



Samsung Mobile Widget Development Guide

1.0 - About This Document

The Samsung Mobile Widget SDK helps you develop widgets for Samsung mobile devices running the TouchWiz™ user interface. This document, the *Samsung Mobile Widget Development Guide*, introduces you to the capabilities of the SDK and explains how to use it to quickly create widgets that run on a wide variety of TouchWiz devices.

The documentation is divided into the following sections:

- **About This Document** - The section you are looking at now, which includes a brief introduction, release notes, and copyright notifications.
- **Introducing the Samsung Mobile Widget SDK** - Describes how to install, update, and launch the SDK.
- **The Widget Perspective** - Describes the user interface of the Samsung Mobile Widget SDK and describes how to customize it.
- **Creating a Widget Project** - A simple guide to creating a project that you can use to quickly produce widgets for Samsung TouchWiz devices.
- **Supporting Multiple Devices with Overrides** - Using one widget project to support many different devices by putting customized files in *override* directories.
- **Tutorial Sample** - A simple widget that uses JavaScript, CSS, and images.
- **Debugging Widget Projects** - How to use publicly available debugging tools to find errors in your projects.
- **Widget-Development Tips** - Techniques and suggestions that will help you get the most out of the SDK.
- **Developing for BONDI** - How to create widgets that use the BONDI framework for gaining access to device-based functionality.
- **SDK Menu Items** - A reference to the menu items that are specific to the Samsung Mobile Widget SDK.



You must have either the **Eclipse IDE for Java Developers** or the **Eclipse IDE for Java EE Developers** to run the Samsung Mobile Widget SDK. If you install the **Eclipse IDE for Java Developers**, several additional components (such as the WTP—*Web Tools Platform*) are also installed when you install the Samsung Mobile Widget SDK. For more information, see [Eclipse Requirements](#).

1.1 - Document Version History

| Version | Date | Notes |
|---------|------------|---|
| 0.1 | 06-25-2009 | Draft version |
| 1.0 | 08-04-2009 | First Release |
| | 08-10-2009 | Reformatting |
| | 08-14-2009 | New update site URL |
| | 08-20-2009 | Modify tutorial, add platform release note |
| 1.0.1 | 09-25-2009 | Automatic update site addition, new device. |
| 1.1 | 12-31-2009 | Numerous product updates, including auto-export, transparency support, seller site integration, and UI improvements. |
| 1.2 | 04-28-2010 | Implementation of BONDI emulation and the addition of the BONDI-enabled Wave device. This release is final on 04-28-2010. |

1.2 - Release Notes

This section lists issues that are specific to recent releases of the Samsung Mobile Widget SDK.

1.2.1 - Version 1.2 Update

The following features have been added for version 1.2.

- **BONDI emulation:** Widgets can be created that utilize the new capabilities of BONDI-enabled devices. Discussed in [Developing for BONDI](#).
- **New device:** The Wave, a newly released BONDI-enabled device, has been added to the Device List.
- **Updated Java requirement to version 6:** Discussed in [section 2.0 - Installing Java and Eclipse](#).

1.2.2 - Version 1.1 Update

The following features have been added for version 1.1.

- **ACCESS NetFront browser is now emulated:** Discussed in [section 4.5 - Previewing a Widget Project](#).
- **Right-click (Context) menu:** There are four new items in the context menu:
 - **Add Device** adds a device to the Device List. Mentioned in the note in [section 4.2 - Selecting Devices](#).
 - **Package** creates a .WGT file. Mentioned in [section 4.6 - Packaging a Widget](#).
 - **Update Project** ensures that the current XML schema for your project is up to date. It may be necessary to use this menu item when you import a project that was created with an earlier version of the Samsung Mobile Widget SDK. Most users will not use this menu item.
 - **Application Store** causes your default browser to open a window to the Samsung Application Store, where you can upload your new widget. Mentioned in [section 4.6 - Packaging a Widget](#).
- **Widget Menu item:** A new menu item has been added, with the same selections as are listed above for the right-click (context) menu. This menu item may not be visible to users who are upgrading from previous versions of the SDK until they reset the perspective (by choosing **Window > Reset Perspective**).
- **Improvements to widget-building routines:** Refinements include auto-export to folder in project and improved naming of packaged filenames. These improvements are discussed in [section 4.6 - Packaging a Widget](#). In previous releases this section was named *Exporting a Widget*; the "export" functionality has been replaced with "packaging."

- **Relaxed Java dependencies:** Java and Eclipse requirements are discussed in a new section, 2.0 - Installing Java and Eclipse.
- **Proxy support:** A new, simpler procedure for configuring proxy support is documented in section 8.4 - Using a Proxy Server. Users of earlier versions of the SDK should remove any system-wide environment variables or browser-specific proxy settings that were specifically set for previous versions of the Samsung Mobile Widget SDK:
 - To reset the system-wide environment variable, open **Control Panel > System**, click the **Advanced** tab, and click the **Environment Variables** button.
 - To reset proxy settings in Opera, go to **Tools > Preferences**, click the **Advanced** tab, open the **Network** node, and click the **Proxy Servers** button.
 - To reset proxy settings in Safari, go to **Edit > Preferences**, select **Advanced**, click the **Change Settings** button under **Proxies**, and click the **LAN Settings** button.
 - To reset proxy settings in Firefox, go to **Tools > Options**, click the **Advanced** tab, click the **Network** tab, and click the **Settings** button in the "Connection: Configure how Firefox connects to the Internet" section.
- **New devices:** Newly released TouchWiz devices have been added to the Device List.
- **Transparent backgrounds:** Transparency is supported on all of the browsers used by the Samsung Mobile Widget SDK. The NetFront emulator supports only 100% transparency. This is mentioned in section 4.5 - Previewing a Widget Project.
- **Improved error handling:** A dialog box warning about errors is displayed when previewing or packaging a project with errors. Discussed in section 4.5 - Previewing a Widget Project.
- **Double-click on device launches preview:** This is mentioned several times in the document. The first mention is in section 4.5 - Previewing a Widget Project.
- **New fields in XML files:** It is possible to add `<author>`, `<license>`, and `<feature>` tags to `project.xml`. This is mentioned in `Editing Project.xml`.
- **Access Network emulation:** When you uncheck the **Access Network** check box in the **Create a Widget Project** dialog box, your new widget does not have network capabilities when it is run on a device. Starting with this version of the Samsung Mobile Widget SDK, the device emulators used during previews disable network capabilities when **Access Network** is not checked. In previous versions, the emulators ignored the state of this check box.

Other refinements, which may not have been mentioned in this list, are noted elsewhere in this document.

1.2.3 - Version 1.0.1 Update

The following features have been added to version 1.0.1.

- **Automatic Update Site:** A new update site has been created for the Samsung Mobile Widget SDK. It is: `http://widget.samsungmobile.com/sdk/`. Current users just need to perform an update using their current update sites. Choose **Help > Check for Updates**. As part of installing the 1.0.1 update, the list of update sites in **Preferences > Install/Update > Available Software Sites** will automatically be updated to point to the new Samsung Mobile Widget SDK update site at `widget.samsungmobile.com`. Future updates will then come from the new site. Use of the update site at `innovator.samsungmobile.com` site is now deprecated.
- **Device List:** The GT-M8910 Pixon12 device has been added to the device list.

1.2.4 - Notes for Beta Users

If you were a beta user of the Samsung Mobile Widget SDK, you will need to uninstall your beta version before installing the current version. To do this, close the widget perspective and then follow the instructions at *Uninstalling the Samsung Mobile Widget SDK*.

If you have used earlier versions of the Samsung Mobile Widget SDK, you will need to make some modifications to preexisting widget projects. Your best course of action is to create a new project and then copy files and settings from the preexisting project into the new one.

- The generic XML file, `configinfo.xml`, is now named `project.xml`. You will need to copy settings from your old XML file into the `project.xml` for your new project.
- The `overrides` directories were named by the screen resolution of the device, but now the names are a combination of the resolution and browser name. You will need to copy files from any older `overrides` directory into the appropriate `overrides` directory in your new project.

1.2.5 - Workspace Setup

Please verify that your workspace is set up to use UTF-8 as the default text encoding.

1. Select **Window > Preferences > General > Workspace**
2. Click the **Other** radio button in the `Text file encoding` section and select UTF-8 from the drop-down list.

1.2.6 - JavaScript Support

- Preferences are in **Window > Preferences > Web > JavaScript**
- Do not manually add or remove libraries from your project. Libraries will be better supported in a future release.
- Until further notice, do not select "Enable JavaScript semantic validation" (see **Window > Preferences > Web > JavaScript > Validator > Errors/Warnings**). Selecting this option currently produces some spurious errors and warnings.
- If you prefer not to be warned about missing semicolons, do the following:
 1. Select **Window > Preferences**
 2. Expand **Web > JavaScript > Validator**
 3. Select **Errors/Warnings**
 4. Check "Enable JavaScript semantic validation"
 5. Expand "Potential programming problems"
 6. Change "Optional Semi-Colon" to "Ignore"
 7. Turn off "Enable JavaScript semantic validation"
 8. Click the **OK** button. Select **Yes** when asked about a full rebuild.

1.2.7 - Run As Widget

If you see a Windows error dialog box when you use the **Run As > Widget** command, you might be missing some VisualStudio runtime libraries. Download and install them [here](#).

Currently **Run As Widget** is unsupported on Mac OS X or Linux. To run your widget on Mac OS X or Linux, follow the instructions for running with a browser in the Widget Development Tips section. See: **Previewing on a Browser**

Watch for future updates to the Samsung Mobile Widget SDK for your favorite platform.

1.3 - Help System

If you are reading this document in the form of a PDF file, you might prefer to use the integrated help system in the Samsung Mobile Widget SDK. The help system includes a persistent menu tree, a robust search facility, bookmarking capabilities, and easy access to a wealth of Eclipse documentation.

- In the **Widget** perspective, choose **Help > Help Contents**.
- Use the **Contents** view to navigate to **Samsung Mobile Widget Development Guide**.

1.4 - Copyright Notifications

Any company and product names not explicitly mentioned below may be trademarks of the respective companies with which they are associated.

Samsung Electronics

This material is copyrighted by Samsung Electronics. Any unauthorized reproductions, use or disclosure of this material, or any part thereof, is strictly prohibited and is a violation under the Copyright Law. Samsung Electronics reserves the right to make changes in specifications at any time and without notice. The information furnished by Samsung Electronics in this material is believed to be accurate and reliable, but is not warranted true in all cases.

"SAMSUNG", "SAMSUNG.com" and "SAMSUNG DIGITaIl Everyone's invited" are trademarks of SAMSUNG in the United States or other countries. Unauthorized use or duplication of these marks is strictly prohibited by law.

The Samsung Logo is the trademark of Samsung Electronics.

Eclipse Foundation, Inc.

Eclipse is a trademark of Eclipse Foundation, Inc. Copyright © 2005, 2009. All rights reserved.

Firefox

Firefox Copyright © 1998-2009 Contributors. All rights reserved. Firefox and the Firefox logos are trademarks of the Mozilla Foundation. All rights reserved.

Microsoft Corporation

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

NetFront

Copyright © 2009, ACCESS CO., LTD.

ACCESS and NetFront are trademarks or registered trademarks of ACCESS Co., Ltd. in Japan and other countries.

Opera

Copyright © 1995-2009 Opera Software ASA. All rights reserved.

Safari

Copyright © 2007-2009 Apple Inc. All rights reserved.

Sun Microsystems, Inc.

Java is a trademark of Sun Microsystems, Inc.

WebKit, Apple, and JavaScriptCore

The WebKit Open Source Project (including portions from the khtml, kcanvas, kdom, and ksvg2 projects) and JavaScriptCore Project (including portions from the kjs project) Copyright © 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009



All contents Copyright © Samsung Electronics Co., Ltd.



Introducing the Samsung Mobile Widget SDK

The Samsung Mobile Widget SDK is a widget-development environment based on the Eclipse™ open development platform. Samsung Mobile Widget SDK helps you develop widgets for Samsung mobile devices running the TouchWiz™ user interface.

This guide shows you how to create a widget project, launch the widget on the device emulator, and package it for release. In addition, you will find a step-by-step description of creating a sample widget that answers many common questions about widget development.

Samsung's TouchWiz user interface includes 3D-effects, haptic feedback, and gesture and voice controls. The TouchWiz widget dock makes it easy for customers to manage and download widgets that customize and personalize their devices.

Samsung Mobile Widget SDK simplifies the task of developing widgets in a number of ways:

- It uses templates to create projects you can use as a starting point.
- It provides an emulator that makes it easy to visualize the final product on multiple devices, browsers, and platforms.
- It simplifies the customization of widgets for different sizes of device screens.
- It provides IDE features like autocomplete, context menus, and error tracking.
- It imports existing widgets, enabling you to examine and modify them.
- It packages widgets for distribution.

The Samsung Mobile Widget SDK is currently supported on Windows XP. It bundles browser libraries to closely emulate the different mobile browsers used on Samsung devices.

This guide assumes that you have some familiarity with Eclipse and that you are familiar with HTML, CSS, and JavaScript.

2.0 - Installing Java and Eclipse

You will need at least version 6 of the JRE and the Galileo release (version 3.5) of Eclipse to install the Samsung Mobile Widget SDK.

2.0.1 - Java Requirements

You will need the JRE (Java Runtime Environment) version 6 or higher to use the Eclipse IDE. To download Java, visit: <http://java.com/>. (The reference platform is Sun Java 2 Standard Edition 5.0, Update 11.) For more information, see: <http://www.eclipse.org/downloads/moreinfo/jre.php>

2.0.2 - Eclipse Requirements

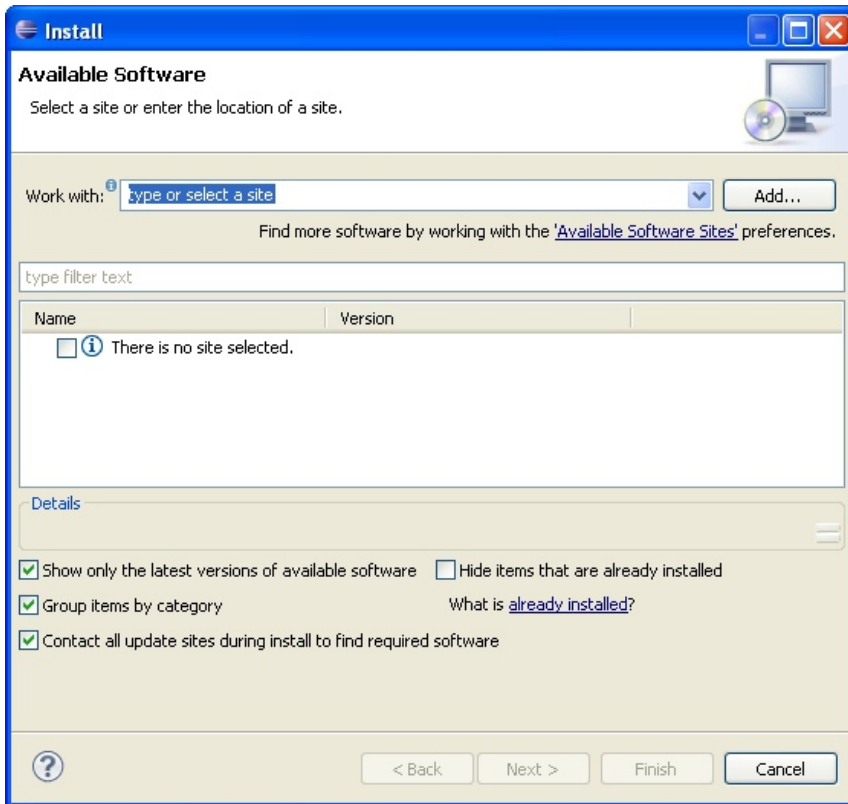
The Galileo release of Eclipse (version 3.5 or higher) is required to run the Samsung Mobile Widget SDK. You can use either Eclipse IDE for Java Developers or Eclipse IDE for Java EE Developers. Eclipse IDE for Java Developers is significantly smaller than Eclipse IDE for Java EE Developers. If you start with Eclipse IDE for Java Developers, the installation routine for the Samsung Mobile Widget SDK automatically downloads additional features that you need, assuming that the <http://download.eclipse.org/releases/galileo> update site is present and enabled in Eclipse. If you are using the

Eclipse IDE for Java Developers and you have trouble installing the Samsung Mobile Widget SDK, ensure that this update site is present and then restart Eclipse.

