

Tutorial

Reference Manual

Java

WireFusion 4.1



Contents

INTRODUCTION.....	1
About this Manual	2
REQUIREMENTS	3
User Requirements	3
System Requirements.....	3
SHORTCUTS	4
DEVELOPMENT ENVIRONMENT	5
Menu items.....	5
Ports.....	6
SYSTEM EVENTS.....	12
USING AWT COMPONENTS	14
Exercise: Adding a Pop-up menu	14
Exercise: Adding a Text field	17
PREFERENCES.....	19
Java Libraries.....	19
Resource Files	20
Exercise: Reading text from a resource file	21
NOTES	23

Introduction

The WireFusion Java object allows programmers to easily write and compile their own Java™ code (programs) directly from WireFusion. Also non-programmers can benefit from the Java object, and the power of the Java language, by obtaining free and ready-made source code from either the Demicron web site or from third parties.

The Java object, which is found in the Misc category (Figure 1), can interact with the rest of the WireFusion environment through In-ports and Out-ports. The integrated Java development environment is easy to learn and will get you started in no time.

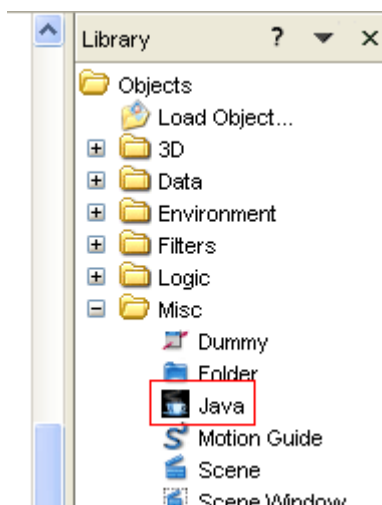


Figure 1: The Misc category, containing the Java object

A function created with a Java object can be saved and reused in other projects, or shared with other WireFusion users. After your function (code) is ready, compiled and working, simply save it using the Java object's local menu or add it to the Library as a Favorite (Figure 2). If you want to share your Java object (function) to other WireFusion users, but not reveal the source code, then you can password protect the object.

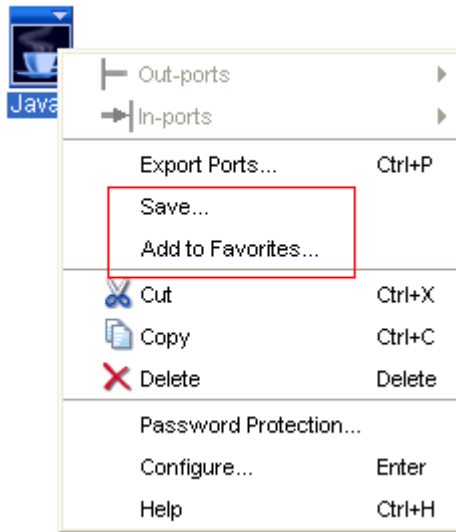


Figure 2: Java object's local menu

Some application areas of the Java object:

- Communicate with the 3DScene object using the WireFusion 3D API
- Create advanced functions (logic)
- Evaluate complex mathematical expressions
- Use Java AWT components
- Communicate with databases and web services

Please send comments and feedback regarding this manual or the software to contact@demicron.com

About this Manual

In this manual we will introduce the WireFusion Java object and explain how to write and compile Java code from inside WireFusion. We will explain the basics through some examples and exercises.

Requirements

User Requirements

To take full advantage of the Java object in WireFusion it is recommended that you know the Java programming language. Non-programmers can also take advantage of the Java object by re-using already programmed Java objects, or by copying and pasting Java source code.

If you've never used WireFusion before, then it is highly recommended that you work through the tutorial **Getting Started, Volume I**, which requires no former WireFusion knowledge.

System Requirements

In order to use Java in WireFusion you have to have either the WireFusion Professional edition, WireFusion Enterprise edition or the WireFusion Educational edition.

Shortcuts

Edit

	PC/Linux	Mac
Cut	CTRL + X	Command + X
Copy	CTRL + C	Command + C
Paste	CTRL + V	Command + V
Find...	CTRL + F	Command + F
Replace...	CTRL + R	Command + R
Undo	CTRL + Z	Command + Z
Redo	CTRL + Y	Command + Y
Preferences	CTRL + P	Command + P

Source

	PC/Linux	Mac
Verify Source	CTRL + F7	Command + F7

Tools

	PC/Linux	Mac
Insert In-port code	CTRL + I	Command + I
Insert Out-port code	CTRL + O	Command + O

Development Environment

To write Java code in WireFusion you need to insert a Java object into your project, found in the Misc category. When you drop the Java object, its dialog will automatically open and you will be presented with an editable Java source body, which will become your main class (Figure 3). When you have created your program and close the Java object dialog, by clicking OK, the source is compiled and in-ports and out-ports, defined by your source code, are automatically generated.

