example, Key1=value1,value2,value3, and so on.

The property files are located in:

ECLIPSE_HOME\plugins\com.sap.ip.me.mdk.pios.docgen_X.Y.Z\emulator\inst\Printer

Where ECLIPSE_HOME refers to the location in the hard disk where Eclipse is installed and X.Y.Z refers to the current version, for example 1.1.0.

Constraints

- The system does not validate for appropriate parameters values. It is strongly recommended to make a backup before modifying a property file.
- Invalid key values may cause unexpected behavior in the emulator. For example, if in *InstalledFontFamilies* a font is added with a space between the coma and the font name, that is “, font”, the emulator will start the JVM but no window is displayed.

- The Emulator loads the configuration files when it is started. The Emulator must be restarted whenever a file is modified for the changes to take effect. Otherwise, changes to the files will take effect the next time the Emulator is started.
- Fonts and their respective styles and sizes must be available (installed) in the OS.
- An MI application can only use an installed font if its configuration can be found in the font configuration file. A font can be installed in the Emulator only if it is available in the operating system and its added to the *InstalledFont* file. The font configuration must be added separately in the *pipremulator.fntcfg* file to be available for use from the mobile application.

Installing a Font in the Emulator Printer

Use

To add a font to the available fonts you should modify the file named *InstalledFont.username*.

This file stores the following information:

- Font capabilities of the emulated printer.
- Installed fonts and their details.
- The default font of the emulated printer - This font is only used for displaying the barcode human readable.

The structure of the file is as follows:

Parameter	**Description**
DefaultFont.Size	The size of the default font. It is specified in unit of measure “point”. It must be listed in the available sizes of the font.
bitmappedFont	Supports bitmapped fonts: 1 - yes 0 - no
DefaultFont.Family	The font family name of the default font. It must be in the list of installed font families. **Note:** it also must conform to the font types that the emulated printer supports, bitmapped or scalable.
scalableFont	Supports scalable fonts: 1 - yes 0 - no
InstalledFontFamilies	The list of installed font family names. The name must be a valid font family name installed in the Operating System where the Printer Emulator is running.
DefaultFont.Style	The style of the default font. It must be included in the available styles of the font.

Font Details: (for a font named *fontname*)

Key	**Description**
fontname.Sizes	A list of available sizes of the font family, with values separated by comma. The unit of measure for each size is “point”.
fontname.Styles	A list of available styles of the font family separated by comma. Possible values are: 0 : plain 1 : **bold** 2 : *italic* 3 : ***bold and italic***
fontname.Type	Font type possible values are: 1: bitmapped font type 2: scalable font type

Adding a font to this file is the equivalent of loading a font into the printer’s memory (in the physical printer). The client API will not be able to use the added font unless it is added to the available font configurations. (Refer to Installing a Font Configuration [Page 122] for more information.)

Prerequisites

- The new font must be a valid font installed in the Operating System on which the Printer Emulator is running.

Procedure

Read Property Files for Printer Peripheral Type [Page 110].

1. Make a backup copy of the file named *InstalledFont.username*, where username refers to the logon name of the user.

   The developer can also modify the default configuration by editing the file named *InstalledFont.dft*. For the changes to take effect in the developer’s emulator, delete the file named *InstalledFont.username*. The next time the Emulator opens, it will be configured based on the default file.

   Changes made to the default file affect any user who launches the Emulator for the first time after these changes are done.

2. Open the file named *InstalledFont.username* in a text editor.
3. Add the font family name to list of supported font family names. (Refer to before and after example below.)
4. Add all three font detail lines: *Sizes*, *Styles*, and *Type*. (Refer to before and after example below.)

Result

The new font is installed in the Emulator.

Example

The following is an example of the file. The sequence of the lines has been adjusted and blank lines have been added for ease of reading.

```
#
#Wed Jun 23 11:29:33 EDT 2004
bitmappedFont=1
scalableFont=1

InstalledFontFamilies=SansSerif,Arial

SansSerif.Sizes=8.0,10.0,12.0,14.0,16.0,18.0,20.0,22.0,24.0
SansSerif.Styles=0,1,2,3
SansSerif.Type=2

Arial.Sizes=8.0,12.0,16.0
Arial.Styles=0,1
Arial.Type=1

DefaultFont.Family=Arial
DefaultFont.Style=0
DefaultFont.Size=8.0
```

Below is an example of the same file after a new font, Batang, was added. For visualization purposes the added font lines have been colored **red**.

Remember to verify that the font is installed in the Operating System.

```
#
#Wed Jun 23 11:29:33 EDT 2004
bitmappedFont=1
scalableFont=1

InstalledFontFamilies=SansSerif,Arial,Batang

SansSerif.Sizes=8.0,10.0,12.0,14.0,16.0,18.0,20.0,22.0,24.0
SansSerif.Styles=0,1,2,3
SansSerif.Type=2

Arial.Sizes=8.0,12.0,16.0
Arial.Styles=0,1
Arial.Type=1

Batang.Sizes=8.0,10.0,12.0,14.0,16.0,18.0,20.0,22.0,24.0
Batang.Styles=0,1,2,3
Batang.Type=2

DefaultFont.Family=Arial
DefaultFont.Style=0
DefaultFont.Size=8.0
```

Uninstalling a Font from the Emulator Printer

Use

To uninstall a font from the Emulator you must manually edit the file called *InstalledFont.username.*

Refer to the section called Installing a font in the Emulator Printer [Page 111] for the overall file structure.

Procedure

1. Make a backup copy of the file named *InstalledFont.username*, where username refers to the logon name of the user.

   The developer can change the default file by selecting the *InstalledFont.dft* instead of *InstalledFont.username*. Delete the file named *InstalledFont.username* for these changes to be reflected in the configuration of the developer.

   Changes made to the default file affect any user who launches the Emulator for the first time after these changes are done.

2. Open the file named *InstalledFont.username* in a text editor.
3. Remove the font family name from the list of supported font family names and delete all the applicable font detail lines.

   If the default font family is the one to be deleted, modify the default font to an available font family.

   The Printer Emulator will not perform properly if all fonts are uninstalled.

Result

The font is uninstalled.

Installing New Media to the Emulator Printer

Use

To add media to the Emulator you must edit the file named *SupportedMedia.username*.

This file stores the following information:

- Media capabilities of the emulated printer.
- Installed media and its details.
- Current default media settings.

A description of the file is given below:

Key	**Value Description**
CurrentPrintHeadWidth	It is set to the value of the selected print head width. This value must be listed in SupportedPrintHeadWidth. This value can also be set via the *Printer Capability…* window: *Media* tab → *Print Head Width* dropdown box
Continuous	1: the emulated printer will use continuous paper. 0: the emulated printer will use non-continuous paper. The emulated printer can only have one type of media active at any given time. This value can also be set via the *Printer Capability…* window: *Media* tab → *Type* dropdown box
PaperLength	This value is used to set the length of the continuous media. The valid range for this key starts at 163 cm/64 inches and ends at 1626 cm/640 inches. This value can also be set via the *Printer Setup…* window: *Media* tab → *Paper Length* text box This text box is displayed only if the continuous media is selected as the type (in *Printer Capability…)*.
SupportedMedia	Lists the media currently supported by the emulator. Any media that is added to the emulator should be added here.
CurrentMedia	It is set to the name of the current media loaded. This value can also be set via the *Printer Setup…* window: List on the left under the *Media* tab.