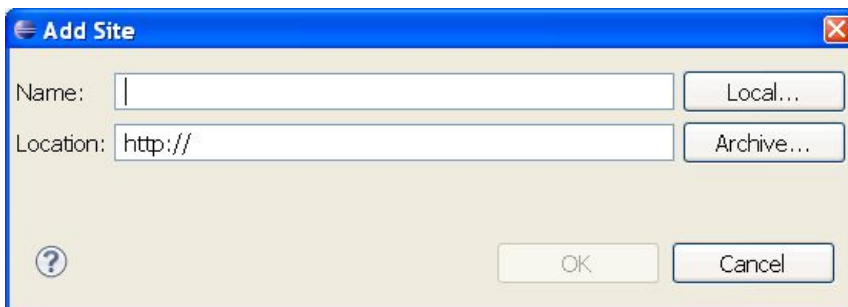
To download Eclipse, visit: <http://www.eclipse.org/downloads/>

2.1 - Installing the Samsung Mobile Widget SDK

The Samsung Mobile Widget SDK widget development environment is implemented as a series of Eclipse plug-ins. To install it, go to the Eclipse **Help** menu and select **Install New Software**. The **Install** dialog box appears:



Click the **Add...** button to add the Samsung Mobile Widget SDK distribution site to your list of available software sites. The **Add Site** dialog box appears:



Give the update site an identifiable name in the **Name** field - something like "Samsung Mobile Widget SDK update site" - and specify the URL in the **Location** field. The URL for the Samsung Mobile Widget SDK update site is <http://widget.samsungmobile.com/sdk/>. Click the **OK** button when you are finished. From the **Work with:** combo box, select the update site that you just added.



The Install dialog box should now display "Samsung Mobile Widget SDK" in the list of available software:



Select the checkbox next to "Samsung Mobile Widget SDK" and then click the **Next** button. A progress bar across the bottom of the window shows you that Eclipse is "Calculating requirements and dependencies."

An Install Details screen appears, listing the items to be installed:



Select the "Samsung Mobile Widget SDK" item and click the **Next** button.

Review and accept the license, and click the **Finish** button.

When the "Operation in progress..." progress bar tells you that the installation is finished, a **Software Updates** dialog box "strongly recommends" that you restart Eclipse. Click **Yes** to follow this suggestion.

In the future, various language packages will also be available at the download site. You will download the language packages as part of the installation procedure if you want to run the Samsung Mobile Widget SDK in one of the supported languages.

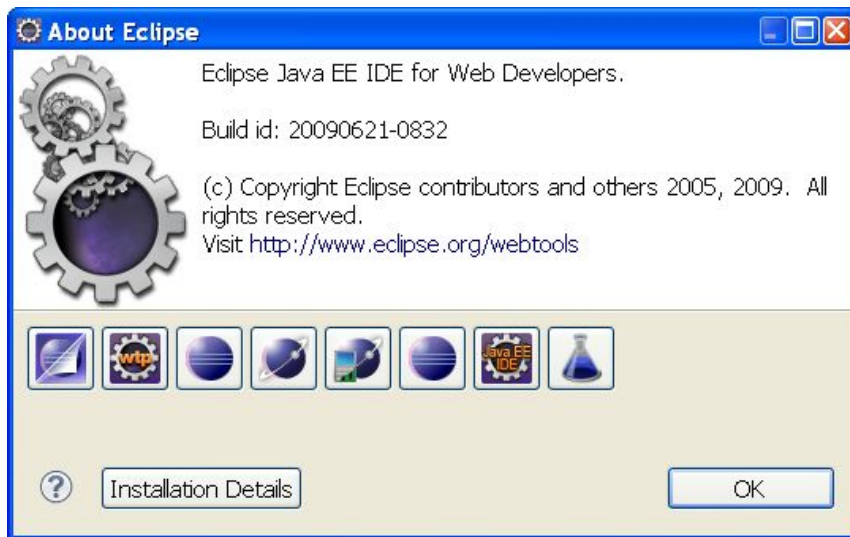
2.2 - Updating the Samsung Mobile Widget SDK

After you have entered the Samsung Mobile Widget SDK installation site as part of the installation procedure, it is easy to stay up-to-date. Just go to **Help > Check for Updates**. Any Samsung Mobile Widget SDK updates will be listed in the **Available Updates** dialog box.

If you need to view or change the Samsung Mobile Widget SDK update site you have specified, go to **Window > Preferences**, expand the **Install/Update** node, and choose **Available Software Sites**.

2.3 - Uninstalling the Samsung Mobile Widget SDK

If you ever needed to uninstall the Samsung Mobile Widget SDK, you would begin by going to **Help > About Eclipse SDK** and clicking the **Installation Details** button.



You should see "Samsung Mobile Widget SDK Widget Development" in the list on the Installed Software tab. To uninstall the Samsung Mobile Widget SDK, you would select "Samsung widget development core" in the list and click the **Uninstall...** button.

2.4 - Launching the Samsung Mobile Widget SDK

After you have installed the Samsung Mobile Widget SDK, you can launch it by opening the new **Widget** perspective in Eclipse. To do this, click the **Open Perspective** icon at the right side of the Eclipse window and choose **Other...** from the bottom of the list.



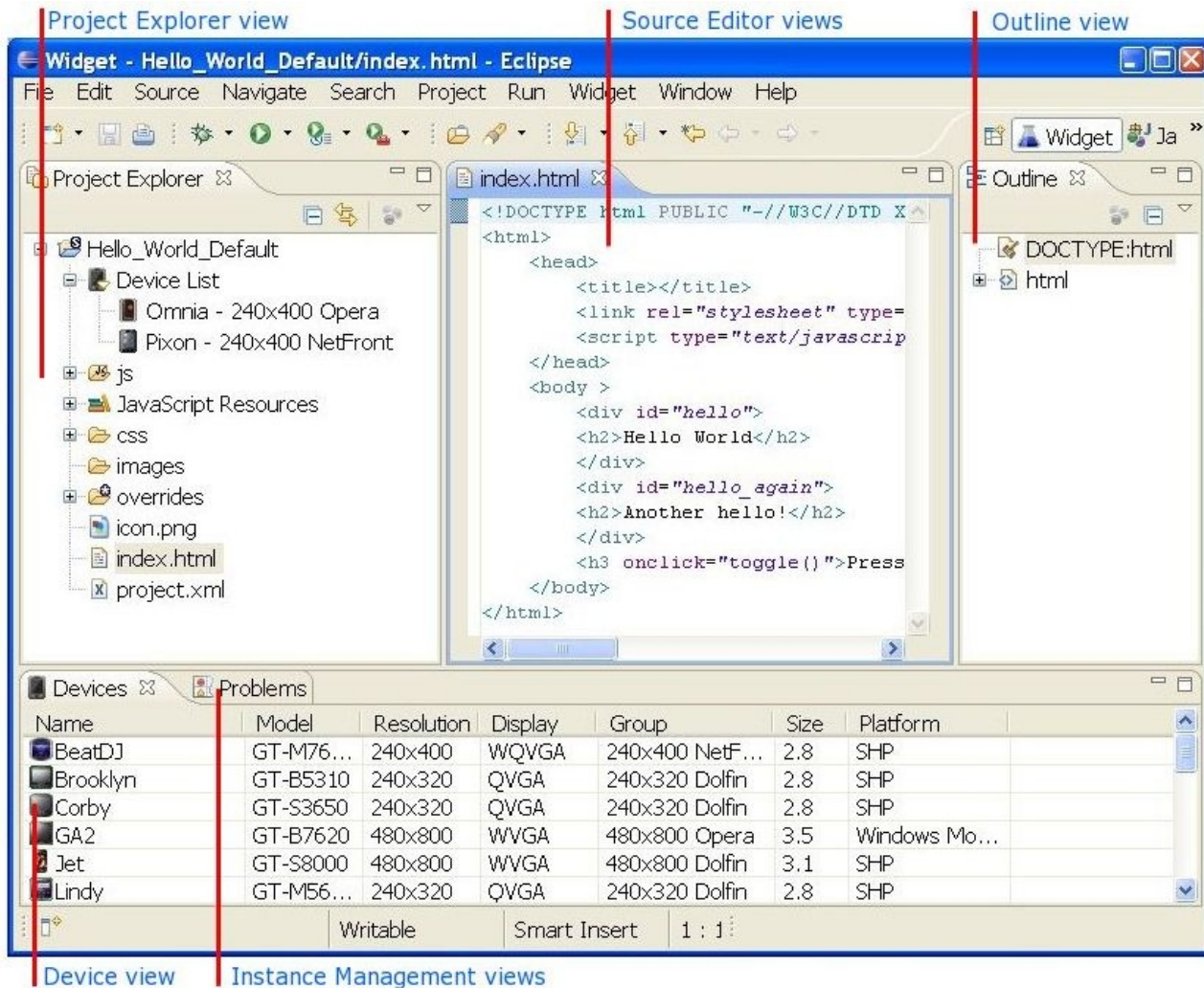
Choose "Widget" from the list in the **Open Perspective** dialog box and click the **OK** button.

The **Widget** perspective opens, revealing the views and editors you will use to help build widgets for the TouchWiz interface.



The Widget Perspective

The Widget perspective in Eclipse looks like this:



Most of this is familiar to experienced Eclipse developers:

- **Project Explorer view** shows you the names and structures of your current projects. Note that widget projects are decorated with an open folder containing an "S".
- **Source Editor views** display the editable files in your project and provide tools for editing them.
- **Outline view** gives you an easy way to see the organization of the file that is currently open in the editor and allows you to jump quickly to given point in the file.
- **Device view** displays the devices that are emulated when you preview your widget. Enables widget developers to add new devices to their projects by dragging them into the project's `Device List`.
- **Instance Management views** give you information you can use to add devices to your project and diagnose problems. Other views appear here when they are invoked; for example a **Search** view displays the results of any search operation, and a

Console view displays debugging output.

3.0 - Customizing a Perspective

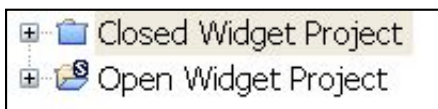
A perspective contains multiple views into the files that make up your project. The **Source Editor** views must remain inside the main Eclipse window, but the other views can be dragged outside and positioned wherever you like. This can give you more room to work in the Source editors. Move or restore a view by clicking its tab and dragging it to a new position.

To return a perspective to its default appearance, choose **Window > Reset perspective**.

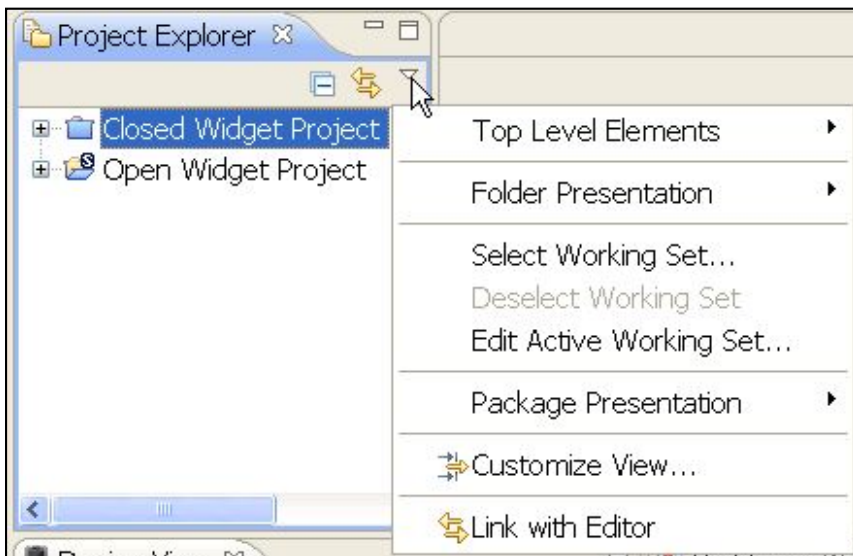
If you find that you usually modify the default perspective, you can save your preferred configuration by choosing **Window > Save Perspective as...** and giving your configuration a unique name. This new perspective now appears among the choices in the **Open Perspective** dialog box.

You can add views to your perspective, if you like. To see a list of views you can add, choose **Window > Show View**. If you choose **Other...** in the menu that appears, you will be presented with a **Show View** dialog box that gives you access to a large number of possibilities.

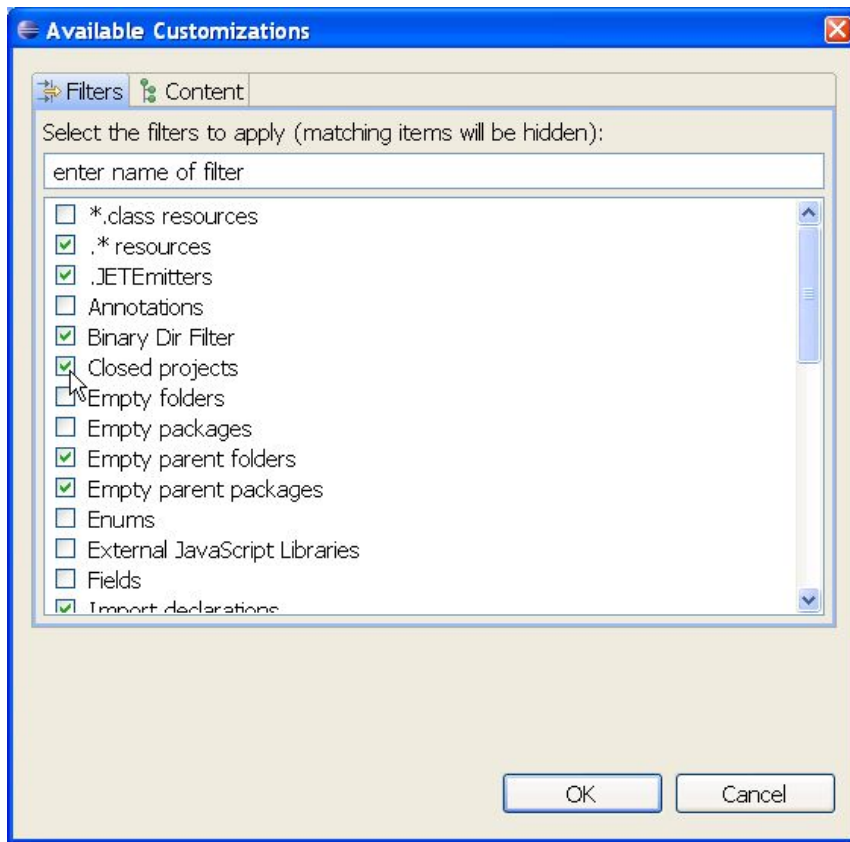
Another common customization is to use filters to choose which features to hide and show. For example, you might want to hide any closed projects in the **Project Explorer** view. To close a project, right-click the name of the project in **Project Explorer** and choose **Close Project** from the context menu. The icon decorating the project name changes from an open folder to a closed folder.



To hide closed projects, click the **View Menu** icon in **Project Explorer** (a small inverted triangle) and choose the **Customize View...** option.



In the **Available Customizations** dialog box that appears, select the "Closed projects" filter in the **Filters** tab and click the **OK** button.



Creating a Widget Project

You can create a new widget project in just a few minutes.