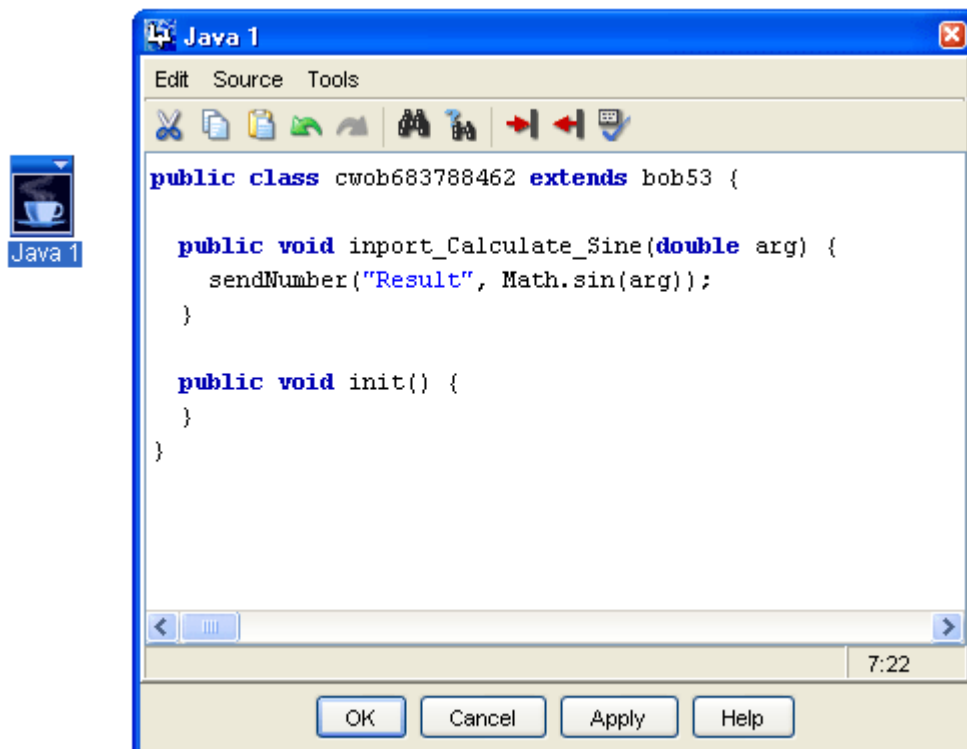


Figure 3: The Java object's dialog window

Menu items

The Java development environment is very basic and easy to use. The different menu options are described below.

Edit

Cut

Cuts text and places it on the clipboard.

Copy

Copies text and places it on the clipboard.

Paste

Pastes text from the clipboard.

Find...

Finds text in the code.

Replace...

Finds and replaces text in the code.

Undo

Undo the last text addition/deletion.

Redo

Redo the last text addition/deletion.

Preferences

Opens the preferences dialog.

Source

Verify Source

Checks for errors in the code.

Tools

Insert In-port code

Auto-generates and inserts code for creation of in-ports.

Insert Out-port code

Auto-generates and inserts code for creation of out-ports.

Ports

In-ports

To add an in-port, you add a Java method following a certain format to the main class. The argument of the method decides what argument the in-port should have, and the name of the port is decided by the name of the method. In the method body you add the code that you would like to be executed when an event is sent to the in-port. Below you can see the format to use for the six different in-port types supported.

```
public void inport_<inport-name>()
```

Defines an in-port named <inport-name> with no argument (pulse).

```
public void inport_<inport-name>(double d)
```

Defines an in-port named <inport-name> with a Number as argument.

```
public void inport_<inport-name>(boolean d)
```

Defines an in-port named <inport-name> with a Boolean as argument.

```
public void inport_<inport-name>(String d)
```

Defines an in-port named <inport-name> with a Text as argument.

```
public void inport_<inport-name>(double x, double y)
```

Adds an in-port named <inport-name> with a 2D Number as argument.

```
public void inport_<inport-name>(int color)
```

Adds an in-port named <in-port-name> with a Color as argument. The int value specifies the color and has the format AARRGGBB (AA is currently not used).

NOTE: If you want to have a space in the port name, then use underscore ('_'). The underscore will be presented as a space in the port menu.

NOTE: If you want to place a port name under a sub menu, then use the following composite port name: <sub menu name>_submenu_<port name>. An example would be, [Sound_submenu_Start](#), which would generate a sub menu named Sound with a port named Start.