SupportedPrintHeadWidth	Lists all the print head widths the emulator supports.
PaperCount	The number of pages for non-continuous media. This key supports a minimum value of 1 and goes up to 1000. 💡 This value can also be set via the *Printer Setup…* window: *Media* tab → *Paper Count* text box This text box is displayed only if the non-continuous media is selected as the type (in *Printer Capability…)*.
PaperLengthDispalyUnit	Sets the unit to be used for the paper length. This can have one of two values: 0 means that paper length will be given in inches. 1 means that paper length will be given in centimeters. 💡 This value can also be set via the *Printer Capability…* window: *Media* tab → *Unit System* dropdown box

Media details: (for a media named *medianame*)

Key	**Value Description**
medianame.Size	**Description:** This value contains the media dimensions in inches. **Required:** Yes **Format:** The format of this key is “width, height”. The unit of measure for this value is “inches”. **Further Notes:** It is important to know that continuous media has zero as its length.
medianame.Margin	**Description:** Holds the margin between printable area and the edge of the media. **Required:** No (optional) **Format:** It is specified in format of “top, left, bottom, right” and the unit of measure is “inches”. **Further Notes:** Although it is visible on the virtual printout area, it is not enforced by the emulated printer. It is just a visual help. This key is only supported for non-continuous media.
medianame.Preprint	**Description:** Used for predefined areas settings in the media, like labels in the media, for example. **Required:** No (optional) **Format:** It is specified in format of “columns, rows, horizontal gap, vertical gap” in the unit of measure of “inches”. The columns and rows entries must be integer values.

	Further Notes: Like the margin, it is just another visual help. The Emulator will calculate the width of the column and the height of the row. This key is only supported for non-continuous media. ⚠ The use of this key will be discontinued. The "Label" key should be used in its place.
medianame.Label	**Description:** Used for predefined areas settings in the media, like labels in the media, for example. Required: No (optional) Format: It is specified in format of "label width, label height, vertical gap, columns" and the unit of measure is "inches". The columns entry must be an integer value. Further Notes: Like the margin, it is just another visual help. The Emulator will calculate the width of the column. 💡 This key should be used instead of the Preprint key.

Procedure

1. Make a backup copy of the file named *SupportedMedia.username*, where username refers to the logon name of the user.

   💡

   The developer can modify the default configuration by editing the file named *SupportedMedia.dft*. For the changes to take effect in the developer's emulator, delete the file named *SupportedMedia.username*. The next time the Emulator opens it will be configured based on the default file.

   ⚠

   Changes made to the default file affect any user who launches the emulator for the first time after these changes are done.

2. Open the file named *SupportedMedia.username* in a text editor.
3. Create a descriptive name for the media.
4. Add the name to list of supported media.
5. Create detail lines for the media. The only media detail line required is *Size*. If desired *Margin* and *Preprint* may be added for non-continuous paper. (Refer to before and after example below.)

Result

The media has been added to the emulator printer.

Example

The following is a sample of the file. The sequence of lines has been adjusted and blank lines are added for ease of reading.

```
#
#Thu Jul 15 11:33:43 VET 2004
continuous=0
CurrentMedia=A4
CurrentPrintHeadWidth=8.5
PaperCount=100
PaperLength=320
PaperLengthDispalyUnit=0
SupportedMedia=Letter,Legal,A3,A4,B4,B5,Plain11x17,Plain4x8,2x4_in_4x
8,Continuous_1,Continuous_2,Continuous_3,Continuous_4,Continuous_8.5
SupportedPrintHeadWidth=1,2,3,4,8.27,8.5,12

2x4_in_4x8.Margin=0.2,0.2,0.2,0.2
2x4_in_4x8.Preprint=2,4,0.05,0.05
2x4_in_4x8.Size=4,8

A3.Size=11.69,16.54
A4.Size=8.27,11.67
B4.Size=10.12,14.33
B5.Size=7.17,10.12
Continuous_1.Size=1,0
Continuous_2.Size=2,0
Continuous_3.Size=3,0
Continuous_4.Size=4,0
Continuous_8.5.Size=8.5,0
Legal.Size=8.5,14
Letter.Size=8.5,11
Plain11x17.Size=11,17
Plain4x8.Margin=0.2,0.2,0.2,0.2
Plain4x8.Size=4,8
```

Below is the file after the new media has been added. For visualization purposes the changes are colored **red**.

```
#
#Thu Jul 15 11:33:43 VET 2004
continuous=0
CurrentMedia=A4
CurrentPrintHeadWidth=8.5
PaperCount=100
PaperLength=320
PaperLengthDispalyUnit=0
SupportedMedia=Letter,Legal,A3,A4,B4,B5,Plain11x17,Plain4x8,2x4_in_4x8,Continuous_1,Continuous_2,Continuous_3,Continuous_4,Continuous_8.5,New Media
SupportedPrintHeadWidth=1,2,3,4,8.27,8.5,12

2x4_in_4x8.Margin=0.2,0.2,0.2,0.2
2x4_in_4x8.Label=2.0,4.0,0.05,2
2x4_in_4x8.Size=4,8

New\ Media.Margin=0.2,0.2,0.2,0.2
New\ Media.Label=2.0,4.0,0.05,2
New\ Media.Size=8,10

A3.Size=11.69,16.54
A4.Size=8.27,11.67
B4.Size=10.12,14.33
B5.Size=7.17,10.12
Continuous_1.Size=1,0
Continuous_2.Size=2,0
Continuous_3.Size=3,0
Continuous_4.Size=4,0
Continuous_8.5.Size=8.5,0
Legal.Size=8.5,14
Letter.Size=8.5,11
Plain11x17.Size=11,17
Plain4x8.Margin=0.2,0.2,0.2,0.2
Plain4x8.Size=4,8
```

Uninstalling Media from the Emulator Printer

Use

To uninstall media from the emulator the file *SupportedMedia.username* must be edited manually.

> Refer to Installing New Media to the Emulator Printer [Page 116] for an overview of the structure of the file.

Procedure

1. Make a backup copy of the file named *SupportedMedia.username*, where username refers to the logon name of the user.

   > The developer can change the default file by selecting the *SupportedMedia.dft* instead of *SupportedMedia.username*. Delete the file named *SupportedMedia.username* for these changes to be reflected in the configuration of the developer.

   > Changes made to the default file affect any user who launches the Emulator for the first time after these changes are done.

2. Open the file named *SupportedMedia.username* in a text editor.
3. Remove all detail lines for the media to be removed.
4. Remove the media name from the list of supported media.
5. If the current media name is no longer in the list, set it to one that is still available.
6. The Printer Emulator will not perform properly if all media is uninstalled.

Result

The media is uninstalled.

Installing a Font Configuration

Use

The Client API uses on a font configuration file containing a list of all fonts known by the API. These fonts must be supported by the physical/emulated printer. Installing a new font on the Emulator (Refer to Installing a Font in the Emulator Printer [Page 111]) is analogous to installing a font on a physical printer. The font configuration file must be modified before the Client API recognizes the new font. Configuration changes to the drivers are done via the MI Configuration Tool. Configuration changes to the emulator are done by manual modification of the configuration files.

The file named *pipremulator.fntcfg* must be edited to modify the emulator font configurations. This file contains the name, description, options, font type, and size. Even though the file can be edited programmatically, a developer may decide to modify the file manually for the emulation of one specific printer hardware.