4.0 - Creating a New Project

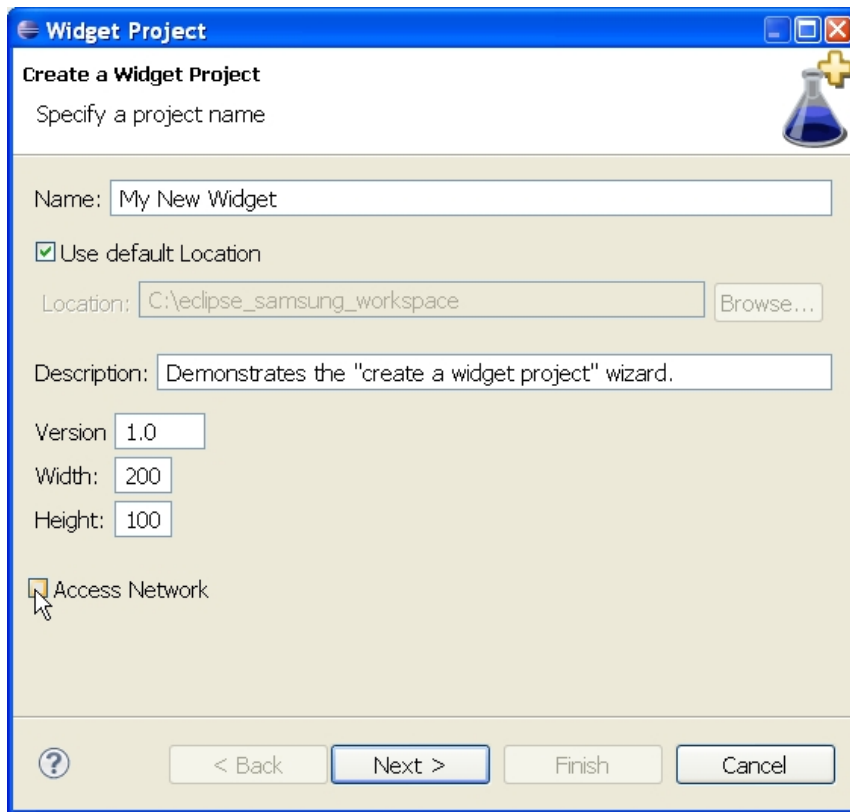
A widget project is populated with the folders and files you need to start developing a new widget.

Open the **Widget** perspective, if you haven't done so already, as described in *Launching the Samsung Mobile Widget SDK*.



4.1 - Configuring your Project

Choose **File > New > Widget Project**. In the **Widget Project** dialog that appears, specify a name for your project. You can use the default Eclipse workspace or you can specify a new location.



The configuration information defines several important things about your widget.

- **Description:** A brief description of your widget.
- **Version:** A version number that identifies your widget. This should be in the form *major_version.minor_version*.
- **Width and Height:** The size, in pixels, of the widget when it appears on the target device.
- **Access Network:** Designates whether the widget accesses the network. The default setting is TRUE. You should uncheck this box if your widget does not the network, as shown in the preceding screen shot.

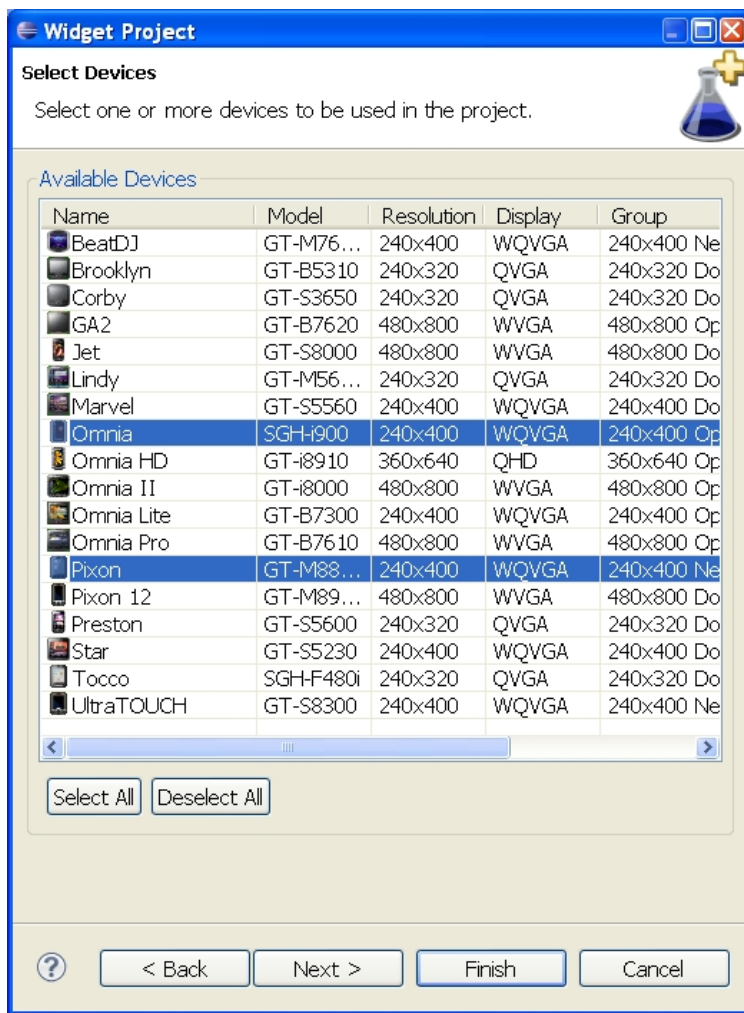
You can edit these settings later, if necessary, by opening the `project.xml` file in the root of your widget project.

The `project.xml` file is a generic configuration file that the Samsung Mobile Widget SDK uses to generate the `config.xml` file that is required by the finished widget. Different versions of the `config.xml` file are required for different platforms. The Samsung Mobile Widget SDK generates the correct form of `config.xml` whenever you package a project, based on your project's `project.xml` file and the platform of the target device.

When you have finished with this **Create a Widget Project** screen, click the **Next** button.

4.2 - Selecting Devices

The next screen in the "Widget Project" wizard enables you to choose one or more devices for your project:



You can select more than one device by using the usual **Ctrl+Click** and **Shift+Click** key combinations, as shown in the preceding screen shot.

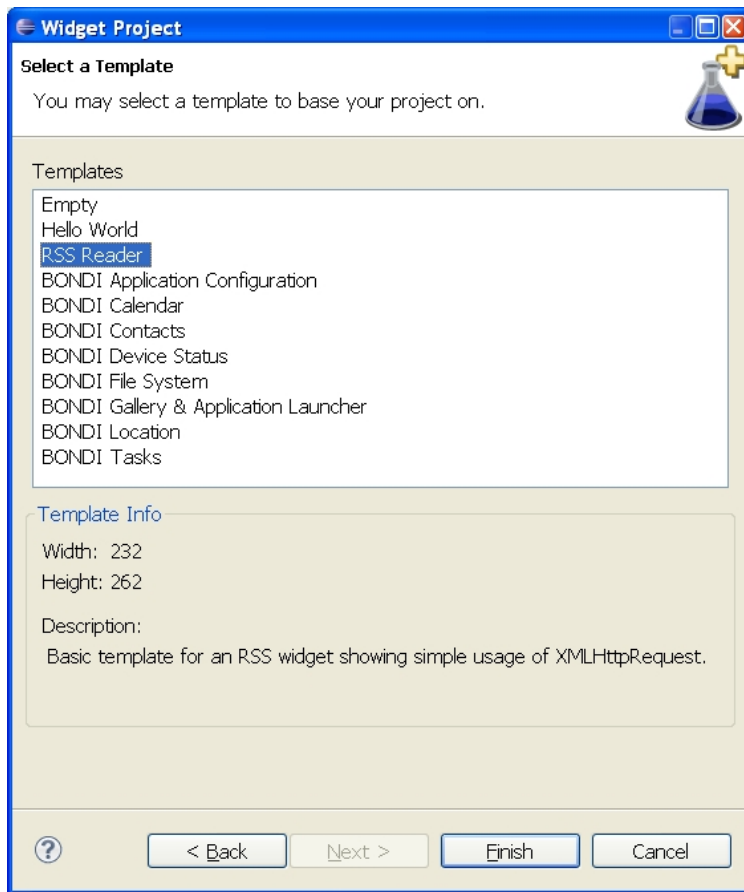
Note

You can add new devices to a project at any time by dragging them from the **Device** view to your project's **Device List** node in **Project Explorer**. Alternatively, you can right-click the **Device List** and choose **Add Device**.

When you have finished with this **Select Devices** screen, click the **Next** button.

4.3 - Selecting a Template

In the next dialog box, **Select a Template**, you can choose between a number of templates that provide a working foundation for your widget project.



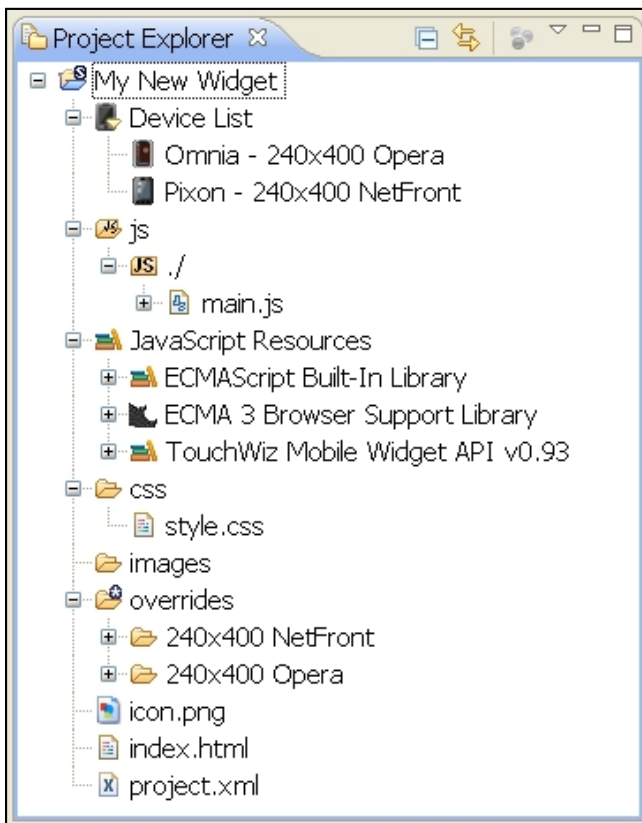
This list of templates changes dynamically as improvements are made to the SDK. The current set of templates may be different from what you see in this screen shot.

- The "Empty" and "Hello World" templates provide simple but valid starting points for your project.
- The "RSS Reader" widget implements many good ideas in cross-platform widget design. For more information, see *Best Practices for Widget Development*.
- The "BONDI" templates assist BONDI developers get started with different BONDI modules. For more information about BONDI, see *Developing for BONDI*.

Click the **Finish** button to create your Widget project.

4.4 - Structure of a Widget Project

Your new widget project looks like this in the **Project Explorer** view:



Most of the files in a new project are starting points for your code; as you would expect, the "Hello World" template creates a very simple widget.

- **Device List** - Lists the target devices for your widget. You can add to this list after you have created the project by dragging devices here from the Device view.
- **js > main.js** - Lists the JavaScript files that supply the functionality of your widget. The default file, `main.js`, is specified at the top of the default HTML file.
- **JavaScript Resources** - A read-only list of the libraries that are available to your widget. As you add devices to your project, libraries available on that device are automatically added to this directory. You can browse through the libraries to look for functions or documentation on functions in the supported libraries.
- **css > style.css** - Lists the CSS files defining the appearance of your HTML file(s). The default file, `style.css`, is linked to at the top of the default HTML file; you can supply more CSS files in this directory, if necessary, and link to them from the HTML file.
- **images** - Lists the PNG images used by your widget. PNG is the preferred file format, because PNG files support transparency and have ample color depth.
- **overrides** - Lists groups that define the screen resolutions and browsers supported by your project's devices. You can use these groups to customize your widget for particular browsers and resolutions. In this case, the `240x400 NetFront` group is for the Pixon device, and the `240x400 Opera` group is for the Omnia device. For more information, see [Supporting Multiple Devices with Overrides](#).
- **icon.png** - An icon that is displayed in the device's widget tray to identify your widget. The size of the icon varies with the screen size of the target device:
 - QVGA (240 x 320) - the maximum width of the tray is 56 pixels. A square icon should be no wider than 44 pixels.
 - WQVGA (240 x 400) - the maximum width of the tray is 60 pixels. A square icon should be no wider than 50 pixels.
 - QHD (360 x 640) - the maximum width of the tray is 81 pixels. A square icon should be no wider than 65 pixels.
 - WVGA (480 x 800) - the maximum width of the tray is 126 pixels. A square icon should be no wider than 102 pixels.
- **index.html** - Provides the HTML code that implements the appearance of your widget. Most widgets use only one HTML file. You might use more than one HTML file if your widget were implemented in multiple languages; the `index.html`

file would allow users to choose a language, and each language would be implemented in its own HTML file.

- **project.xml** - Supplies configuration information, including the information you specified when you created the project. If you needed to change a widget's width and height, for instance, you would need to edit this file.

Note

Your icon file must be named `icon.png` and your root-level HTML file must be named `index.html`.

4.5 - Previewing a Widget Project

The simplest way to preview your new widget is to double-click the device on which you would like to run the preview in the **Device List** in **Project Explorer**. There are a number of other ways to start a preview, though. The complete list is:

- Double-click the device in the **Device List**.
- Right-click the device and choose **Run As > Widget**.
- Click the small triangle next to the **Run** icon and choose **Run As > Widget**.
- Use the **Run** menu: **Run > Run As > Widget**.

The Samsung Mobile Widget SDK uses three different emulators to preview your widget. An emulator based on the WebKit browser handles the previews for devices that use Dolfin. For example, a preview of the "Hello World" widget on the Tocco device looks like this when the emulator appears:

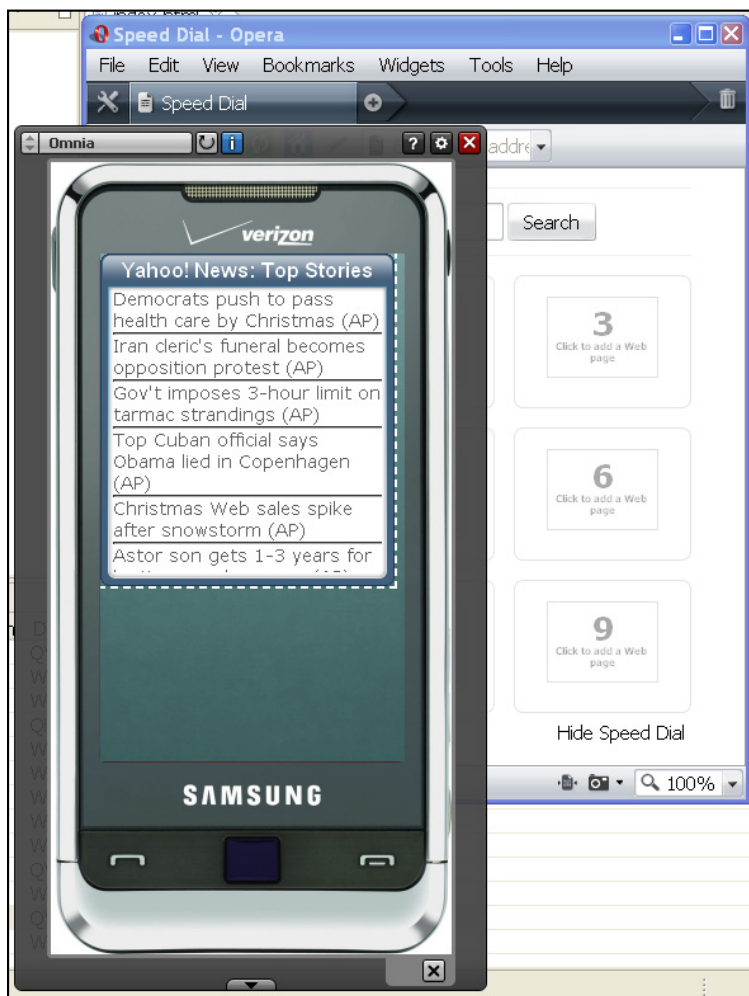


While you are previewing a widget with the WebKit emulator, you can right-click the widget and choose **Inspect Element** to bring up WebKit's **Web Inspector**. You can use **Web Inspector** to enable debugging, inspect the DOM, and research memory usage and loading time.