Example: Create an In-port

```
// Entering the following code results into a Java object  
// with an in-port named "Print", sending a Text.  
// Events to the in-port causes the Java object to print  
// the Text value to the standard Java output (Java Console).
```

```
public class cwob<class id> extends bob53 {  
    public void inport_Print(String arg) {  
        System.out.println(arg);  
    }  
  
    public void init() {  
    }  
}
```

NOTE: Replace <class id> with the class id found in your Java object source code.

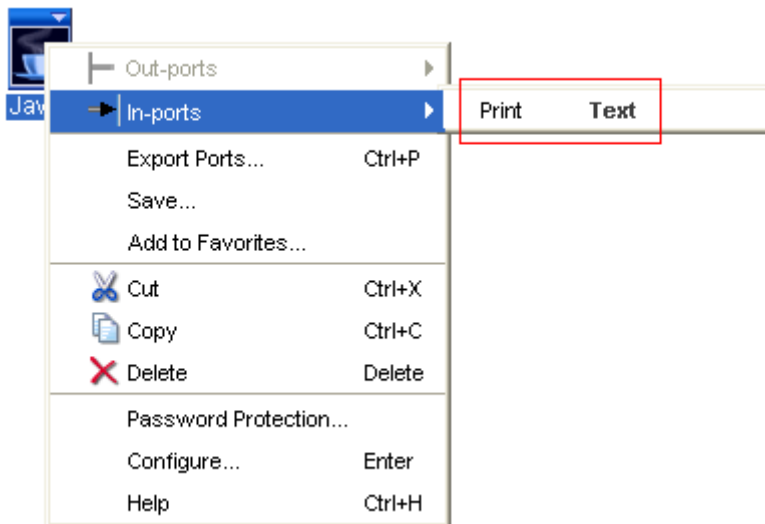


Figure 4: The new in-port Print

Out-ports

You can also send events from the Java object, through out-ports, by calling one of the following methods from your main class:

```
public void sendPulse(String portName)
```

Sends a pulse event through a port named portName.

```
public void sendNumber(String portName, double argument)
```

Sends a Number event through a port named portName.

```
public void sendBoolean(String portName, boolean argument)
```

Sends a Boolean event through a port named portName.

```
public void sendText(String portName, String argument)
```

Sends a Text event through a port named portName.

```
public void send2DNumber(String portName, double x, double y)
```

Sends a 2D Number event through a port named portName.

```
public void sendColor(String portName, int color)
```

Sends a Color event through a port named portName. The color argument should have the format AARRGGBB, where each letter represents a byte in the color `int` value. R is red, G is green and B is blue (AA currently not used).

Example:

```
sendColor("My Port", 0xFF0000); // Sends the color blue through "My Port"
```

When exiting the development environment, your Java code is automatically analyzed and checked for calls to these methods, and the appropriate out-ports are automatically created.

NOTE: The portName argument must be an explicit String (i.e. not a String reference), otherwise the code analyzer will not be able to find the port name.

Example: Create an Out-port

```
// A Java object with this code can receive a Number value,
// calculate the sin value of the incoming Number
// value, and then send the result through an out-port.
public class cwob<class id> extends bob53 {
    public void inport_Calculate_Sine(double arg) {
        sendNumber("Result", java.lang.Math.sin(arg));
    }

    public void init() {
    }
}
```

NOTE: Replace <class id> with the class id found in your Java object source code.

NOTE: You can define sub menus and spaces for out-ports in the same way you do for in-ports.

Auto Generate Ports

An alternative to manually entering code for in-ports and out-ports is to use the menu options Insert In-port code and Insert Out-port code, which auto-generates code.

Insert In-port code

Insert In-port code allows you to quickly insert the code needed to create an in-port. When chosen, either from the Tools menu or from the toolbar (Figure 5), the In-port code generator dialog is opened (Figure 6).

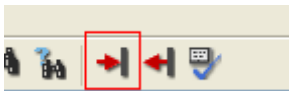


Figure 5: The Insert In-port code button

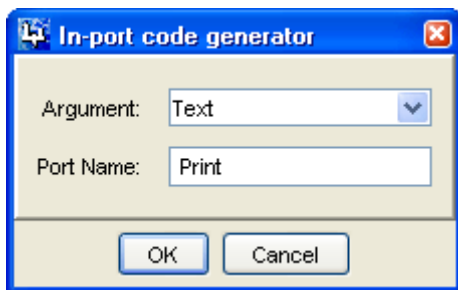


Figure 6: The In-port code generator dialog

Argument

Choose which type of argument the in-port shall have: Any Argument (pulse), Number, Boolean, Text, 2D Number or Color.