This file is located in:

```
ECLIPSE_HOME\plugins\com.sap.ip.me.mdk.pios.docgen_X.Y.Z\emulator\pio
s\
config\pipremulator.fntcfg
```

Where ECLIPSE_HOME refers to the location in the hard disk where Eclipse is installed and X.Y.Z refers to the plugin version, for example 1.1.0.

This file does not exist the first time the emulator runs. It is created and read when the Printer API opens the connection to the Emulator.

This file is read during the opening of the connection to the printer. If a change is made after the connection has been opened and before the connection is closed, changes will take effect the next time the connection opens.

The file structure is explained below:

File Variables

Variable	**Description**
Configs	Lists the font configurations available to the PIOS Client API.

Fonts Parameters

A font configuration called *fontcfgname* needs the options explained below. Font configurations must be listed in the Configs variable mentioned above.

Parameter	**Description**
fontcfgname	This is left blank, but the line must be included in the file.
*fontcfgname.*_Type	This parameter is always set to “Font“.
*fontcfgname.*Name	The name of the font.
*fontcfgname.*Description	Parameter that contains a short description of the font.

*fontcfgname.*Options	Specify the options of the font. Possible values are: 0 - Normal 2 - Bold 4 - Italic 6 - Bold and italic 8 - Underline 10 - Bold and underline 12 - Italic and underline 14 - Bold, italic, and underline
*fontcfgname.*FontType	Tells whether the font type is scalable or bitmapped. Possible values for this parameter are: 1 - bitmapped 2 – scalable
*fontcfgname.*Size	This value tells the size of the font.

Make sure font configurations added to this file match those installed in the *InstalledFont.username*. The font configurations included in this file map installed fonts in the emulator.

Procedure

2. Make a backup copy of the file named *pipremulator.fntcfg*.
3. Open the file named *pipremulator.fntcfg* in a text editor.
4. Add the font family name to list of supported configurations (*Configs*). (Refer to before and after example below.)
5. Add the font configuration parameter lines: *_Type*, *Name*, *Description*, *Options*, *FontType*, and *Size*. (Refer to before and after example below.)

Result

Font configuration is added.

Example

The following is an example of font configuration file. The sequence of lines has been adjusted and blank lines are added for ease of reading.

```
#Fri Jul 16 16:02:53 VET 2004
#Initial Installation

Configs=Bitmapped

Bitmapped=
Bitmapped._Type=Font
Bitmapped.Name=Courier New
Bitmapped.Description=plain bimapped font
Bitmapped.Options=0
Bitmapped.FontType=1
Bitmapped.Size=10
```

Below is an example of the same file after a new font configuration, ScalableBoldItalic, is added. For visualization purposes the added font lines have been colored **red**.

```
#Fri Jul 16 16:02:53 VET 2004
#Initial Installation

Configs=Bitmapped,ScalableBoldItalic
Bitmapped=
Bitmapped._Type=Font
Bitmapped.Name=Courier New
Bitmapped.Description=plain bimapped font
Bitmapped.Options=0
Bitmapped.FontType=1
Bitmapped.Size=10

ScalableBoldItalic=
ScalableBoldItalic._Type=Font
ScalableBoldItalic.Name=Batang
ScalableBoldItalic.Description=bold&italic scalable font
ScalableBoldItalic.FontType=2
ScalableBoldItalic.Options=6
ScalableBoldItalic.Size=10
```

Uninstalling a Font Configuration

Use

A font configuration can be uninstalled programmatically but a developer may decide to do it manually. To uninstall a font configuration manually from the Emulator you must manually edit the file called *pipremulator.fntcfg*.

Refer to the section called Installing a Font Configuration [Page 122] for the overall file structure.

Procedure

1. Make a backup copy of the file named *pipremulator.fntcfg*.
2. Open the file named *pipremulator.fntcfg* in a text editor.
3. Remove the font family name from the list of supported font family names and delete all the applicable font configuration parameter lines.

   The Printer Emulator will not perform properly if all font configurations are uninstalled.

Result

The font configuration is uninstalled.

Using the PIOS Emulator for the Printer Peripheral Type

Use

Provide an overview on how to use the Peripheral I/O Emulator for the printer peripheral type.

Prerequisites

- SAP NetWeaver Developer Studio is installed and configured.

Procedure

1. Write code that uses the Printer API of the MI Client API.
2. Once the code is ready for testing, set the SAP NetWeaver Developer Studio in emulation mode by pressing the *Set MDK peripheral emulation mode* button on the MI perspective.
3. Restart the Mobile Infrastructure client.
4. Launch the Emulator by pressing the *Launch the Peripheral I/O Emulator* button on the MI perspective.
5. Turn the Emulator offline by clicking on the *On/Offline* button.
6. Configure desired emulator capabilities by clicking on the *Capability…* button.
   a. Under the general tab, select the desired capabilities to be emulated.
   b. Under the barcode tab, select the desired symbology capabilities to be emulated.
   c. Under the font tab, select the desired font capabilities to be emulated.
   d. Under the media tab, select the desired media *Type* and *Print Head Width* to be emulated.
7. Configure the appropriate Emulator settings by clicking the *Setup…* button.
8.
   a. Select the media that the printer will emulate in the Virtual Printout Area and set the Paper Count/Page Length.
   b. Set the default font. This is only used for the barcode’s Human Readable attribute.
   c. Set the printing resolution for the emulator printer.
9. Set the emulator Online by clicking on the *On/Offline* button.

   Make sure the Emulator is Online by checking the *Status* property on the *State Buffer*.
10. Run your application.

Result

The application print output is displayed on the virtual printout area.

Using the Measuring String

Use

The Measuring String is used to determine the distance between two points inside the Virtual Printout Area. This distance is measured in points. It displays two numbers, the horizontal and vertical spacing.

Prerequisites

- The printer peripheral panel is displayed.

Procedure

1. Move the mouse over the Virtual Printout Area, press and hold the left mouse button over the first point, drag the mouse to the next point. A line appears between the first point and actual mouse pointer position.
2. While the mouse button remains depressed, the horizontal and vertical distance between the two points is displayed.

Result

The Measuring String displays the horizontal and vertical distance between the two points selected.

Scanner Peripheral Panel

Definition

The scanner peripheral panel can be divided in two areas: an input area and a state area. The input area contains a barcode record list of the selected data. The data list is under the scanner node in the project tree, otherwise, if the project tree is hidden, the list is in the input area above the barcode record list. The state area contains a scan history buffer, an active symbology list and a state buffer. These are further discussed below.

Structure

- **Data List -** This list is displayed under the scanner node, in the project tree. A project may have more than one data (barcode) group for the scanner.
- **Input Area -** It allows the user to prepare the barcode data that is going to be used in the scanning emulation. The data and barcodes are listed in this area.
  - **Data List -** If the project tree is hidden, the list is displayed on top of the barcodes list.
  - **Barcode List -** As indicated above, each data (group) has a list of barcodes. This list contains all barcode records that belong to the selected data. Each barcode has the following information:
    - Symbology - The symbology used by the barcode
    - Alias - An alias that helps identify each barcode. Since there can be several barcodes for a given symbology, the user can assign an appropriate identifier for each barcode.
    - Data - The data for the barcode. Depending on the symbology the data may include the check digit.
- **State Area -** It displays several states of the emulated scanner.
  - **Scan History -** All scanning results are recorded here, both successful and failed scans.
  - **Active Symbologies -** It displays every active symbology.