The SDK's second emulator is based on the ACCESS NetFront browser. When you preview your widget on a device that uses NetFront, a window named **NetFront Widget Emulator** appears, displaying the widget's icon, and the widget appears on the skin for the chosen device:



The third emulator is based on the Opera browser. This emulator is used for Windows Mobile devices. For example, a preview of the "RSS Reader" widget on the Omnia (Opera) device looks like this when the emulator appears:




The RSS Reader widget retrieves headlines from *Yahoo! News*. The Opera browser runs behind the emulated widget, as shown above. When you choose one of the stories in the list, the widget displays story details:



The back arrow above the story returns the reader to the list of stories. The plus button launches a browser that displays the full story on the *Yahoo! News* site.

If there is an error in your project—if there are missing or incorrect entries in your `project.xml` file, or if you have a JavaScript error, for example—the file with the problem is decorated with a red **x** in **Project Explorer**. If you try to preview or package a project that contains errors, an **Errors in Project** dialog box warns you that you may have overlooked a bug.

Transparency is supported in all of the emulators, but the NetFront emulator supports only 100% transparency.

You can use the **Run** button as a shortcut after you have run several widget previews. When you click the small arrow to the right of the **Run** button (), a list of recent launches appears. Choose the project you want to preview from this list.



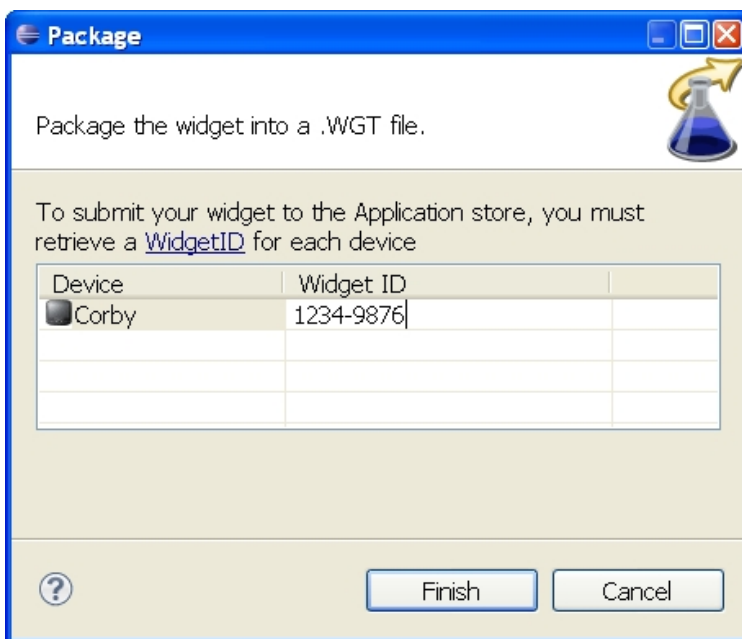
4.6 - Packaging a Widget

When you have written and debugged your widget, you are ready to *package* it. Packaging a widget project bundles all of the widget files together into a `.WGT` file. The packaged widget includes any customized files you may have included in the project's `overrides` directory and a device-specific `config.xml` file that the IDE produces automatically from your project's `project.xml` file.

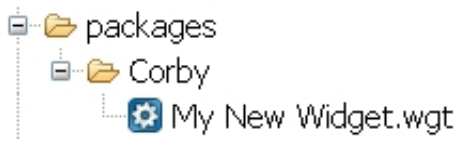
Bring up the Package dialog box by doing one of the following:

- Right-click a device in your `Device List` and choose **Package** from the context menu that appears. To package your widget for more than one device at a time, select the root of the project or the root of the `Device List` and choose **Package** from the context menu.
- Select a device, the `Device List` node, or the root of the project, and then choose the **Widget > Package** menu item.
- Select a device, the `Device List` node, or the root of the project, and then use the `CONTROL+SHIFT+P` key combination.

The Package dialog box gives you the opportunity to enter a Widget ID for any widget you plan to submit to the Samsung Application store. This ID appears in the `config.xml` file of your packaged widget. Click the **Finish** button to create a `.WGT` file for your widget, in the `packages` directory in **Project Explorer**.



In this example, the widget being packaged for a Corby device has an ID of "1234-9876." When you click **Finish**, a package appears in your project's `packages` directory:



A packaged widget is given the name of your project: in this case "My New Widget.WGT." You could rename this file, if necessary, by selecting the file and pressing F2, or by choosing **Rename** in the Context or **File** menus. A .WGT file is really a standard compressed file with a .WGT filename extension; to view its contents, you can rename the file by adding a .ZIP extension and then open it with any compression utility.

The config.xml file in the .WGT file that was just produced includes this line:

```
<widget xmlns="http://www.w3.org/ns/widgets" height="100" id="1234-9876" version="1.0" width="200">
```

As you can see, the ID specified in the **Packages** dialog box is included here in the packaged config.xml file. The width, height, and version number are taken from the project.xml file; you specified these values when you first created the project.

If you are packaging a widget for multiple devices and the package operation fails for one of the widgets, it also fails for the rest of the widgets in that operation. If errors occur when you are packaging multiple widgets, you should assume that the packaging operation failed for all of them.

You can use the Samsung Widget Test Page to test your widgets on your target device. You upload your widget to this page, send it to a handset by SMS WAP Push, and download it using Samsung Mobile Widgets for Partners. For a guide to this process, see the **Widget Test Page Guide** on the Samsung Mobile Innovator web site.

When you are ready to publish your widget to the Samsung Application Store, you can choose **Application Store** (either from the right-click menu in **Project Explorer** or from the **Widget** menu). This launches your default browser and displays the entry page at <http://seller.samsungapps.com>, where you can log in and upload your widget.



Supporting Multiple Devices with Overrides

5.0 - Developing for Multiple Devices

The Samsung Mobile Widget SDK makes it easy to customize your widget for different devices. Instead of having to create a new widget project for each new device, you can use *overrides* to customize your widget.

Whenever you need to provide a customized file in your project - an icon, image, CSS file, JavaScript tweak, HTML file, or `project.xml` - you should supply it in one of your project's *overrides* directories. When you preview or package your project, the IDE automatically picks up the customized files from the *overrides* directory for the target device.

5.1 - Device Groups

Every device you can use in the Samsung Mobile Widget SDK is a member of a group. Groups classify devices by their screen size and the browser they support. The *Group* column in *Device* view gives this information for each device.

Your project's *overrides* directory contains a subdirectory for each group to which your project's devices belong. For example, in the following screen shot the project uses five devices: a BeatDJ, an Omnia, a Star, and a Tocco. The *overrides* directory contains subdirectories for four groups. There are two Dolphin groups - one for the Tocco, whose screen is 240x320, and one for the Star, whose screen is 240x400. The 240X400 *NetFront* group supports both the BeatDJ and the Pixon device.

The Opera group supports the Omnia.



5.2 - Overrides

You might choose to lay out your widget differently for different screen sizes, or to use special-case code for different browsers. You can do this by putting modified files into the `overrides` directory for a specified group. The files at the root of your project are used by default whenever you run or package a widget; if you put a modified file into one of the `overrides` directories, that file will *override* the default file.

You could use a single widget project to support both a QVGA (240 x 320) device and a WVGA (480 x 800) device, for example. Let's say that the HTML and JavaScript files were the same for both of these devices; the HTML is simple, and the JavaScript contains any required special-case code for window resizing. In this case, you might need to support the different screen resolutions with customized icon files, separate images, different CSS files for positioning objects, and separate `project.xml` files containing the size of the widgets. A schematic view of the project might look something like this:

| | | |
|--------------------|-------------------------------|---|
| Device List | | |
| | <i>Device One (240x400)</i> | |
| | <i>Device Two (240x320)</i> | |
| | <i>Device Three (480x800)</i> | |
| js | | |
| | main.js | <i>(only JavaScript for this project)</i> |
| css | | |
| | style.css | <i>(default styles for this project)</i> |
| images | | |
| | myimage.png | <i>(default image for this project)</i> |
| overrides | | |
| | 240x320 BrowserA | |
| | css | |
| | | style.css <i>(styles for this smaller resolution)</i> |
| | images | |
| | | myimage.png <i>(smaller image)</i> |
| | js | |
| | | icon.png <i>(smaller icon)</i> |
| | | project.xml <i>(custom values for width and height of widget)</i> |
| | 480x800 BrowserB | |
| | css | |
| | | style.css <i>(styles for this larger resolution)</i> |
| | images | |
| | | myimage.png <i>(larger image)</i> |
| | js | |
| | | icon.png <i>(larger icon)</i> |
| | | project.xml <i>(custom values for width and height of widget)</i> |
| | icon.png | <i>(default icon for this project)</i> |
| | index.html | <i>(only HTML for this project)</i> |
| | project.xml | <i>(default configuration information for this project)</i> |

In this case, whenever you previewed or packaged your widget for *Device One*, the IDE would pick up the default files: the styles, images, icon, HTML, XML, and JavaScript. When you previewed or packaged the widget for Device Two, the IDE would build a widget using the default HTML and JavaScript, but would override the other files with the contents of the *240x320 BrowserA* directory. Similarly, when you previewed or packaged the widget for Device Three, the IDE would build a widget using the default HTML and JavaScript, but would use the contents of the *480x800 BrowserB* directory.

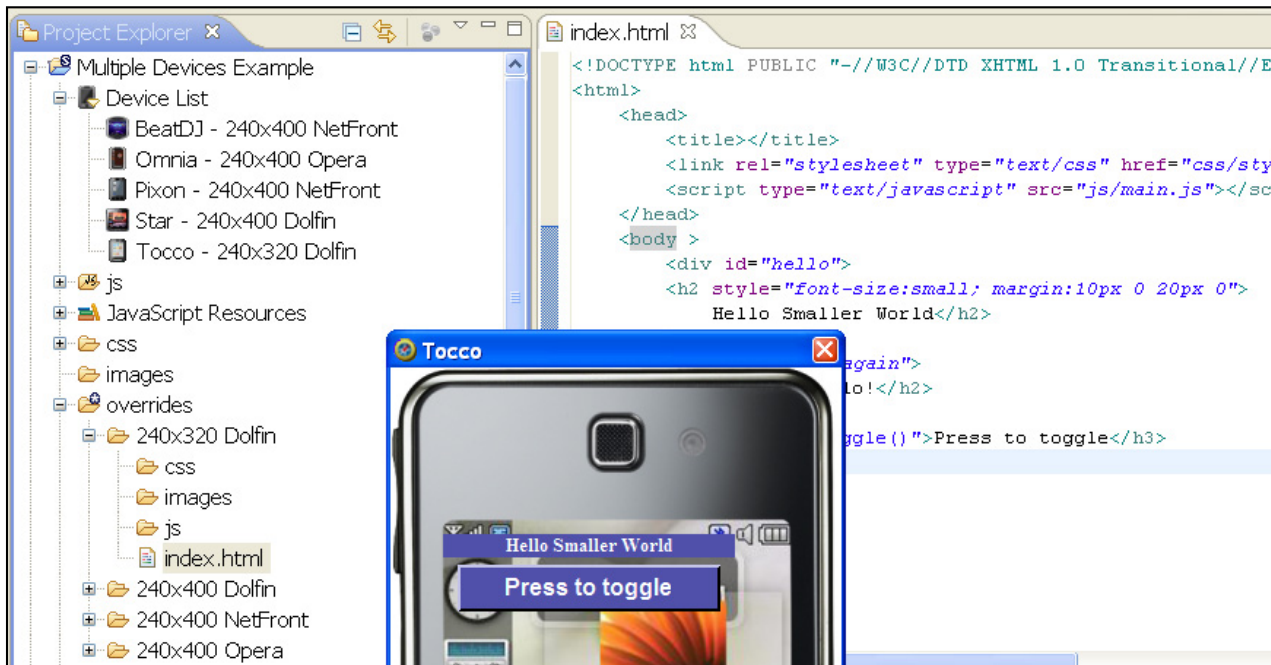
Example: Overriding index.html

For example, you could change the `index.html` file for the Tocco's smaller screen to say "Hello Smaller World." Copy and paste the project's `index.html` file into the `overrides > 240x320 Dolphin` directory (or CTRL + Drag), edit the

text in the HTML file, and preview the result on the Tocco and the other devices. The changes you make to the files in the overrides > 240x320 Dolphin appear only when you preview on the Tocco - the widgets for the other devices have not changed. Note that you do not need to copy any JavaScript or CSS files in this case - the only file you need to put in the 240x320 Dolphin directory is the file that is changing for this override.

To preview your widget on a particular device, double-click the device in the Device List in Project Explorer.

The output on the Tocco emulator looks like this:



The new copy of index.html in the 240x320 Dolphin directory is picked up automatically when you preview the Tocco. Notice that the size of the text in this example has been changed with an inline style to prevent line-wrap problems.



Tutorial Sample

This section of the *Samsung Mobile Widget Development Guide* leads you through the process of creating a simple widget. This widget uses simple CSS and JavaScript to animate buttons and change the visibility of paragraphs - it is only slightly more complicated than the "Hello World" template.

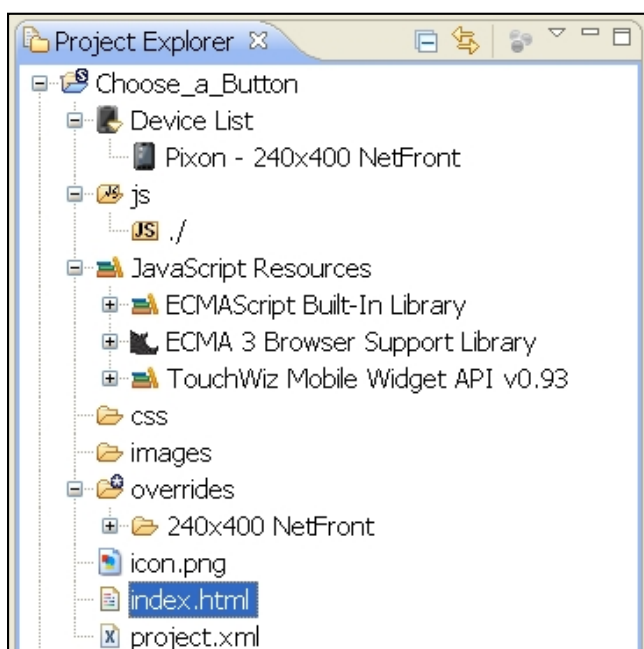
6.0 - Create the Project

First, create a widget project, as outlined in the preceding sections. There is one important difference in the procedure given in the following list, however. When you create this project, you should not use a template as a starting point.

1. Ensure that you are using the **Widget** perspective.
2. Choose **File > New > Widget Project** and give your project a name. This sample uses the name "Choose_a_Button".
Give your project a unique ID and add other configuration information. For the "Choose_a_Button" project, use 180px for the height and width.
Uncheck the **Access Network** checkbox and click the **Next** button.
3. Select the **Pixon** from the list of **Devices** on the next dialog box and click **Finish**. Do not click the **Next** button - we do not want to start with any of the supplied templates for this project.

The "Choose_a_Button" project appears in the **Project Explorer** view. This project is even simpler than the project created by the "Hello World" template; it has an empty **HTML** file.

When you expand the nodes of the "Choose_a_Button" project in **Project Explorer** view, it should look like this:



6.1 - Edit the CSS File

Add a file named `style.css` to the `css` directory, and add the following to change the appearance of any paragraph text in the widget:

```
body {  
    background-image:url(../images/abstract_180x180.png);  
}  
  
p {  
    color:#882222;  
    font-family:sans-serif;  
    font-size: medium;  
    font-weight:bold;  
    position:relative;  
    margin-left:20px;  
}  
  
.output {  
    position:absolute;  
    top:90px;  
    left:0;  
}
```

The `body` section loads a background image, superseding the default transparent color. The image looks like this:

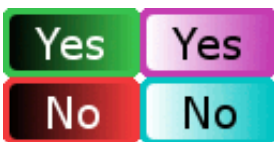


The `p` section defines the color, font characteristics, and position for default paragraphs. The `.output` class is used to modify the paragraphs that appear when the user presses a button.