Port Name

Choose a name for the in-port.

Insert Out-port code

Insert Out-port code allows you to quickly insert the code defining an out-port. When chosen, either from the Tools menu or from the tool bar (Figure 7), the In-port code generator dialog is opened (Figure 8).



Figure 7: The Insert Out-port code button

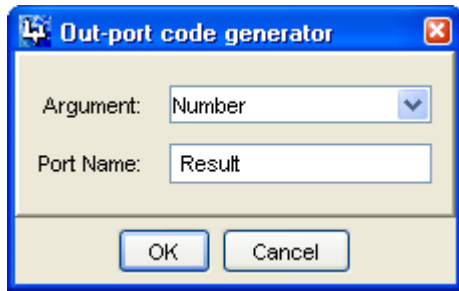


Figure 8: The Out-port code generator dialog

Argument

Choose which type of argument the out-port shall have: No Argument (pulse), Number, Boolean, Text, 2D Number or Color.

Port Name

Choose a name for the out-port.

System Events

You can listen to a number of System Events. To do this, call the `enableSysEv(int eventType)` method, where `eventType` is one of the following constants:

FRAME:

Sent when a new frame has been rendered.

START:

Sent when the presentation is started.

STOP:

Sent when the presentation is stopped.

KEY:

Sent when a key is pressed. (See the Java API for information about the `java.awt.Event` for information about key event fields).

System events that have been enabled will be sent to the following method, which you should override to listen to the events:

```
public void handleSysEv(Event ev)
```

Example: Creating a Hello World message

```
// Sends out an event with the Text value
// "Hello world!" at presentation startup

import java.awt.*;

public class cwob<class id> extends bob53 {

    public void init() {
        enableSysEv(START);
    }

    public void handleSysEv(Event ev) {
        sendText("My output", "Hello world!");
    }
}
```

NOTE: Replace `<class id>` with the class id found in your Java object source code.

If you listen to multiple types of system events, you can tell them apart by checking the [Event.id](#) field, which will be the event type value.

Using AWT Components

You can add AWT (Abstract Windowing Toolkit) components, like text fields and pop-up menus, to WireFusion presentations using the Java object. This can be useful if there is no corresponding WireFusion component available. AWT components do not work as normal visual WireFusion components. For example, they will not be visible in the Layers view, and, they are not included in the WireFusion layer hierarchy. They are always placed above, or on the top of, a WireFusion presentation. This means that you cannot have, for example, a lens effect on an AWT button; the AWT button will always be placed above the lens effect. Lightweight components, like Swing components, are not supported.

Exercise: Adding a Pop-up menu

In this example we will add a WireFusion button that will show an AWT pop-up menu when you click the button. We will add two items to the pop-up menu. The first item is a standard menu item that, when you select it, will show a label named "Item has been selected". The other menu item is a checkbox that will show/hide a label named "Checkbox checked".

Step 1

Insert a Button and two Label objects into an empty project.

In the Button dialog:

- Set the label to "Show pop-up menu"

In the Label dialogs:

- For 'Label 1', set the label to "Item has been selected".
- For 'Label 2', set the label to "Checkbox checked".

Step 2

Deactivate both Label objects at the presentation startup by unchecking their Activated checkboxes, found in the Layers view (Figure 9).

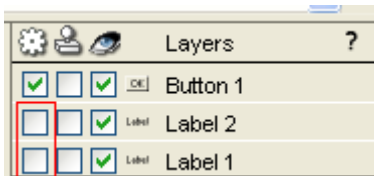


Figure 9: Deactivating the Label objects in the Layers view

Step 3

Open the Java object and enter the following code.

```

import java.awt.*;
import java.awt.event.*;

public class cwob<class id> extends bob53 implements ActionListener, ItemListener {
    PopupMenu popupMenu = new PopupMenu();
    MenuItem menuItem = new MenuItem("Standard item");
    CheckboxMenuItem checkBoxItem = new CheckboxMenuItem("Checkbox item");

    public void inport_popup() {
        // m.mp.x and m.mp.y contains the mouse cursor coordinates
        popupMenu.show(a.getCmp(), m.mp.x, m.mp.y);
    }

    public void actionPerformed(ActionEvent ev) {
        if (ev.getSource()==menuItem) {
            sendPulse("Standard item selected");
        }
    }

    public void itemStateChanged(ItemEvent ev) {
        if (ev.getSource()==checkBoxItem) {
            if (checkBoxItem.getState())
                sendPulse("Checkbox checked");
            else
                sendPulse("Checkbox unchecked");
        }
    }

    public void init() {
        popupMenu.add(menuItem);
        menuItem.addActionListener(this);
        popupMenu.add(checkBoxItem);
        checkBoxItem.addItemListener(this);
        a.getCmp().add(popupMenu);
    }
}

```

NOTE: Replace <class id> with the class id found in your Java object source code.