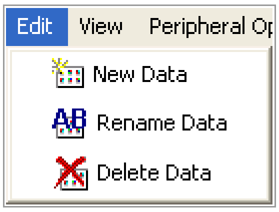
    If an active symbology tree is expanded, the window will display all of the active options under the symbology.
  - **State Buffer -** It indicates the following settings on the scanner
    - Connection - Open or Closed.
    - Mode - Wedge or Scan Aware.
    - Armed - false or true. Indicates whether the scanner is ready to read or not.
    - Listener - Ready or Not Ready. Indicates if there is a client listener or not.
    - Trigger - On or Off. Indicates if the scanner’s beam is on or off

- Wedge Window - Default or Application Title. Shows where the wedge result will go.
- UPC Addon - Required or Not Required. Indicates whether to scan a UPC/EAN barcode with or without an add-on.
- Addon 2 - Active or Inactive. Indicates whether the scanner will read the two digit add-on when it scans a UPC/EAN barcode.
- Addon 5 - Active or Inactive. Indicates whether the scanner will read the five digit add-on when it scans a UPC/EAN barcode.
- Beep - Describes the beeping behavior of the scanner. The possible values are:
  - Off
  - On Read
  - On Fail
  - On Scan
- Pre-amble - This is a string that will be inserted in front of scanned barcode data.
- Post-amble - This is a string that will be appended to the scanned barcode data.

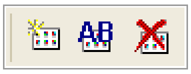
Edit Menu for Scanner Peripheral Type

Definition

The Edit menu is used to prepare input data for input peripherals such as the scanner. It allows the user to manipulate data groups.



The peripheral data toolbar provides access to the same functions.



Use

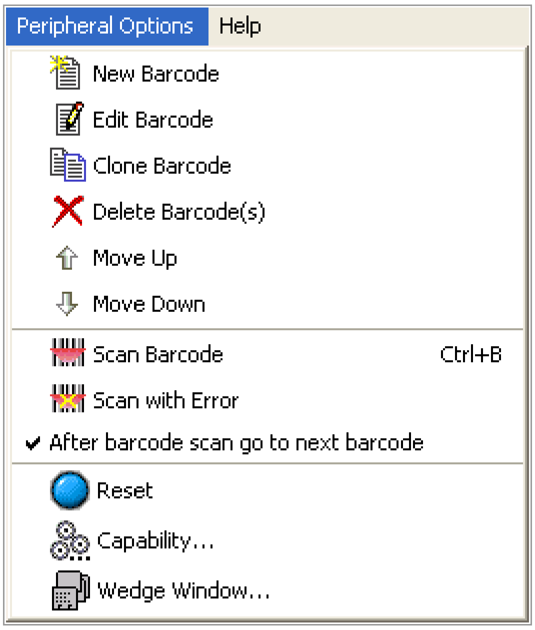
Edit menu options for the scanner

Menu option	**Button**	**Description**
New Data		Creates and names a new group of data
Rename Data	AB	Renames an existing data group
Delete Data	✗	Deletes an existing data group. All barcode data under the group will be lost.

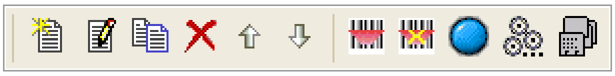
Peripheral Options Menu for Scanner Peripheral Type

Definition

This menu provides functions to prepare barcode data, set the emulation behavior, and emulate scanning. Each function is explained below.

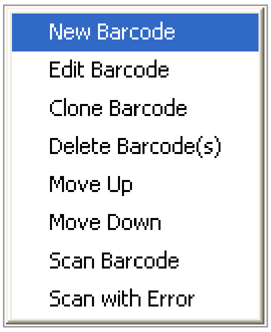


There is also a toolbar, the Peripheral Options Toolbar, which relates directly to this menu:



The relationship between each button and a function from the menu is explained in the next table.

Some of these functions are also available in a popup menu which can be triggered by right clicking on the input area of the scanner peripheral panel.



Use

Peripheral Options menu functions for the scanner

Menu option	Button	Description
New Barcode	📄	Only available after a Data group has been selected. Allows the user to create a new barcode by entering all the necessary information.
Edit Barcode	📝	Only available after an existing barcode has been selected. Allows the user to modify the information of the selected barcode.
Clone Barcode	📋	Only available when an existing barcode is selected. It duplicates an existing barcode and triggers the Clone Barcode dialog box that prompts the user to modify all information of the duplicated barcode.
Delete Barcode(s)	✗	Only available when at least one barcode is selected. Deletes all selected barcodes. - 💡 - This function supports multiple selections in the barcode list.
Move Up	⇧	This function is only available when a barcode is selected and is not in the top row of the barcode list. The selected barcode moves up one row in the barcode list each time the button is pressed.
Move Down	⇩	This function is only available when a barcode is selected and is not in the bottom row. The selected barcode moves down one row in the barcode list each time the button is pressed.
Scan Barcode	▥	Only available when a barcode is selected. Starts an emulated scanning process. The scanning may or may not be successful based on the barcode itself and the scanner's settings.
Scan with Error	▥	Only available when a barcode is selected. Emulates a scanning process that generates a hardware scanning error. When the application is in "Scan Aware Mode", this function triggers an error event.

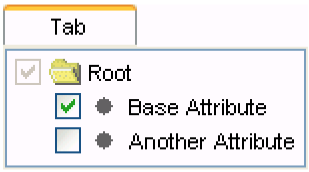
After barcode scan go to next barcode		Toggles a switch in the scanner emulator. When the switch is on, the next barcode in the list will be selected after a successful scan of the barcode currently selected.
Reset	●	Resets the scanner. It sets the scanner to its initial states.
Capability...	⚙	This function provides access to the scanner configuration attributes in each of the available categories. These categories are explained in the structure section below. • 💡 • The PIOS Emulator emulates attributes supported by the Scanner API [Page 44].
Wedge Window...	🖥	Displays the wedge window selection dialog box. Here the user may select another application as the target for the scanning result in wedge mode. This is an emulated wedge mode; a special instance of the wedge mode. The data does not go to the Emulator window, which has the focus. Instead, the data goes to the application selected in the wedge window selection dialog box. • 💡 • If no application is selected the data is sent to the Emulator's default wedge window.

Structure

When you click on *Capability…* a window with several tabs is displayed. Each tab is explained in the next table:

Capability... Categories

Category	**Description**
General	Contains general scanner attributes. The state of each attribute can be modified by clicking on the attribute node selection checkbox or selecting the node and pressing the space bar.
Symbology	Contains a list of the supported symbologies and their supported attributes. The state of each attribute can be modified by either clicking on the attribute node selection checkbox or selecting the node and pressing the space bar.

- Some attributes are grouped in folders. The first entry inside these folders is referred to as the **base attribute**. This base attribute is automatically selected if any of the other options within the folder is selected. If the base attribute is unselected any other selected entries within the folder are cleared as well.
  -
  - 
    Tab
    ☑ Root
    ☑ Base Attribute
    ☐ Another Attribute
- Selecting a folder will select all attributes in that folder.

Integration

The peripheral options menu contains options specific to an individual peripheral type. Menu options will vary per peripheral type.

Using the PIOS Emulator for the Scanner Peripheral Type

Use

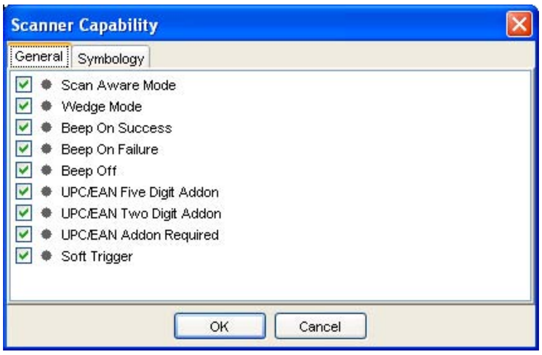
Provide an overview on how to use the Peripheral I/O Emulator for the scanner peripheral type.