6.2 - Create and Import the Images

Create PNG files for the images you will use as buttons in this widget. We need a "Yes" button and a "No" button. Each button also needs a version that gives the user feedback when it has been selected.

The four buttons might look like this:



In this example, these buttons are named `yes.png`, `yes_inverted.png`, `no.png`, and `no_inverted.png`.

After you have created the buttons, you can drag them from a window in the file system to the **images** directory in **Project Explorer** to import them into your project.

6.3 - Create and Import the Icon

Create a PNG file that will serve as an icon for this widget. The screen size of the Samsung Pixon is given in the **Device view** as 240 x 400 - therefore, according to the list given in *Structure of a Widget Project*, the icon should be no wider than 50 pixels. The new file should be named `icon.png` - if you give it another name, you should change the `icon` entry in `project.xml` to match the new name.

Replace the default `icon.png` file in the "Choose_a_Button" project with your new icon by dragging it to the root of the "Choose_a_Button" project in **Project Explorer**. Here is one possible icon for the project:



6.4 - Edit the JavaScript File

Create a `main.js` file in the `js` directory.

The **Choose_a_Button** widget creates two buttons and reacts to user input. The code that animates the buttons is in the `mouseClick(button)` function. Add the following to the `main.js` file.

```
/*=====
 * The start() function is called when the onload() event occurs.
 * It ensures that the responses to button presses are initially
 * hidden.
 *=====*/

function start()
{
    document.getElementById("yesreaction").style.visibility="hidden";
    document.getElementById("noreaction").style.visibility="hidden";
}

/*=====
 * The mouseClick() function is called when the onclick() event
 * occurs. It animates the buttons and changes the visibility
 * of the paragraphs that display responses to the click.
 *
 * button parameter - string identifying the button that was clicked
 *=====*/

function mouseClick(button)
{
    if (button=="yesbutton") {
        // change the button image for 300 milliseconds
        document.yesbutton.src = "images/yes_inverted.png";
        setTimeout("document.yesbutton.src = 'images/yes.png'",
                    300);
        // ensure other button is in unselected state
        document.nobutton.src = "images/no.png";

        // change visibility of button responses
        document.getElementById("yesreaction").style.visibility="visible";
        document.getElementById("noreaction").style.visibility="hidden";
    }
}
```

```

    }
    else if (button=="nobutton") {
        document.nobutton.src = "images/no_inverted.png";
        setTimeout("document.nobutton.src = 'images/no.png'",
                    300);
        document.yesbutton.src = "images/yes.png";

        document.getElementById("yesreaction").style.visibility="hidden";
        document.getElementById("noreaction").style.visibility="visible";
    }
    else {
        document.write("Something went wrong.");
    }
}

```

6.5 - Edit the HTML File

Finally, you need to edit the `index.html` file so that the widget will use the images, CSS, and JavaScript you have just put in place.

When the page loads, the `onload` event calls the JavaScript `start()` function to hide the sentences that are displayed when the user clicks a button. The buttons are animated when the buttons are clicked. The `onclick` event calls JavaScript that is implemented in the `main.js` file. Finally, the responses to the button clicks are displayed. These output paragraphs use the `output` class to ensure that the "yes" and "no" responses appear in the same place.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title></title>
        <link rel="stylesheet" type="text/css" href="css/style.css" />
        <script type="text/javascript" src="js/main.js"></script>
    </head>
    <body onload="start()">
        <p>Would you care to <br />press a button?</p>
        </img>
        </img>
        <p id="yesreaction" class="output">
            Thank you <br />for your input.</p>
        <p id="noreaction" class="output">
            You are trying <br />to confuse me.</p>
    </body>
</html>

```

6.6 - Preview the Widget

Now that the images, CSS file, JavaScript file, and HTML file are in place, it is possible to preview the widget. Double-click the Pixon device in the **Device List** in **Project Explorer**. The emulator should look like this:



Use your mouse to verify that the buttons animate correctly and that they handle the mouse clicks.

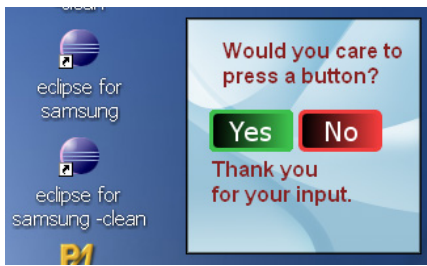
6.7 - Package the Widget

When you are satisfied with "Choose_a_Button," package it and verify that it works successfully outside the emulator. Right-click your target device in the **Device List** and choose **Package**. Then follow the directions in *Packaging a Widget* to produce a `.WGT` file and test your widget on your target device.

If you have installed the Opera browser, you can test a packaged widget on your computer.

First, drag a device that runs Opera (the Omnia, for instance) from the **Devices** view to your project's **Device List**. Any widget packaged for an Omnia device will run on Opera. Right-click the Omnia device and choose **Package**.

Double-click the `.WGT` file inside the `packages` directory to launch the Opera browser, or drag the `.WGT` file into an Opera window. Drag the Opera window away from the widget to verify that the widget is running on your desktop. The following screen shot shows the `Create_a_Button` widget running on the desktop:



To close the widget, right-click it and choose **Close**.



All contents Copyright © Samsung Electronics Co., Ltd.



Debugging Widget Projects

This section of the documentation discusses the debuggers that are integrated into the Samsung Mobile Widget SDK, and how to install an external debugger when your widget project requires it.

7.0 - Platforms and Browsers

You can choose among several good debuggers when you are writing your widget. Most of the time - especially when you pay careful attention to interoperability/cross-browser issues - your JavaScript problems can be solved by any debugger.

In browsers with debuggers, you can browse the DOM, examine source code, and set breakpoints as needed. You can also use the browser's reload mechanism (for pages or widgets) to start over whenever you like. However, you must edit your widget in the Samsung Mobile Widget SDK, not in the debugger. After editing, re-run the widget from the SDK to relaunch the browser with all your changes and overrides included.

The most popular browsers on current TouchWiz devices are:

- NetFront (ACCESS), for some Samsung Handset Platform (SHP) devices
- Opera, for Windows Mobile devices
- Dolfon (WebKit), for some SHP devices

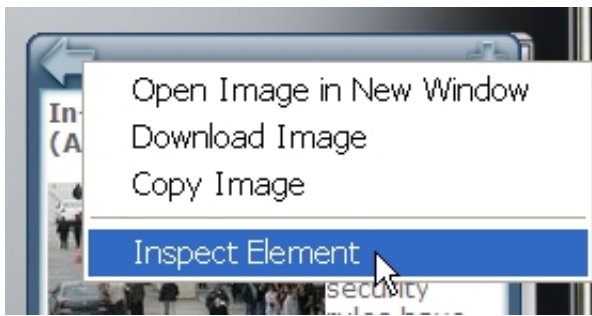
Both the WebKit emulator (for Dolfon devices) and the Opera emulator have built-in debuggers. If you are developing for either of these platforms, using the internal debugger is usually the easiest way to debug your widget.

You can preview your widget on any web browser you have installed by setting up a new run configuration for your project. For more information, see *Previewing on a Browser*.

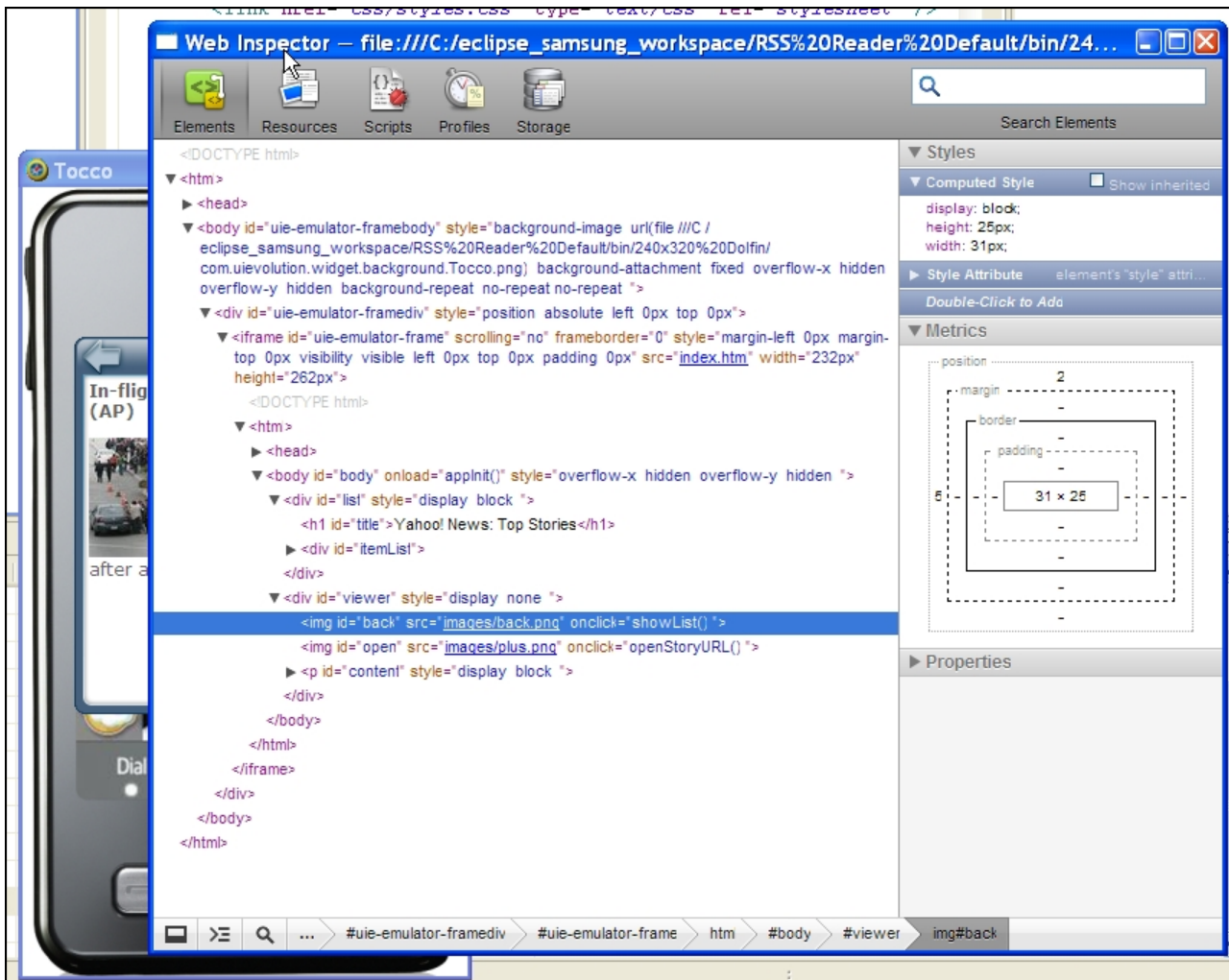
Most users of the Samsung Mobile Widget SDK attempt to develop widgets that will reach as many TouchWiz devices as possible. The most successful widgets will run on any platform and on any browser. Building widgets like this requires careful attention to interoperability issues, as outlined in *Best Practices for Widget Development*.

7.1 - Debugging using WebKit

The WebKit browser supports Dolfon devices. You can use WebKit to debug your Dolfon widget by right-clicking an element in the widget while it is being emulated and selecting **Inspect Element**.



The Inspect Element menu item launches WebKit's Web Inspector:



In this screen shot, the element being inspected is the left arrow (back) button in the RSS reader application. The Metrics window in the Web Inspector shows the size and relative position of the element.

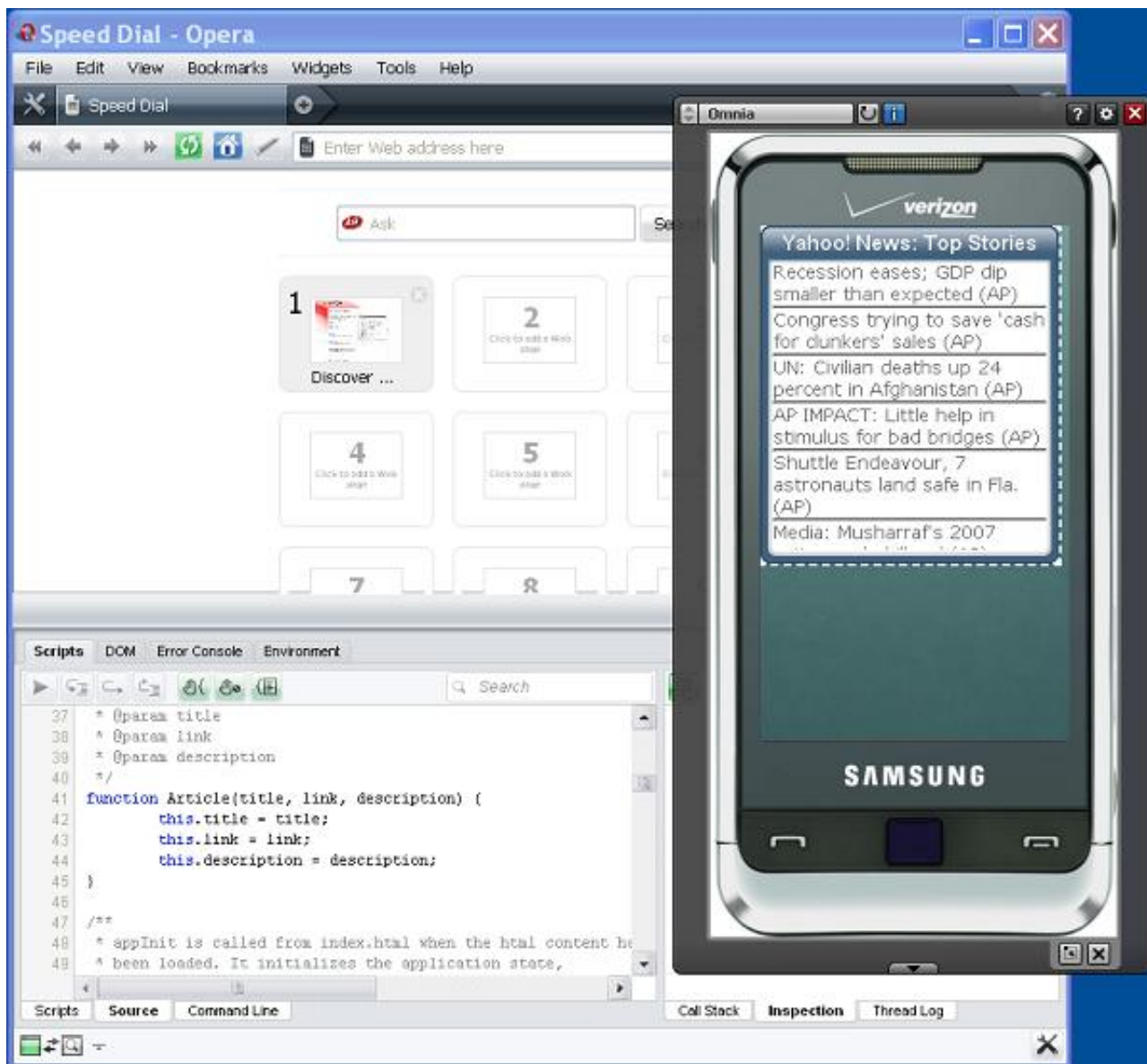
The Web Inspector makes many tools available to you. When you click the headings at the top of the window (Elements, Resources, Scripts, Profiles, or Storage), you are given the opportunity to enable various tools: for example, when you click Scripts, you can choose whether to enable debugging for this session or for all sessions.

7.2 - Debugging for Windows Mobile using Opera

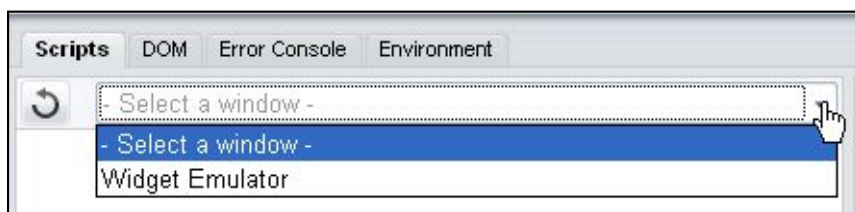
Samsung's Windows Mobile devices use the Opera browser to display widgets. Therefore, the Samsung Mobile Widget SDK launches Opera automatically whenever you double-click a Windows Mobile device in Device List (or choose Run As > Widget). Debugging on Opera can be a good way of uncovering problems that might occur on those devices.

Dragonfly is the name of Opera's debugger. To open the Dragonfly views, choose **Run As > Widget**; then, after Opera launches, select **Tools > Advanced > Developer Tools** in Opera. A new work area appears at the bottom of the screen, labeled "Loading Dragonfly." When Dragonfly has finished loading, debugging windows appear.

The following screen shot shows Dragonfly displaying information about the "RSS Reader" widget, while the Opera Speed Dial feature is displayed at the top of the screen. Dragonfly is displaying part of the widget's `main.js` file.

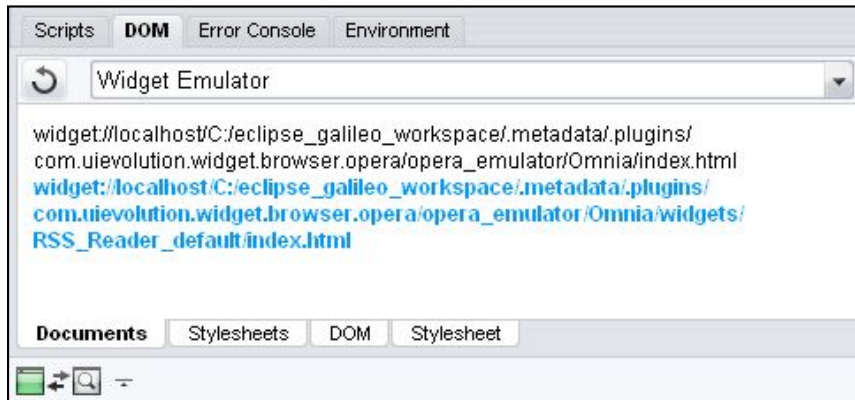


When you first run Dragonfly, you will see a drop-down window in the Scripts tab labeled "Select a window." Click the down arrow and choose "Widget Emulator" to start your debugging session.



If the "Select a window" drop-down window is empty, please close other instances of Opera that you have running.

You can use the Dragonfly views to see the HTML and script files for both the emulator and your widget. You should be sure to work with the widget files when you are debugging, not the emulator files. To do this, click the **DOM** tab above the Dragonfly work area and then the **Documents** tab at the bottom of the window, and choose the path that ends with `...DeviceName/widgets/ProjectName/index.html`, not `...opera_emulator/DeviceName/index.html`.



The Scripts view also shows the script for both the emulator itself and for your widget. Normally, your source files will be under the second `index.html` tree control.

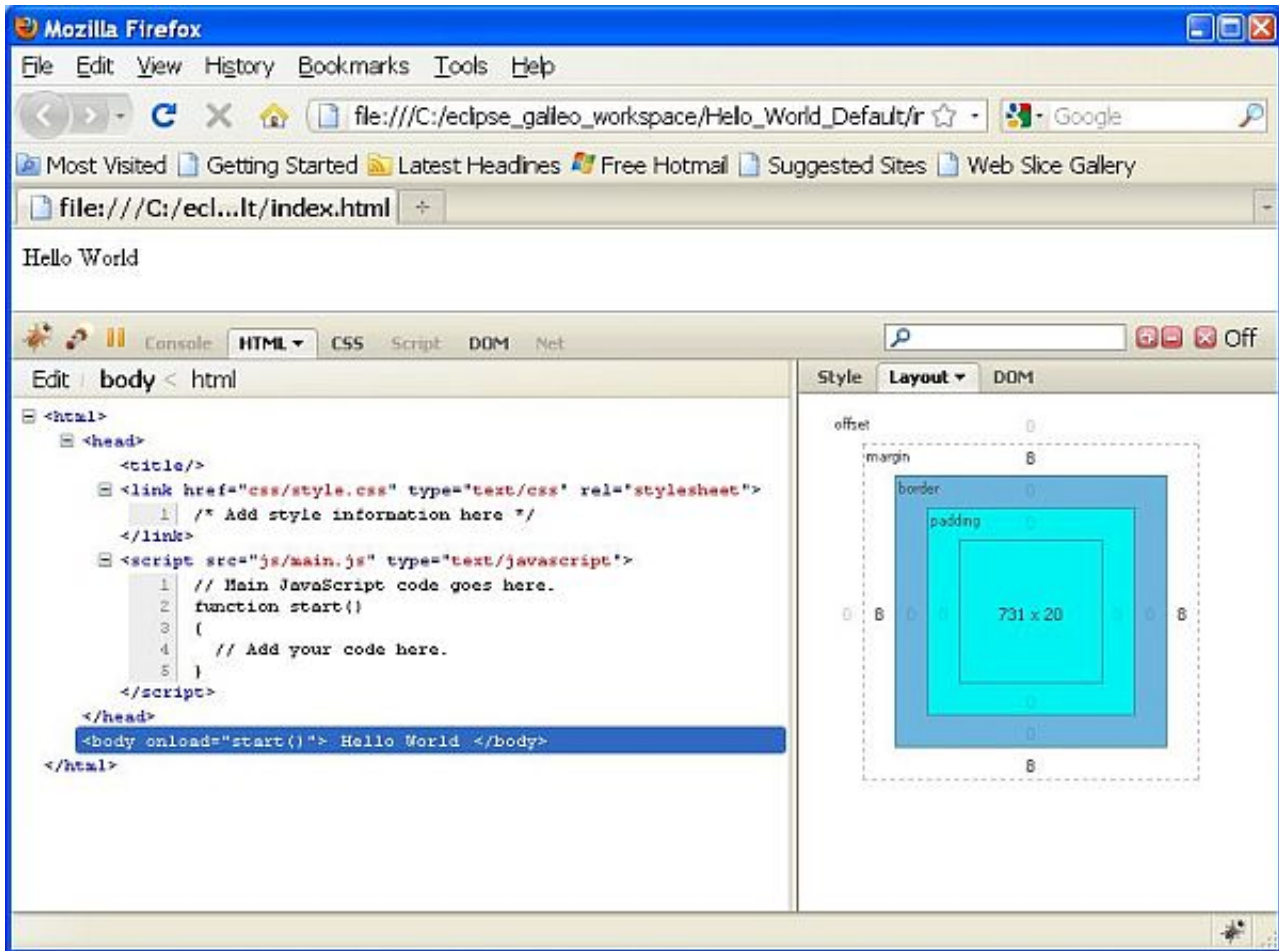
Information about Dragonfly can be found at <http://www.opera.com/dragonfly>. A discussion of using Dragonfly to debug widgets is here: <http://dev.opera.com/articles/view/debugging-widgets-using-opera-dragonfly/>.

7.3 - Debugging with Firefox and Firebug

The Firebug debugger is an add-on for Mozilla's Firefox browser. It has many useful features, including:

- HTML editor
- CSS metrics
- JavaScript debugging, with breakpoints, stepping through code, expression watching, stack tracing, and more.
- DOM exploration

The following screen shot shows Firebug displaying information about the "Hello World" code, while the output is displayed at the top of the screen. The Layout view is a graphical representation of the current styles.



Firefox does not allow you to use XMLHttpRequest from a local file. If your widget uses XMLHttpRequest and you want to debug it in Firebug, add the following (temporarily) to your code. This will present a dialog box when the request is made:

```
try {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");
} catch (e) {
    // There should be no errors. This try/catch construction should allow
    // you to run this code in any browser.
}
```

For more information about working with XMLHttpRequest, see *XMLHttpRequest Issues*.

You can learn more about Firebug and download it here: <http://getfirebug.com/>





Widget-Development Tips

This part of the *Samsung Mobile Widget Development Guide* discusses advanced topics for widget development.

8.0 - Widget SDK User Interface Tips

This section introduces several ideas for using the Samsung Mobile Widget SDK in ways you may not have discovered on your own.

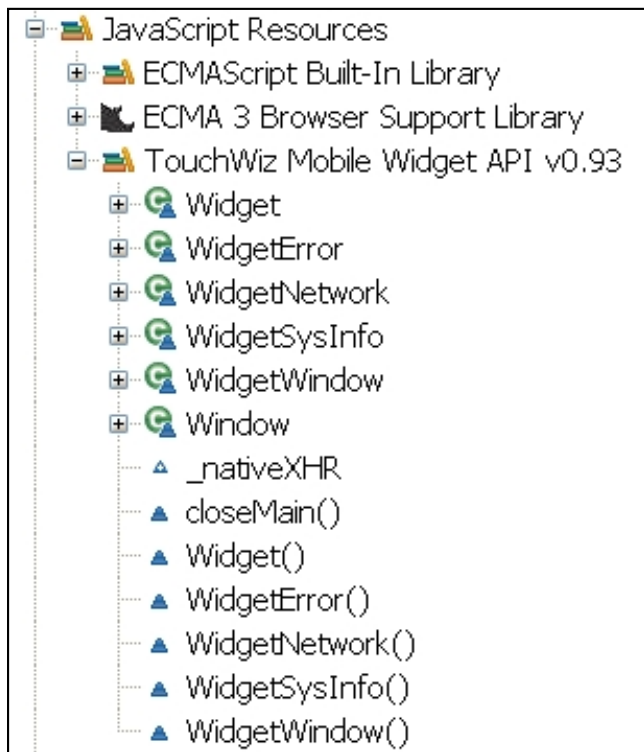
Viewing the Error Log

You can add an **Error Log** view to the Instance Management views at the bottom of the IDE window. You can use the **Error Log** to track problems as you work with the Samsung Mobile Widget SDK. To add this view:

1. Select **Window > Show View**
2. Choose **Other** to display the **Show View** dialog box
3. Expand the **General** node and select **Error Log** from the list
4. Click the **OK** button

Viewing TouchWiz API

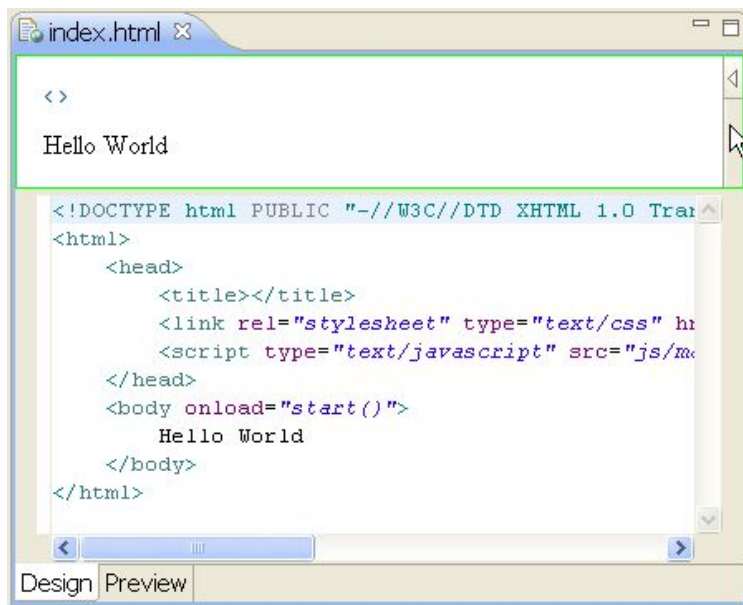
You can look at the implementations of the TouchWiz API whenever you like, by expanding the **JavaScript Resources** node in the **Project Explorer** view and then expanding the **TouchWiz Mobile Widget API** node:



Both the comments in this code and the implementation details can help you develop your widgets. Many of these API elements work in different ways on different platforms; you should implement cross-browser strategies in your code to accommodate these differences. For information, see *Best Practices for Widget Development* and *Samsung Mobile Widget Specification V0.93*.

Using Open With Web Page Editor

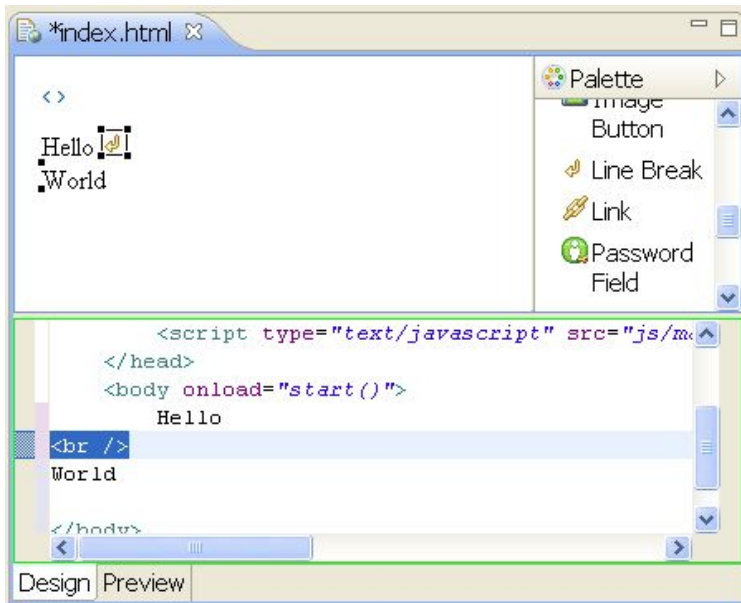
When you right-click an HTML file in **Project Explorer** view, you can choose **Open With > Web Page Editor** and work with the file in an interesting way. The Source Editor view opens with two windows: the top window shows a preview of the HTML output and the bottom window shows the editable source code:



There are tabs along the bottom of this view. In the preceding screen shot we are looking at the **Design** tab. For a better preview of the HTML file you can click the **Preview** tab.

The mouse cursor in the preceding screen shot is near a small left arrow that you can click to reveal a palette of visual items

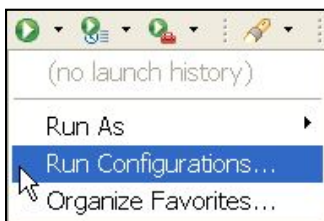
you can drag into the editor. For example, in the following illustration, the **Line Break** item has been dragged from the **HTML 4.0** menu and dropped between "Hello" and "World":



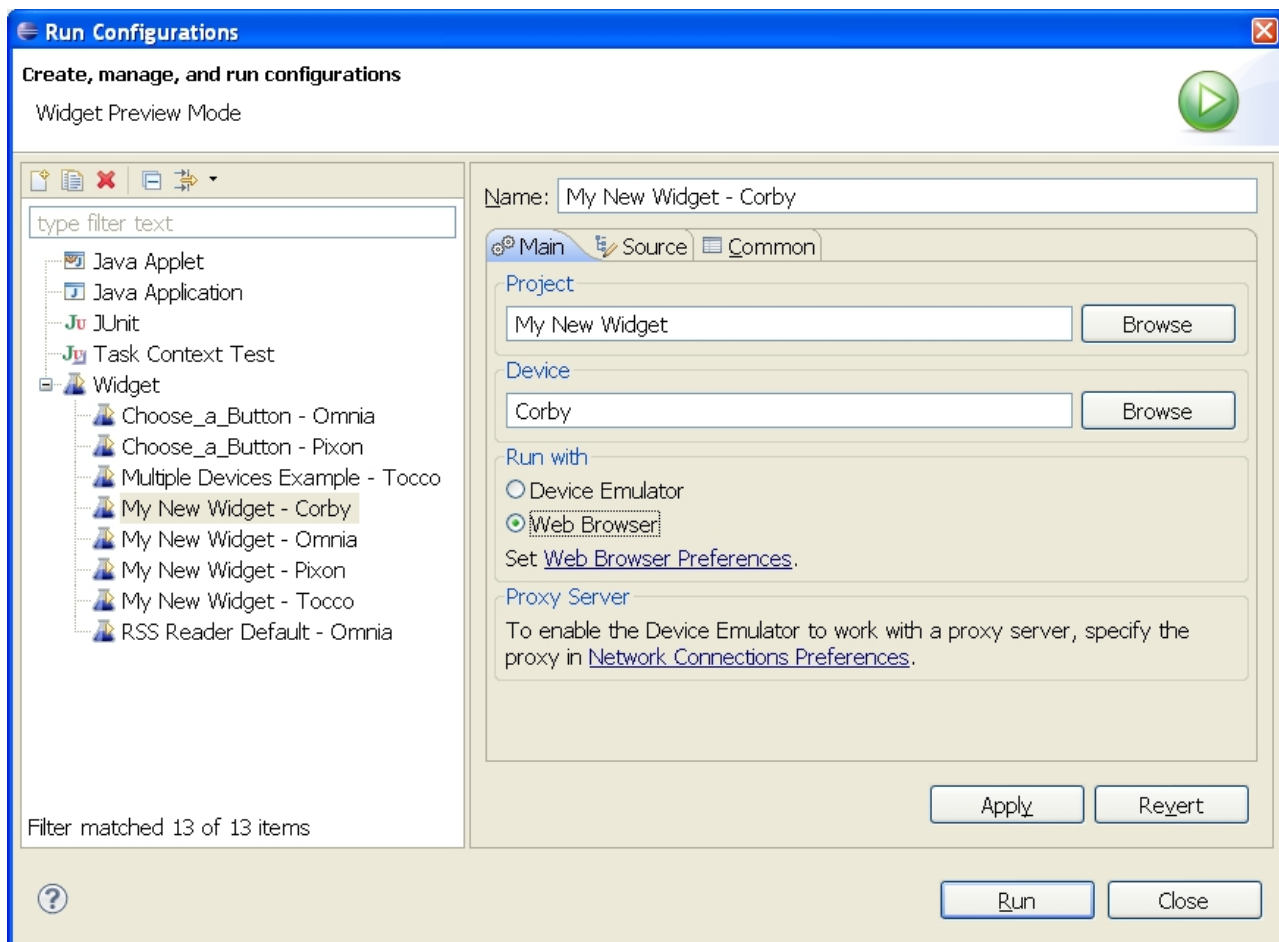
8.1 - Previewing on a Browser

Sometimes you might prefer to preview your widget on a web browser instead of in the IDE's device emulator. To do this, you must set up a new *run configuration* for your widget project.

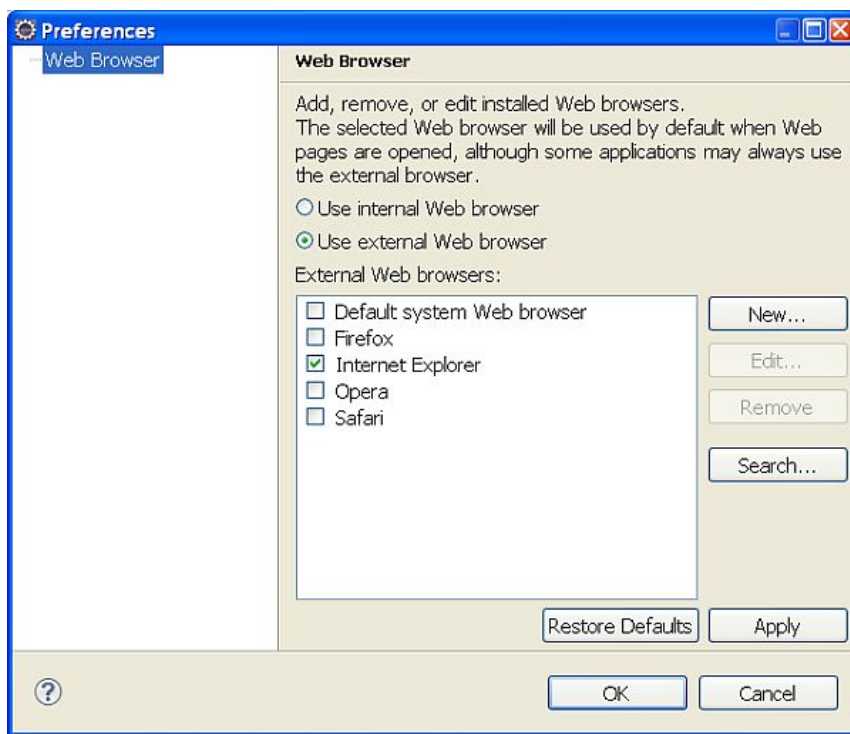
First, select the name of the project in **Project Explorer** that you would like to run in a browser. Then select the small triangle next to the **Run** icon and choose the **Run Configurations...** item.



Open the **Widget** node in the **Run Configurations** dialog box that appears and select one of the *Project - DeviceName* entries. Give this new run configuration a descriptive name. The **Project** and **Device** entry fields are filled in with values taken from the project that was selected when you opened this dialog box. Select "Web Browser" in the **Run with** section. The dialog box should look something like this:



Click the Web Browser Preferences link to choose the web browser on which you would like to test your widget:



Ensure that the "Use external Web browser" radio button is selected, as shown above.

To add a new browser to this list, click the New... button and navigate to the browser's location on your hard drive.



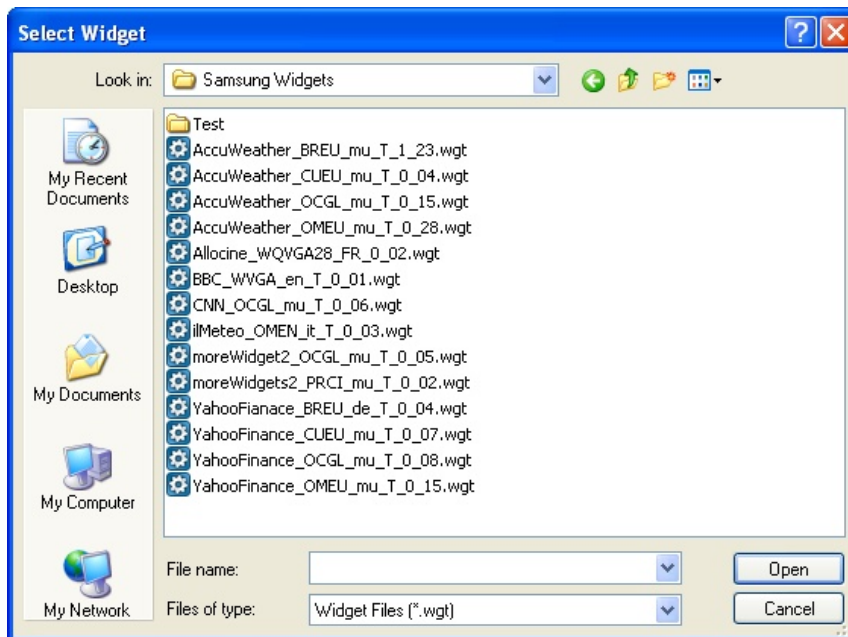
Browsers handle security with XMLHttpRequest in different ways. For example, the Dolphin browser used by many Samsung devices (based on the WebKit browser used in the Samsung Mobile Widget SDK) allows XMLHttpRequest access to remote sites, but other browsers may not. For more information, see *Working with Multiple Browser Platforms*.

8.2 - Importing an Existing Widget

You may sometimes need to bring an existing widget into the Samsung Mobile Widget SDK. Follow this procedure to import it successfully:

1. Choose **File > Import...**
2. Expand the Samsung node and choose **Widget**. Click the **Next** button.
3. Click the **Browse** button and navigate to the directory containing the **.WGT** file you want to import.
4. Select the desired **.WGT** file and click the **Open** button.
5. Click the **Finish** button.
6. Drag a device into the Device List for the new project.

The following screen shot shows a list of available widget files that could be displayed when a user clicks the **Browse** button.



When you import an existing widget, the Samsung Mobile Widget SDK reads the widget's `config.xml` file and transfers the values to the new project's generic `project.xml` file. Different versions of a widget's `config.xml` file are required for different platforms. When you package (export) a project, the Samsung Mobile Widget SDK generates the correct form of `config.xml`, based on the project's `project.xml` file and the platform of the target device. For information about packaging a project, see *Packaging a Widget*.

Note for earlier widgets developed using the Opera Widget Emulator:

Widgets written to work with the Opera Widget Emulator may have the following line of code in the head of `index.html` to allow them work with the Opera Widget Emulator:

```
<script type="text/javascript">if(parent.emulator)parent.emulator.begin(window);</script>
```

You should remove this line from your `index.html` file. The Samsung Mobile Widget SDK automatically works with the Opera Widget Emulator. Calling `emulator.begin` again overwrites the Samsung Mobile Widget API widget object, causing incorrect emulation behavior for `setScroll` and `resizeWindow`.

8.3 - Best Practices for Widget Development

You can use these "best practices" to reduce development time and avoid frustrations. Following these tips ensures that your use of the Samsung Mobile Widget SDK is efficient and that your widgets will run on multiple platforms.

1. Test your widget in multiple browsers. See *Working with Multiple Browsers*, below, for more information.
2. Use overrides, not different projects, to support multiple screen resolutions and browsers. For an example of handling window resizing in your code, see *Window Resizing*, below.
3. Handle differences caused by multiple screen resolutions in your CSS files, rather than by hard-coding the sizes in JavaScript or HTML.
4. Be careful about differences in how browsers handle events. See *Event Handling*, below, for more information.
5. There are a number of cross-platform issues with the XMLHttpRequest DOM API, including security restrictions and the possibility of a null response. For more information, see *XMLHttpRequest Issues*, below.

For technical details about some of the requirements of cross-platform design, see the *Samsung Mobile Widget Specification V0.93*.

Working with Multiple Browser Platforms

New mobile devices use many different web browsers. Samsung devices might use NetFront (ACCESS), Opera, or Dolfin (WebKit), and more browsers are likely to be supported soon. It is important to test your widget on the browsers for all of your target devices.

JavaScript engines vary from browser to browser and sometimes from version to version within a single browser. This means you may not be able to use some browser objects and their corresponding event handlers to develop your widget, depending on the target browser and its JavaScript implementation.

You can run your widget in the emulator or in any browser you have set up in the **Run Configurations** dialog box (see *Previewing on a Browser*). In addition, you can package your widget and run it in Opera.

To package your widget, right-click a device in your **Device List** and choose **Package** from the context menu (or use the **CONTROL+SHIFT+P** key combination). You can find a complete description of packaging widgets in *Packaging a Widget*.

To preview your widget, double-click the `.WGT` file or drag the file into an Opera window. Please note that Opera caches the widget, so you will need to close both the widget and Opera when making changes.

Window Resizing

A common cross-browser issue is window resizing. SHP devices support the `resizeWindow` API, but Windows Mobile devices may support only the `resizeTo` API. You can solve this problem with a `try/catch` construction like this:

```
try {
    widget.window.resizeWindow(width,height);
} catch(e) {
    window.resizeTo(width,height);
}
```

You should use overrides to support devices that have different screen sizes. For more information, see *Overrides*.

Event Handling

This is a good idiom to use in event handlers:

```
function doSomething(e) {  
    if (!e) {  
        var e = window.event;  
        // e gives access to the event in all browsers  
    }  
}
```

If you use this idiom with inline event handling, you must pass the event to the handler manually.

Restrict event handling to the W3C DOM Level 2 Event specification: <http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/>

For additional information, use resources such as <http://www.quirksmode.org/js/introevents.html> and http://www.quirksmode.org/js/events_access.html.

XMLHttpRequest Issues

Various browsers have different security restrictions when using `XMLHttpRequest` from a local file.

- IE - allowed with warning dialog
- Safari - allowed
- Opera - disallowed - test as widget instead
- Firefox - disallowed - add the following (temporarily) to your code. This will present a dialog box when the request is made:

```
try {  
    netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");  
} catch (e) {  
    // There should be no errors. This try/catch construction should allow  
    // you to run this code in any browser.  
}
```

The `responseXML` from a `XMLHttpRequest` can be null if the request header has not been initialized correctly. The results can vary on different browsers. To correctly get the desired response, set the content header before making the request. For example:

```
request.setRequestHeader("Content-type", "text/xml");
```

The `XMLHttpRequest` API is not currently supported by Internet Explorer. Your code should create a cross-browser `XMLHttpRequest` object by creating a new `ActiveXObject`, when necessary, as in the following sample from the RSS Reader template's `network.js` file:

```
function getXMLHttpRequest() {
```

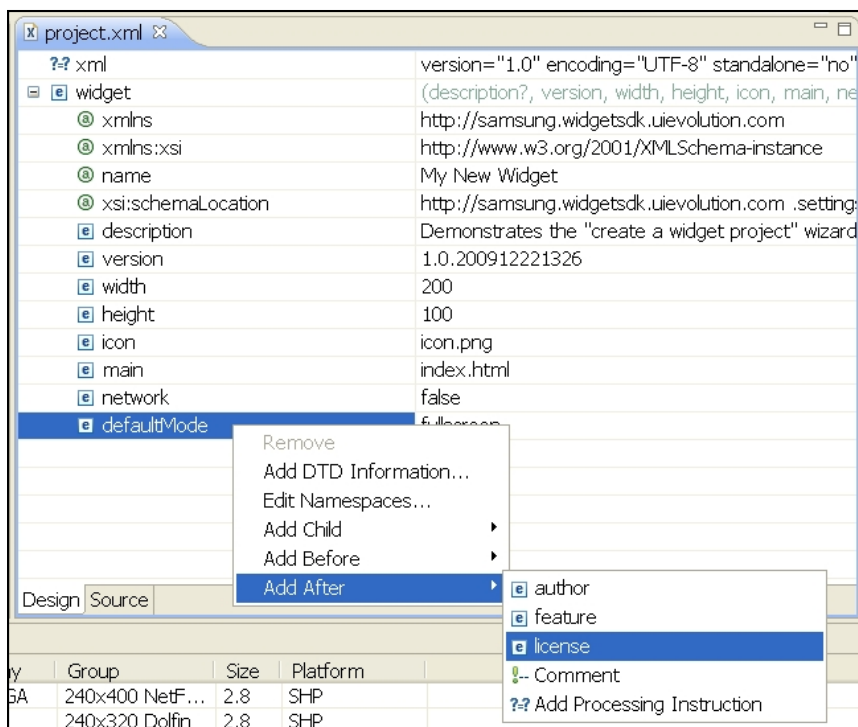
```