Step 4

Close the Java object and make the following connections:

Connect:

- 'Button 1' > Out-ports > Button Clicked
to
'Java 1' > In-ports > popup

- 'Java 1' > Out-ports > Standard item selected
to
'Label 1', In-ports > Activate
- 'Java 1' > Out-ports > Checkbox checked
to
'Label 2', In-ports > Activate
- 'Java 1' > Out-ports > Checkbox unchecked
to
'Label 2' > In-ports > Deactivate

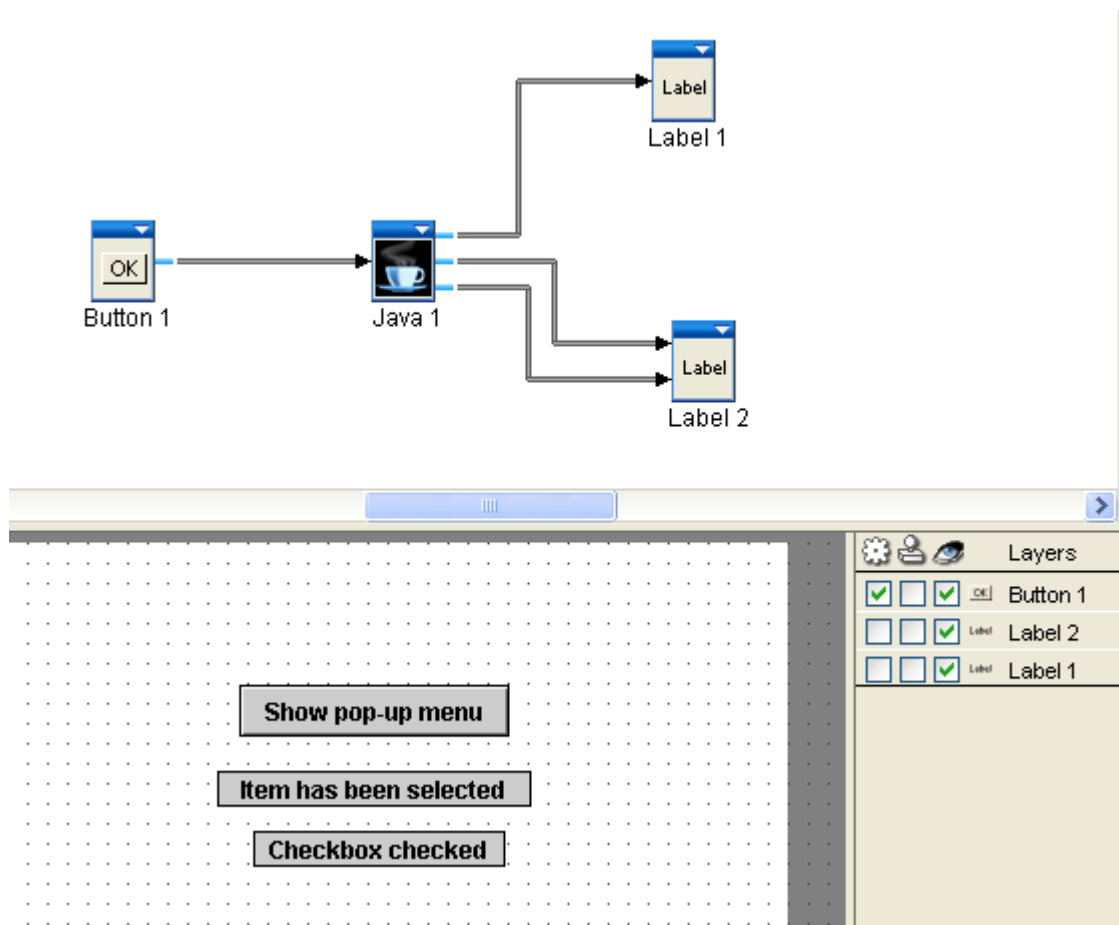


Figure 10: Objects and connections done for the pop-up menu example

Step 5

Now you can view the presentation and see the result of using the pop-up menu. Press F7 to test your presentation (Figure 11).