Prerequisites

- SAP NetWeaver Developer Studio is installed and configured
- Prepared barcode data. (see Creating Barcode Data [Page 136])

Procedure

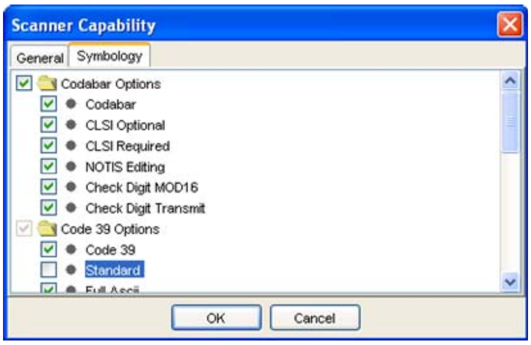
1. Write code that uses the scanner API of the MI Client API.
2. Once the code is ready for testing, set the SAP NetWeaver Developer Studio in emulation mode by pressing the *Set MDK Peripheral Emulation Mode* button on the MI perspective.
3. Restart the Mobile Infrastructure client.
4. Launch the Emulator by pressing the *Launch the Peripheral I/O Emulator* button on the MI perspective.
5. Configure desired emulator capabilities by clicking on the *Capability…* button.



- ☑ ● Scan Aware Mode
- ☑ ● Wedge Mode
- ☑ ● Beep On Success
- ☑ ● Beep On Failure
- ☑ ● Beep Off
- ☑ ● UPC/EAN Five Digit Addon
- ☑ ● UPC/EAN Two Digit Addon
- ☑ ● UPC/EAN Addon Required
- ☑ ● Soft Trigger

In this example, the scanner:

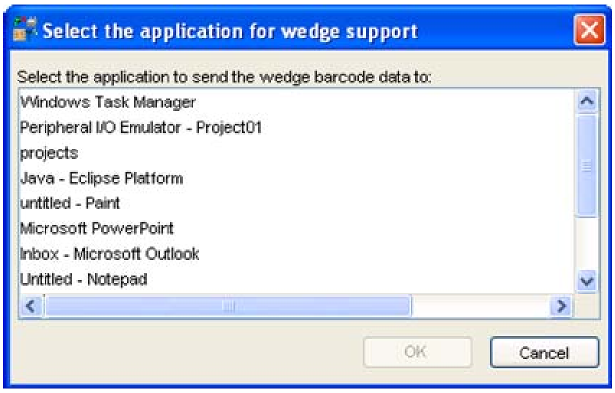
1. Can be opened with either wedge or scan_aware mode.
2. Can configure all UPC/EAN add-on options.



- ☑ Codabar Options
  - ☑ ● Codabar
  - ☑ ● CLSI Optional
  - ☑ ● CLSI Required
  - ☑ ● NOTIS Editing
  - ☑ ● Check Digit MOD16
  - ☑ ● Check Digit Transmit
- ☑ Code 39 Options
  - ☑ ● Code 39
  - ☐ ● Standard
  - ☑ ● Full Ascii

1. Codabar can be added as active symbology with all the options.
2. Code128 can be added as active symbology, but its Check Digit Transmit option is not configurable.

6. Run application to open the connection, add symbologies and their options, initialize other settings, arm the scanner and get into an idle state to wait for scanning results.
7. In wedge mode, if user wants to specify a different target wedge application than the default (emulator itself), user can select it by clicking *Wedge Window…* button.



Select the application to send the wedge barcode data to:

- Windows Task Manager
- Peripheral I/O Emulator - Project01
- projects
- Java - Eclipse Platform
- untitled - Paint
- Microsoft PowerPoint
- Inbox - Microsoft Outlook
- Untitled - Notepad

The list contains all running applications with their titles.

A user can go back to the default setting by selecting the Emulator itself (“Peripheral I/O Emulator - *ProjectName*”).

8. Scan prepared barcodes. (see [Scanning Existing Barcode Data [Page 139]])

Result

All the scan results are recorded under Scan History.

The formatted scanned results are either in a wedge window or passed to the application for processing. Formatted results are those that have been processed; for instance, the preamble and postamble have been added to the barcode data.

Creating Barcode Data

Use

Each project has only one scanner node. The scanner node may have more than one data node (data group) and each data node may have multiple barcodes, displayed in the input area under "Barcodes:". Each barcode has information such as symbology, alias, data, and check digit.

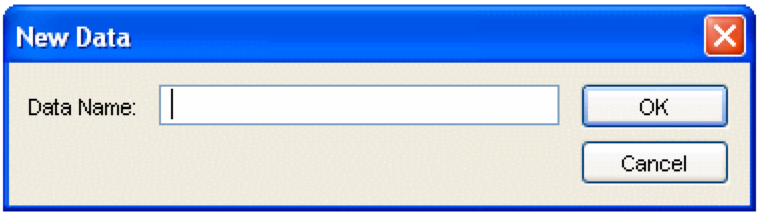
Because the scanner emulator cannot scan real barcodes, all barcode data must be entered into a project before doing any scanning emulation.

Prerequisites

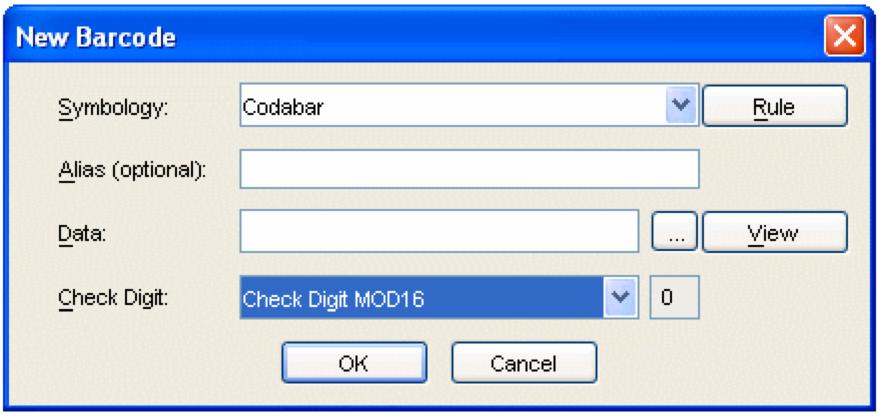
The scanner peripheral panel is displayed.

Procedure

1. Create a new Data group by pressing the button. The New Data dialog box will be displayed and prompt the user to input the name of this Data group.



2. Create a barcode record for the selected Data group by pressing the button. The New Barcode dialog box is launched and prompts to input information for the barcode.



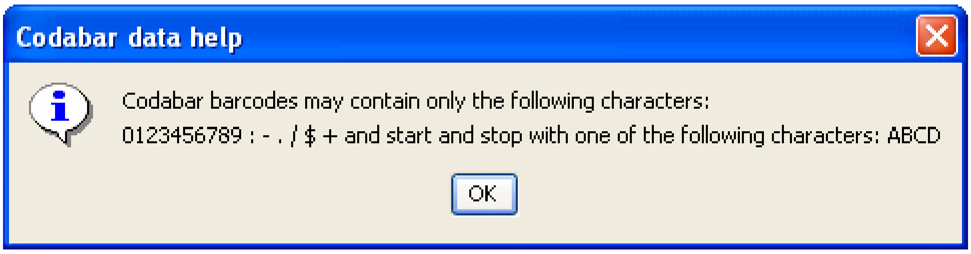
Symbology: Codabar

Check Digit: Check Digit MOD16 0

   a. Select the required Symbology.

      Each symbology has its own set of rules for data. You can press the Rule button to show the data rules.

      Below is the rule window for Codabar:



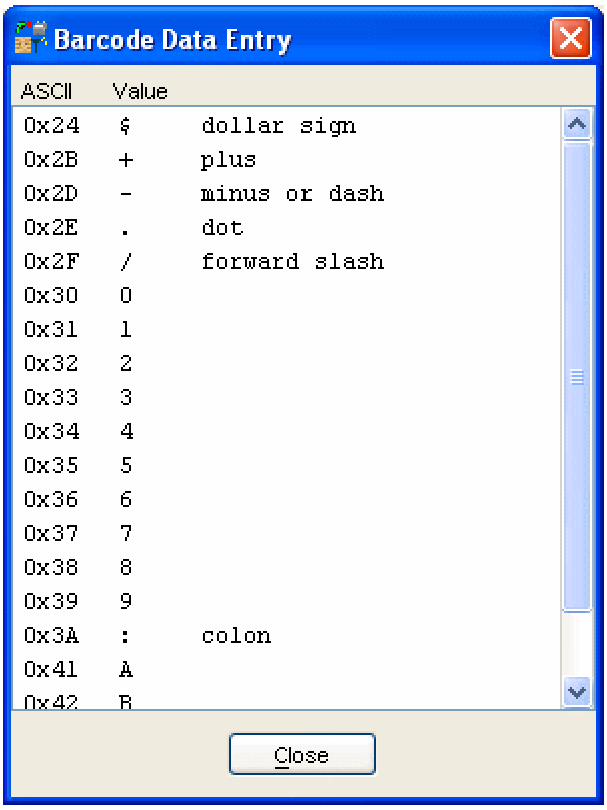
Codabar barcodes may contain only the following characters:
0123456789 : - . / $ + and start and stop with one of the following characters: ABCD

b. Input Alias (optional). This will help identify each barcode.

c. Input the barcode data.

The user can press the ... button to launch the Barcode Data Entry dialog box and get help entering valid characters. The Barcode Data Entry displays all of the supported (valid) characters for the selected symbology. The user can select any character, including non-printable characters, and press the Enter key to send the character to the Data field in Barcode dialog. The same result can be accomplished by double clicking on the desired entry. The target position of the chosen character, in the data field, is at the last cursor position when the ... button was clicked.

By using the ... button, the user can insert non-printable characters as part of the barcode data.

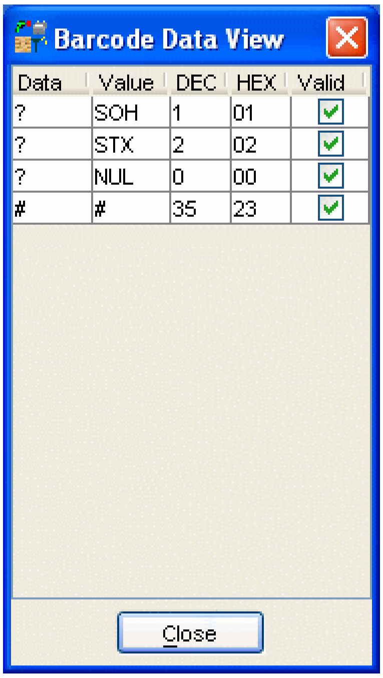


ASCII	Value	
0x24	$	dollar sign
0x2B	+	plus
0x2D	-	minus or dash
0x2E	.	dot
0x2F	/	forward slash
0x30	0	
0x31	1	
0x32	2	
0x33	3	
0x34	4	
0x35	5	
0x36	6	
0x37	7	
0x38	8	
0x39	9	
0x3A	:	colon
0x41	A	
0x42	B	

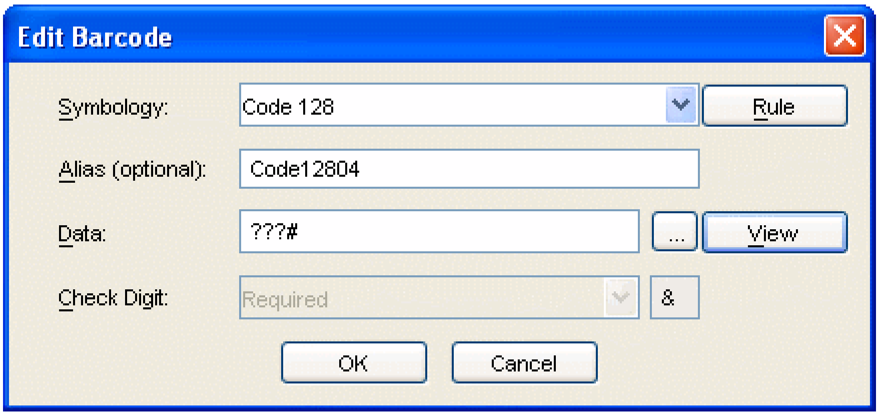
Any non-printable character will be displayed as “?” in the Data field of Barcode Dialog. To see complete information behind each character, the user can press the View button to display Barcode Data View.

The barcode data entry panel works with Unicode. If the ALT+CODE combination (where CODE refers to a number combination) is used, to input data directly from the keyboard, a 4-digit code (Unicode) is required. Remember to add a leading zero when entering ASCII characters. Otherwise, the incorrect character is produced.

The following screenshot is an example for a Code128 barcode.



Data	Value	DEC	HEX	Valid
?	SOH	1	01	☑
?	STX	2	02	☑
?	NUL	0	00	☑
#	#	35	23	☑



Symbology: Code 128

Alias (optional): Code12804

Data: ???#

Check Digit: Required

   d. Set Check Digit option if it is required. The value of the check digit will be generated automatically based on the option.

      ⚠

      The calculation of French CIP check digit for Code 39 is an approximation that may differ from the actual check digit.

   e. Press OK to save and Cancel to quit.

3. Repeat step 1 and/or 2 until all desired barcodes have been created.
4. Save the project by pressing 💾 button.

Result

All data including data list and barcode list are stored in project file.

Scanning Existing Barcode Data

Use

This procedure explains how to get existing barcode data to either a wedge window or an application's listener. Depending on the scanner's settings, the scanning may result successful or in a scanning error. An emulated scanning error emulates an error that happens at the hardware level.

All scanning actions are recorded under scan history. Each history record contains either the scanned data or an error message for the failure.

Prerequisites

- The scanner peripheral panel is displayed.
- Some barcode data has been created
- Client application has run properly to arm the scanner.

Procedure

1. Select a data from the Data list
2. Select a barcode from the Barcode list
3. Press either the button for a normal scan or the button for an emulated error scan.

Result

The scanning result is recorded under Scan History.

The formatted result is sent to either wedge window or client application for processing. Formatted results are those that have been processed; for instance, the preamble and postamble have been added to the barcode data.

RFID Peripheral Panel

Definition

The RFID peripheral panel can be divided in two areas: a tag information area and a state area. The tag information area contains a tag record list of the selected data. The data list is under the RFID node in the project tree, otherwise, if the project tree is hidden, the list is in the tag information area above the tag record list. The state area contains an *Operation History* buffer, a *Last Operated Tags* list and a *State Buffer*. These are further discussed below.