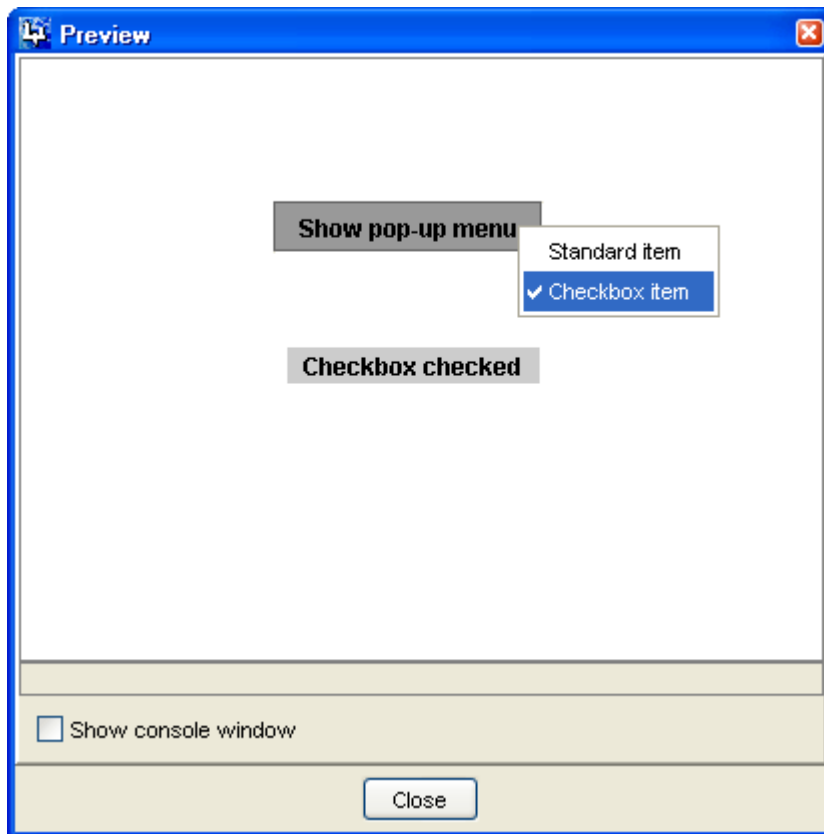


Figure 11: Preview of AWT popup menu example

Exercise: Adding a Text field

In this example a text field will be added to the presentation, and when the Enter key is pressed in the text field, the text will be shown in the status bar.

Step 1

Insert a Java object into a new and empty project and enter the following code.

```
import java.awt.*;
import java.awt.event.*;

public class cwob<class id> extends bob53 implements ActionListener {
    TextField textField = new TextField("Enter a text and press enter");

    public void init() {
        a.getCmp().setLayout(null); // Use null layout
        a.getCmp().add(textField);
        // Use setBounds since no layout manager is use
        textField.setBounds(20,50,200,
            textField.getPreferredSize().height);
        textField.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ev) {
```

```
        sendText("Text entered", textField.getText());  
    }  
}
```

NOTE: Replace <class id> with the class id found in your Java object source code.

Step 2

Insert a System object into the project and make the following connection.

Connect:

'Java 1' > Out-port > Text entered

to

'System 1' > In-ports > Set Status Bar Text

Step 3

Press F7 to test your presentation (Figure 12).