Use

- **Data List -** This list is displayed under the RFID node, in the project tree. A project may have more than one RFID data (tag) group.
- **Tag Information Area -** It allows the user to prepare tag data that is going to be used during the RFID emulation. The data and tags are listed in this area.
  - **Data List -** If the project tree is hidden, the list is displayed on top of the tag list.
  - **Tag List -** As indicated above, each data (group) has a list of RFID tags. This list contains all tag records that belong to the selected data. Each row represents an RFID tag. Each tag has the following information:
    - Tag Id - This column displays the identifier for each tag on the list. The tag ID can be from 1 to 16 characters long depending on the tag type.
    - Tag Type - Displays the tag type for the tag.
    - TID Password - Tag ID password. If the tag ID is lockable, this column displays the password used to lock the tag.
    - TID Locked - This column contains a checkbox. If checked, it means the tag ID has been locked.

    To edit the value of this column, you must select the tag and click on the *Edit Tag* button. Once the *Edit Tag* dialog box opens, click on *Tag Id Locked*.

    - Valid - This column contains a checkbox. If checked, it means the tag type is supported by the emulated RFID reader.

    The value of this checkbox is affected by either enabling/disabling a tag type (see Editing RFID Tag Types [Page 146]), or by editing a tag type and rendering the tag invalid (not fitting the under the redefined tag type description).

    - In Range - This column contains a checkbox. If checked, it means the tag is inside the range of the RFID reader.

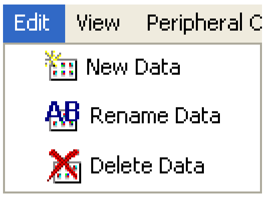
The value of this checkbox can be set by clicking on either the *Move Away* or the *Move Near* buttons.

- **State Area -** It displays several states of the emulated RFID reader.
    - **Operation History -** All operations (like write and read) are recorded here, both successful and failed operations.
    - **Last Operated Tags -** Displays the tag (s) that were accessed last by the RFID reader. The box clears as soon as the connection is closed.
    - **State Buffer -** It indicates the following settings on the RFID reader.
        - Connection - Open or closed.

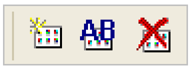
Edit Menu for the RFID Peripheral Type

Definition

The Edit menu is used to prepare data for the RFID reader. It allows the user to manipulate data groups.



The peripheral data toolbar provides access to the same functions.



Use

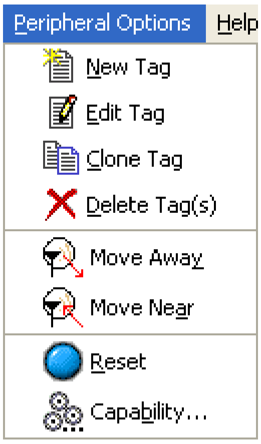
Edit Menu options for the RFID peripheral type

Menu option	Button	Description
New Data		Creates and names a new group of data.
Rename Data		Renames an existing data group.
Delete Data		Deletes an existing data group. All RFID tag data under the selected group will be lost.

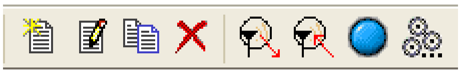
Peripheral Options Menu for RFID Peripheral Type

Definition

This menu provides functions to prepare tags, set the tags either in or out of range, and to modify the emulated RFID reader's capabilities. Each function is explained below:



There is also a toolbar, the Peripheral Options Toolbar, which relates directly to this menu:



The relationship between each button and a function from the menu is explained in the next table.

Some of these functions are also available in a popup menu which can be triggered by right clicking the mouse in the *Tags* area of the RFID peripheral panel.

Use

Peripheral Options menu functions for the RFID reader

Menu option	Button	Description
New Tag		It triggers the *New Tag* dialog to prompt the user to input all information of a new tag. This function is only available when a Data group is selected.
Edit Tag		It triggers the *Edit Tag* dialog to prompt the user to modify all information of an existing tag. This function is available only when an existing tag is selected.

Clone Tag	📋	It duplicates an existing tag and triggers the *Clone Tag* dialog to prompt user to modify all information of the duplicated tag. This function is available only when an existing tag is selected.
Delete Tag(s)	✖	It deletes selected tag(s). Please note that it supports multiple selections in the tag list. This function is available only when at least one tag is selected.
Move Away	↘	It takes the selected tags and sets them as out of range of the RFID reader. This function is available only when at least one tag is selected.
Move Near	↖	It takes the selected tags and sets them as in range of the RFID reader. This function is available only when at least one tag is selected.
Reset	●	It sets the RFID reader to its initial states.
Capability…	⚙	It triggers the *Rfid Capability* dialog to let user customize the RFID’s attributes.

Structure

When you click on *Capability…* ⚙ a window with two tabs is displayed. Each tab is explained in the next table:

RFID Reader's Capabilities

Category	**Description**
General	Contains general RFID reader attributes. The state of each attribute can be modified by clicking on the attribute node selection checkbox or selecting the node and pressing the space bar.
Tag Type	This tab provides functionality to add, edit, delete, disable, and enable tag types. The state of each attribute can be modified by selecting the tag type and pressing either *Enable* or *Disable* button.

Integration

The peripheral options menu contains options specific to an individual peripheral type. Menu options will vary per peripheral type.

Using the RFID Emulator

Use

This procedure provides an overview on how to use the Peripheral I/O Emulator for the RFID peripheral type.

Keep in mind the following when using the RFID emulator:

- Tags should have a unique ID. Only one tag with a particular tag ID can be in range at any given time. If more than one tag has the same tag ID the last one to get in range will stay in range.
- Programming several tags with the same tag ID can cause tags to be set as "out of range".
- Tags can be set in range and out of range while the application is running.

Prerequisites

- SAP NetWeaver Developer Studio is installed and configured.
- Prepared RFID tag data (see Creating RFID Tags [Page 147]).

Procedure

1. Write code that uses the RFID API of the MI Client API.
2. Once the code is ready for testing, set the SAP NetWeaver Developer Studio in emulation mode by pressing the *Set MDK Peripheral Emulation Mode* button on the MI perspective.
3. Restart the Mobile Infrastructure client.
4. Launch the Emulator by pressing the *Launch the Peripheral I/O Emulator* button on the MI perspective.
5. Configure desired emulator capabilities by clicking on the *Capability…* button.
6. Set any tags you want to work with in range by selecting the tags and clicking on *Move Near* button.
7. Run the application.
8. Perform operations (like identify, read, write and so on) on the RFID tags.

Result

All the operations are recorded under *Operation History*, whether it failed or was successful.

Adding RFID Tag Types

Use

The RFID emulator comes with a set of predefined RFID tag types. These tag types are but a fraction of all the tag types that exist today. Also, as RFID technology advances, more RFID tag types may become available. The RFID emulator provides the user with the ability to add tag types to the RFID emulator as the need arises.

This procedure explains how to add, edit, and delete tag types from the RFID emulator.

Prerequisites

The RFID peripheral panel is displayed.