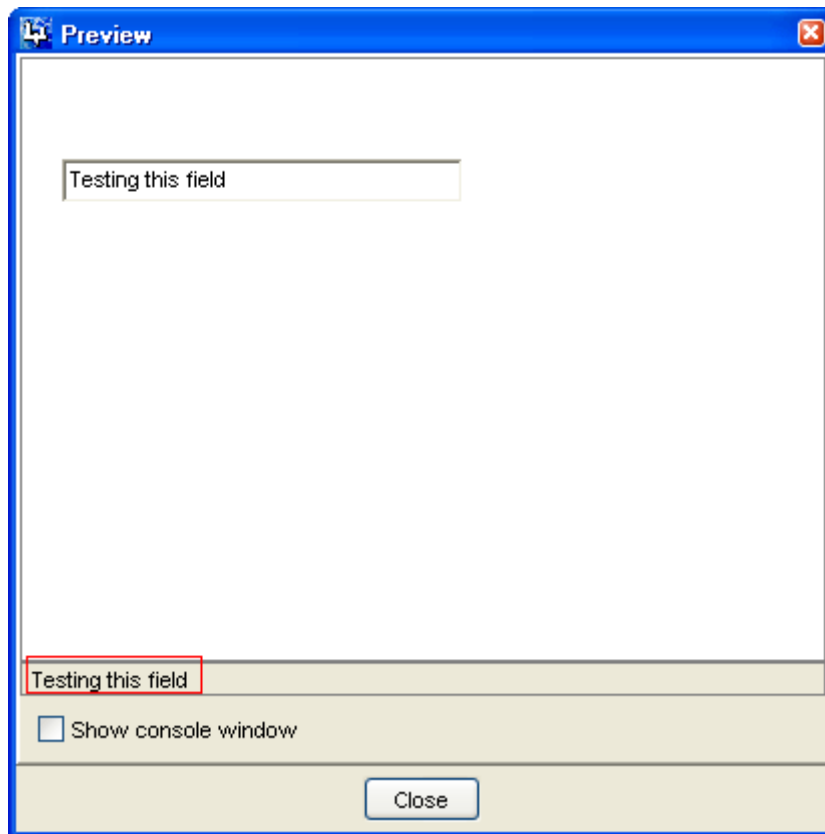


Figure 12: Example 5: printing a text in the status bar

Preferences

In the Java Preferences dialog (Figure 14), which is reached from the Java object > Edit > Preferences, you can specify external resources and Java libraries. These resources and libraries can then be used by the Java object.

Java Libraries

To specify a Java library, select the Java Libraries panel in the Preferences dialog, and then add your Java Archive file (*.jar*). You can now use the classes in your archive within the Java object.

Examples of useful libraries:

- Communicate with a MySQL server using the MySQL Connector library, available from <http://www.mysql.com/>
- Nano XML, a very compact XML parser, available from <http://nanoxml.cyberelf.be/>

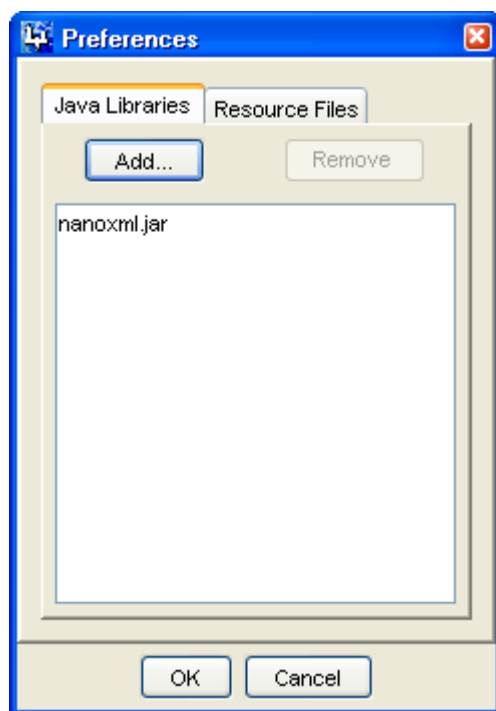


Figure 13: NanoXML library added

Creating your own Java libraries

If you want to use your favorite Java development tool for writing code, instead of using the WireFusion Java object, then you can develop your own Java libraries and import them into the WireFusion Java object. You will still have to write some code using the WireFusion Java editor though, to create in- and out-ports and to communicate with your Java library.

For example, if you want to create the Java library from some class files in a package called "mypackage", which is, for example, located in the folder `c:\mypackage\`, then execute the following command from a command shell:

```
jar cvf MyClasses.jar mypackage\*.class
```

`MyClasses.jar` will then be created and can be imported as a Java library.

NOTE: You can read more about JAR at <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>

Resource Files

To specify a resource file, select the Resource Files panel in the Preferences dialog, and then add the resource file.

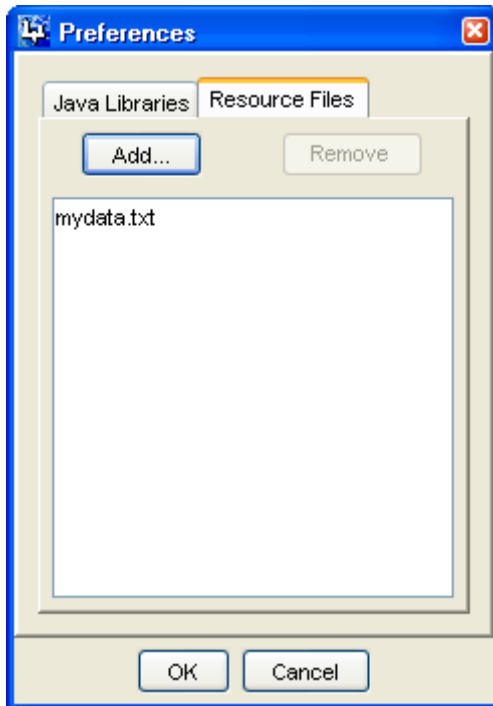


Figure 14: The Resource Files panel in the Preferences dialog

You can now load and access the resource file from the Java object. Create an input stream to the resource file using the `get()` and `is()` methods:

```
InputStream is = get(<myfilename>).is();
```

Exercise: Reading text from a resource file

In this example we will add a text file as resource. We will then load the text file and print the text in the console window.