Procedure

1. Click on the *Capability…* button. The *Rfid Capability* window opens.
2. Click on the *Tag Type* tab.
3. Click the *Add* button. The *New Tag Type* window opens.
4. Enter the following information:
    - a. Type Name - The name that will be given to the tag type being added.
    - b. Under the *Tag Id* tab:
        - i. Total Length - Total length of the tag ID in bytes.
        - ii. Prefix Length - Prefix length of the tag ID in bytes. These are bytes in the tag ID that are the same for all the tags that belong to this tag type.
        - iii. Prefix - Double click on each of the bytes to enter the hex code for each.

    When you double click on a byte, the Byte Entry window is displayed. This window presents three columns: *HEX* (hexadecimal code), *DEC* (decimal code), and *CHR* (character). Each row is a byte. Rows go from 0 to 255 (DEC).

        - iv. Programmable - Check this box if the Tag ID is programmable.
        - v. Lockable - Check this box if the Tag ID can be locked.
        - vi. Password Length - The length, in bytes, of the password used to lock a Tag ID.
    - c. Under the *Memory* tab:
        - i. Size - The size of the whole tag including reserved, read only, and/or writable areas.

    By default the size is in addition to the tag ID. To include the Tag ID as part of the tag type size, click on the desired starting position (Offset) for the tag ID and click on the *Include TID* button.

        - ii. Bytes Per Block - Set by default to 1. If you want to configure the number of bytes per block, keep in mind that this number must be a factor of the *Size*.
5. Click on the *OK* button.

Result

The new tag type has been added to the RFID emulator.

Editing RFID Tag Types

Prerequisites

- The RFID peripheral panel is displayed.
- Tag Types present in the RFID emulator (see Adding RFID Tag Types [Page 145]).

Procedure

1. Click on the *Capability…* button. The *Rfid Capability* window opens.
2. Click on the *Tag Type* tab.
3. Select one of the tag types.
4. Click on the *Edit* button. The *Edit Tag Type* window opens.
5. Change one or more fields.
6. When you finish making changes, click on the *OK* button.

   Changes done to a tag type may cause existing tags, for that type, to become invalid.

Result

The Tag Type has been changed.

Deleting RFID Tag Types

Prerequisites

- The RFID peripheral panel is displayed.
- Tag Types present in the RFID emulator (see Adding RFID Tag Types [Page 145]).

Procedure

1. Click on the *Capability…* button. The *Rfid Capability* window opens.
2. Click on the *Tag Type* tab.
3. Select one or more of the tag types.
4. Click on the *Delete* button. The *Confirm Delete* window opens.
5. Click on the *Yes* button.
6. When you finish making changes, click on the *OK* button.

Result

The tag type or types are deleted.

Creating RFID Tags

Use

Each project has only one RFID node. The RFID node may have more than one data node (data group) and each data node may have multiple tags, displayed in the tag information area under *Tags*. Each tag has information fields such as Tag Id, Tag Type, TID Password, TID Locked, Valid, and In Range.

Because the RFID emulator cannot work with physical RFID tags, all tag data must be entered into a project before doing any RFID emulation.

Prerequisites

- The RFID peripheral panel is displayed.
- At least one tag type is enabled.

Procedure

1. Create a new Data group by pressing the button. The *New Data* dialog box will be displayed and prompt the user to input the name of this Data group.
2. Create a tag record for the selected Data group by pressing the button. The *New Tag* dialog box is launched and prompts to input information for the RFID tag:
   a. Tag Type - Select the tag type for the new tag.
   b. Tag Id - These are the tag ID bytes.

      💡

      Red colored bytes indicate the tag ID prefix. These bytes cannot be changed.

      i. Double click on any of the bytes. The *Byte Entry* window is display.
      ii. Select any value from 1 to 255.
      iii. Click on the *Enter* button.
      iv. Repeat the process for each of the remaining tag ID bytes.

      ⚠

      If a tag has all the bytes in the tag ID set to "FF", it is treated as a blank tag.

   c. Tag Id Locked - Checkbox is enabled if tag ID is lockable. Check to lock the tag ID.
   d. Tag Id Password - If you locked the tag ID, you can set the password to reset the tag. The procedure is the same as described for the *Tag Id*.
   e. Memory Map - You can set the value of any of the Writable bytes by clicking on the *Value…* button and selecting values from the *Byte Entry* window.
3. Click on the *OK* button.

Result

A new tag is created.

Editing RFID Tags

Prerequisites

- The RFID peripheral panel is displayed.
- A data group under the RFID node is selected.
- The data group has at least one tag.

Procedure

1. Select a tag in the tag information area.
2. Click on the *Edit Tag* button. The *Edit Tag* window is displayed.
3. Make any necessary changes.
4. Click on the *OK* button.

Result

The tag has been edited.

Deleting RFID Tags

Prerequisites

- The RFID peripheral panel is displayed.
- A data group under the RFID node is selected.
- The data group has at least one tag.

Procedure

1. Select one or more tags in the tag information area.
2. Click on the *Delete Tag(s)* button. The *Confirm Delete* dialog window is displayed.
3. Click on the *Yes* button.

Result

The tag or tags are deleted.

Cloning RFID Tags

Prerequisites

- The RFID peripheral panel is displayed.
- A data group under the RFID node is selected.
- The data group has at least one tag.

Procedure

1. Select one tag in the tag information area.
2. Click on the Clone Tag button. The Clone Tag dialog window is displayed.
3. Make any changes deemed appropriate to the tag (for example, change the *Tag Id*).
4. Click on the OK button.

Result

The tag is cloned. That is, a new tag is created which, except for the changes made by the user in the *Clone Tag* window, is an exact copy of the original.

Driver Selection Tool

Purpose

The Driver Selection Tool (DST) enables the SAP MI Web Admin administrator to select peripheral driver(s) that meet the mobile application peripheral requirements. This selection process also considers the mobile application target device operating system, processor, virtual machine, and available transports.

Integration

The DST is integrated into the SAP MI Web Admin.

Features

- **Display Matched Drivers**

  Displays available drivers that match the target OS, processor, VM, transport and application requirements.

- **Display Non-Matched Drivers**

  Displays available drivers that do not match the target OS, processor, VM, transport and/or application requirements. It also displays the first selection criteria that is not met.

Constraints

- The DST only recommends driver add-ons registered in the DST driver catalog. The catalog is updated by un-deploying old SDA files and deploying new SDA files.
- The DST only displays the first reason for a mismatch. It may display the operating system, processor, virtual machine, transports, or attributes in that order. On two instances more information is presented:
    - Transports - If the reason for a mismatch is the transports, the DST will display all the transports that did not match.
    - Attributes - If the reason for a mismatch is the attributes, the DST will show all attributes that were not matched. For each attribute, if more than one option does not match, the DST will display only the first option that did not match.

      Refer to Driver Requirements Document Editor [Page 93] for an example of attributes and options.

Using the Driver Selection Tool

Use

This section describes the process of launching the Driver Selection Tool and searching for drivers that match selected requirements.

Prerequisites

- You started the SAP MI Web Admin.
- Mobile application with a Driver Requirements Document is available in the SAP Web Admin

Procedure

1. Identify an application in the SAP Web Admin that requires peripheral support.
2. Choose *Select driver* on the left side of the mobile application identified.

   The Driver Selection Tool (DST) is launched.
3. Select the target operating system, processor, virtual machine, and transports.
    - **Transport -** In this field you select one or more transports. The transport is part of the mobile device. It enables you to establish the connection between the mobile device and the peripheral hardware.

      When more than one transport is selected, the DST searches for at least one of them. In other words, it follows a logical OR behavior. When at least one of the transports is found to be supported by the driver, the requirement is considered met.

Result

A list of matching drivers is displayed.