Step 1

Create a text file and add the text "Hello". Save the text file as mydata.txt

Step 2

In the Java object, add the text file as a resource file under Edit > Preferences > Resource Files

Step 3

Enter the following code into the Java object:

```
import java.io.*;

public class cwob<class id> extends bob53 {
    public void init() {
        DataInputStream is = new DataInputStream(get("mydata.txt").is());
        try {
            String line = is.readLine();
            while (line != null) {
                System.out.println(line); // Print to Console
                line = is.readLine();
            }
        } catch (IOException e) {e.printStackTrace();}
    }
}
```

NOTE: Replace <class id> with the class id found in your Java object source code.

Step 4

Preview the project by pressing F7 and select the "Show console window" checkbox. "Hello" will be printed in the Console window.

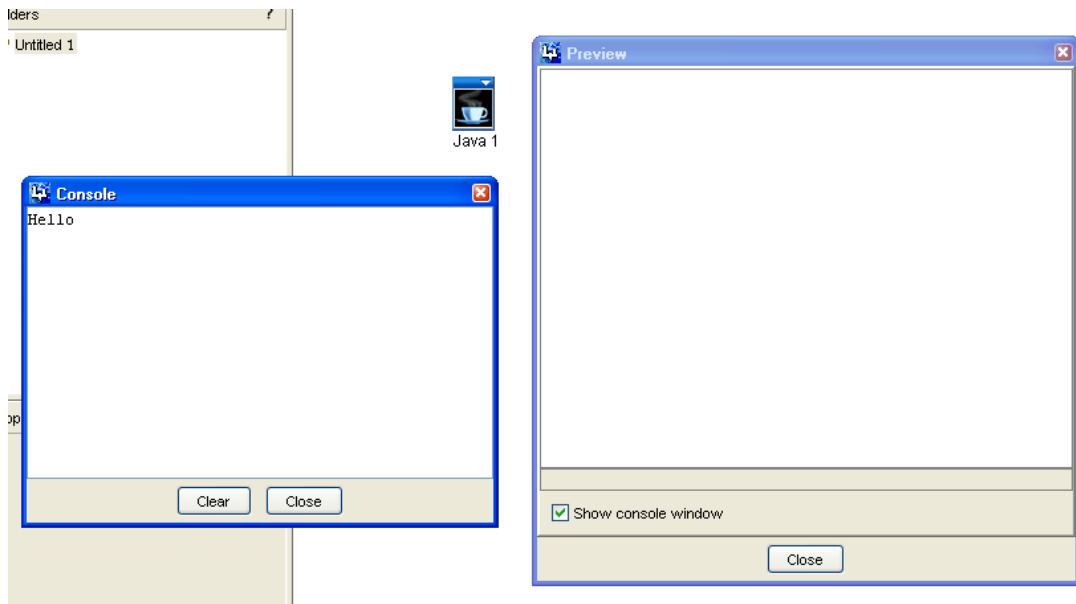


Figure 15: Output in the Console from the example project

If you, after you have added a resource file to a Java object, specify the resource file as dynamic (in the Loading Manager), it will be placed outside of the preload archive (preload.jar). This will allow you to easily edit the file on the web server (assuming you publish the presentation as an applet). If you also add an XML parser as a Java Library, for example NanoXML (see above), you can let the resource file contain XML code and you will be able to parse the XML file from within the Java object.

Notes

- If you need to initialize the class, enter the initializations code in the `init()` method, which is automatically called at the presentation startup.
- The name of the main class is chosen automatically (cwob<class id>) and should not be modified.
- There is no Target Area associated with the Java object, so you cannot directly manipulate pixels from the Java object.
- To create additional classes to the main class, use private classes defined in the main class file or inner classes.

* * *

Publication date: December 14, 2005

The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

All rights reserved © 2005 Demicron AB, Sundbyberg, Sweden. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Demicron and its licensors, if any.

Trademarks — Demicron and WireFusion are trademarks of Demicron AB. Acrobat, Photoshop are registered trademarks of Adobe Systems Incorporated. Java, SunSoft are trademarks of Sun Microsystems, Inc. Windows95, Windows98, Windows ME, Windows NT, Windows 2000, Windows XP are trademarks or registered trademarks of Microsoft Corporation. Pentium is a trademark of Intel Corporation. OS X is a trademark of Apple Computer. All other trademarks are the property of their respective owners.