var xmlHttp = null;
try {
    xmlHttp = new XMLHttpRequest();
}
catch (e) {
    try {
        xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (e) {
    }
}
return xmlHttp;
}

```

Editing Project.xml

It is possible to add several tags to your widget's `project.xml` file, in support of W3C specifications for `config.xml`. These tags are `<author>`, `<license>`, and `<feature>`. To add these tags, edit `project.xml` in design mode, right-click the `defaultMode` tag, and select **Add After**:



Opera Debugging

Developers who are targeting the Opera browser often use calls to the `opera.postMessage` function as a debugging technique. You should not leave those calls in your code when running your widget on a device. Calls to `opera.postMessage` will produce an exception. Protect any debugging calls by surrounding them with `try/catch` statements.

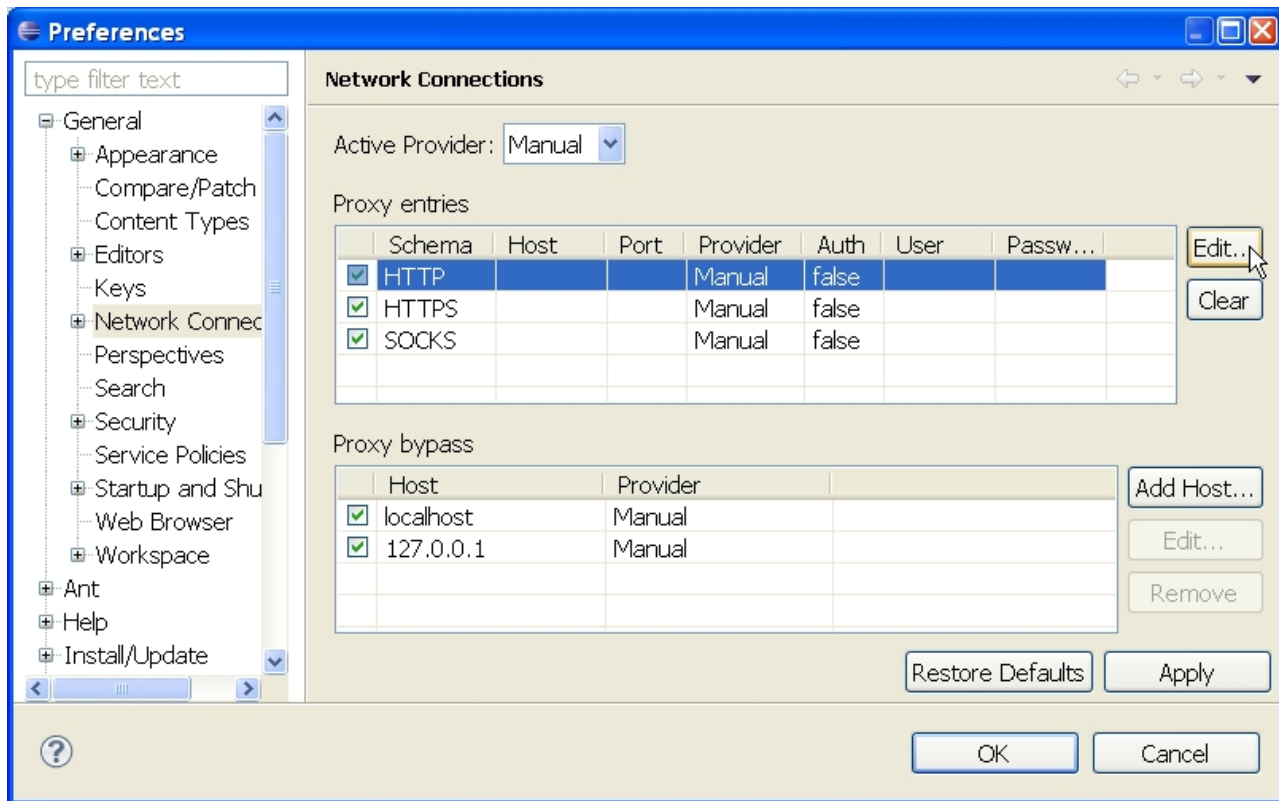
8.4 - Using a Proxy Server

To successfully work with the Samsung Mobile Widget SDK when your development environment uses a proxy server to connect to the Internet, you will need configure Eclipse to recognize your proxy. To do this, follow these steps:

- Go to **Window > Preferences**, open the **General** node, and choose the **Network Connections** node.
- Select **Manual** in the **Active Provider** drop-down list at the top of the **Network Connections** dialog box.

- You will need to specify your proxy server for both the HTTP and HTTPS schemas. Select HTTP and click the Edit button.

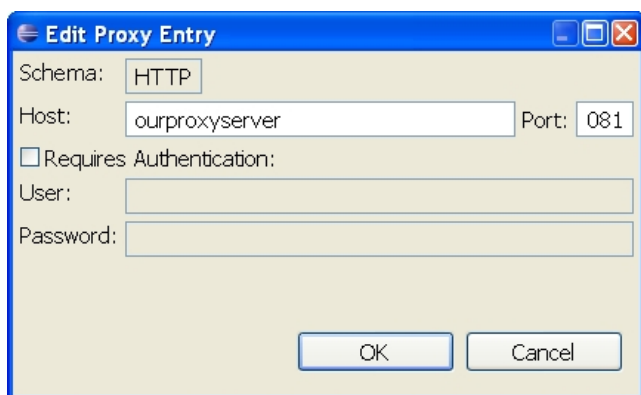
The dialog box should look something like this:



You will specify your proxy server by entering information in this form:

`http://proxy_server_name:port_number`

For example, the Edit Proxy Entry dialog box in the following screen shot has been filled in with a typical host name and port:



The port number is truncated here by the size of the entry field.

When you click the OK button in this dialog box, your new proxy server appears on the Network Connections page for the HTTP schema. Repeat the process for HTTPS.

There are several browser-specific details you should be aware of:

- **Opera:** You must restart the Opera browser for your proxy settings to take effect.
- **NetFront:** The NetFront browser requires that the settings for HTTP and HTTPS be identical. In addition, if the proxy server is local to your machine, the NetFront browser does not work if you specify that your host is `localhost`; in this case, use `127.0.0.1` instead.



All contents Copyright © Samsung Electronics Co., Ltd.

Developing for BONDI

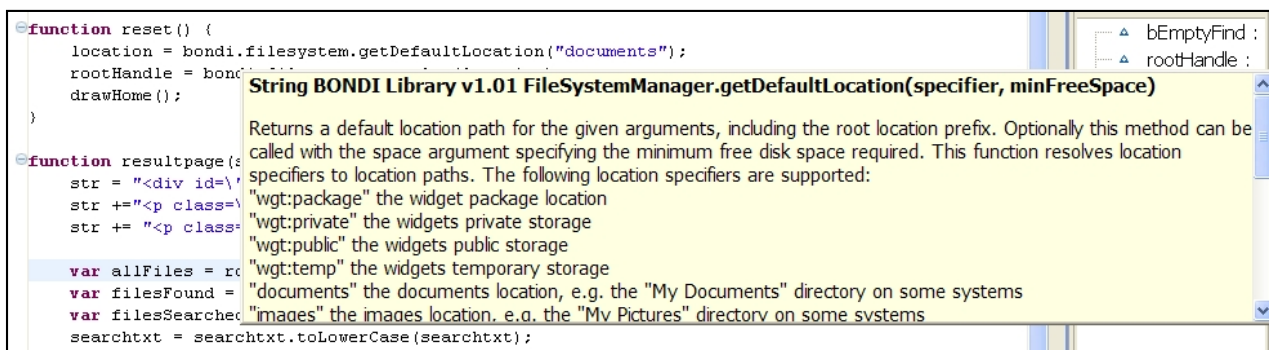
This section of the *Samsung Mobile Widget Development Guide* is for developers who want to take advantage of BOND I functionality in their widgets. BOND I is not widely supported on current devices; in this implementation, it is currently available only on the Dolfin browser on the Wave device. More BOND I-enabled devices are likely to arrive soon.

9.0 - About BOND I

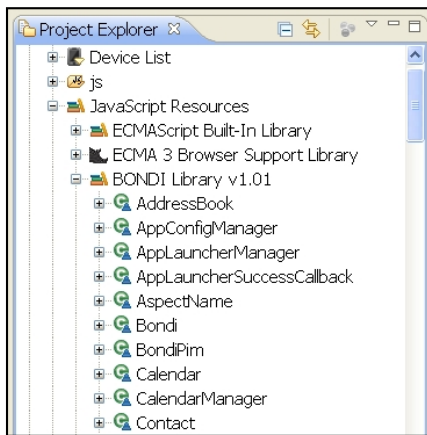
BOND I is a framework for delivering Web-based applications to mass-market devices on multiple platforms. It is a set of JavaScript APIs that grant a Web developer access to device-based capabilities, coupled with a security model that prevents misbehaving applications from damaging or misappropriating core functionality. BOND I is supported by Open Mobile Terminal Platform (OMTP), whose members include 3 Group, AT&T, T-Mobile, Telenor, Telefonica, Telecom Italia, and Vodafone.

The BOND I JavaScript APIs have been incorporated into the Samsung Mobile Widget SDK. Developers can use the APIs in their code and test BOND I widgets in the SDK's emulator.

Detailed documentation of the BOND I API is available as part of the user interface in the Samsung Mobile Widget SDK. To see a description of an API element, hover your mouse cursor over it in the editor. You can give the pop-up documentation the focus by pressing F2; this enables a scrollbar in the pop-up window, as shown below:



To see a list of API, and to see documentation of them in their comments, open the **JavaScript Resources** node of your project in **Project Explorer** and expand the **BOND I Library** node:



You can also find BOND I documentation here: <http://bondi.omtp.org/1.01/apis/>.

9.1 - BOND I Implementation in this SDK

Mobile devices have many capabilities that are usually absent on PCs. A few of these capabilities are the ability to take photographs, navigate using GPS units, detect motion using inertial sensors, and (of course) make and receive telephone calls. A PC cannot realistically emulate many of these capabilities.

Despite these inherent limitations, the Samsung Mobile Widget SDK returns reasonable results for supported modules even when your widget takes advantage of device capabilities that are absent on the PC. For example, a call to `bondi.geolocation.getCurrentPosition` will return meaningful coordinates, even though your PC does not have a GPS receiver. This kind of functionality is implemented by giving you the ability to specify appropriate values in a series of property pages and preferences, as described below, in *BOND I UI Elements in the SDK*.

9.1.1 - BOND I Modules

The current version of the Samsung Widget Runtime implements a subset of the BOND I API. The modules that are currently supported are:

- `bondi.applauncher`
- `bondi.ui`
- `bondi.filesystem`
- `bondi.gallery`
- `bondi.devicestatus`
- `bondi.appconfig`
- `bondi.geolocation`
- `bondi.pim.contact`
- `bondi.pim.calendar`
- `bondi.pim.task`

Currently, the following modules are not supported:

- `bondi.messaging`
- `bondi.camera`
- `bondi.commlog`

9.1.2 - Samsung Widget Runtime 1.0 and BOND I 1.01

The Samsung Widget Runtime version 1.0 and BOND I version 1.01 differ in several ways in their API support. The table in this

section highlights those differences. The emulator in the Samsung Mobile Widget SDK attempts to follow the implementation of the Samsung Widget Runtime, so that the emulated widget performs the same way as it would on the target device.

When a BONDID API element is not supported on the target device, it is also not supported for that device in the Samsung Mobile Widget SDK. For example, the `bondid.ui` module includes several methods for controlling the light on the device. Because these methods are not supported in the BONDID implementation on the current Samsung device, neither `bondid.ui.lightOn` nor `bondid.ui.lightOff` is supported in the SDK.

Except as noted, the following functionality is not supported in the Samsung Widget Runtime version 1.0. Do not use unsupported API elements in your widget.

Application Launcher

```
bondid.applauncher.getDefaultApplication() bondid.applauncher.  
getInstalledApplications/launchApplication
```

- In addition to the standard known names, the Samsung Widget Runtime also supports `file://Camera` and `file://ImageViewer`. The `file://Camera` and `file://ImageViewer` strings are not reported in calls to `getInstalledApplications` in the emulator, but they can be used in `launchApplication`, as they can on the device.

File System

```
bondid.filesystem.registerEventListener()  
bondid.filesystem.unregisterEventListener()  
File.parent  
File.metadata  
FileStream.readBytes()  
FileStream.writeBytes()  
FileStream.readBase64()  
FileStream.writeBase64()  
File.copyTo/moveTo
```

- The `copyTo` and `moveTo` functions are supported, but the ability to overwrite an existing file is not.

Gallery

```
MediaItem.mimeType  
MediaItem.metadata  
bondid.gallery.getGalleries
```

- The SDK emulator returns only one gallery in calls to `getGalleries`. The Samsung Widget Runtime can return *N* galleries, organized by file type.

`gallery.changeView` options:

- The following sort options are *not* currently supported:
 - `MEDIA_SORT_BY_TYPE`
 - `MEDIA_SORT_BY_TITLE`
 - `MEDIA_SORT_BY_AUTHOR`
 - `MEDIA_SORT_BY_ALBUM`
 - `MEDIA_SORT_BY_DATE`
 - `MEDIA_SORT_DESCENDING`
- The following sort options *are* currently supported:
 - `MEDIA_SORT_BY_FILENAME`
 - `MEDIA_SORT_BY_FILEDATE`
 - `MEDIA_SORT_ASCENDING`

- The following view filter attribute is *not* supported for `changeView`:
- `secondarySortOrder`

Geolocation

`PositionOptions.enableHighAccuracy`

User Interface

```
bondi.ui.getSoftKey()
bondi.ui.setOnActivate()
bondi.ui.setOnDeactivate()
bondi.ui.setOnKeyPress()
bondi.ui.lightOn()
bondi.ui.lightOff()
bondi.ui.setOnOrientationChange
MenuItem.setOnSelect
MenuItem.getMenuItemById
MenuItem.appendMenuItem
MenuItem.removeMenuItem
```

Device Status

- Please note that accessing device-status properties is currently limited to Samsung-certified partners. The emulator allows normal access to device status, but calling `getPropertyValue` on a Samsung BONDI-enabled device will result in a permission-denied error.
- Currently only a limited number of aspects and properties is supported in the Device Status module. Please see the *Samsung Widget API Reference 1.0* on the Samsung Mobile Innovator website, <http://innovator.samsungmobile.com>, for a full list.

```
bondi.devicestatus.listVocabularies
bondi.devicestatus.setDefaultVocabulary
bondi.devicestatus.getComponents
bondi.devicestatus.getPropertyValue
bondi.devicestatus.setPropertyValue
bondi.devicestatus.watchPropertyChange
bondi.devicestatus.clearPropertyChange
```

The editor does not include a validator for unsupported APIs. If your widget calls a method from the `bondi.camera` module, for example, no error is generated until you preview the widget in the emulator.

9.2 - BONDI UI Elements in the SDK

You can use tabs in the **Project Editor** to set up the environment for your emulated BONDI device. When you double-click `project.xml` in your BONDI project in the **Project Explorer** window, the **Project Editor** appears:

9.2.1 - Project Editor Tabs

Tabs across the bottom of the Project Editor correspond to the features of a BONDI device that your widget might need to use:

| Tab | Description | BONDI module |
|----------|---|---------------------------------|
| General | Information that applies to every widget created by the Samsung Mobile Widget SDK, including its name, description, version number, and whether it accesses the network. | <i>Not BONDI-specific</i> |
| Features | Specifies the BONDI modules used by the widget. At least one of these modules must be selected when you create a BONDI widget. | <i>All</i> |
| Contacts | Enables you to add contacts to a virtual Contacts List that will be available to your widget in the emulator. | <code>bondi.pim.contact</code> |
| Calendar | Enables you to add calendar events to a virtual Calendar that will be available to your widget in the emulator. | <code>bondi.pim.calendar</code> |
| Tasks | Enables you to add tasks to a virtual Task List that will be available to your widget in the emulator. | <code>bondi.pim.task</code> |
| Device | Settings for such device-specific features as the battery, signal strength, and language. Selecting an Aspect and then an Aspect Property opens a Property Details section in which you can set values for that property. | <code>bondi.devicestatus</code> |
| Location | Settings for Geolocation capabilities, including the starting location, accuracy of the location, and the movement of the device. These settings are described in section 9.2.2 below, <i>Geolocation Services</i> . | <code>bondi.geolocation</code> |
| Source | Source file in which your settings are stored. It is generally safer and easier to make changes to <code>project.xml</code> by using the other tabs in Project Editor. This page is used by all widgets created by the Samsung Mobile Widget SDK, not just BONDI widgets. | <i>All</i> |

All data specified in the Project Editor is initialized in the emulator device when the widget is started. Any changes to the device data made in the widget are not synchronized back into the project data. This one-way flow of data allows you to test widget operations that delete or modify data but still have your original test data the next time you preview your widget.

9.2.2 - Geolocation Services

The capabilities exposed by the `bondi.geolocation` module have settings you can change in Project Editor's **Location** tab:

- **Location** is specified with a latitude and longitude. The altitude is specified in meters.
- **Accuracy** is specified in whatever units are returned by the device.
- **Movement** is specified with a heading (in degrees) and speed (in meters per second). The heading is the direction of travel; zero represents true north, with values increasing clockwise.

The speed and heading are used to simulate movement when calling `bondi.geolocation.watchPosition`. Location update events are sent to the widget based on how quickly it is moving, but are capped at about ten updates per second. The smallest change that will be reported is about five meters.

Typical speeds that you might use in simulation include:

- walking: 1.3 m/s
- city driving: 13.4 m/s
- highway driving: 27 m/s

The Samsung Mobile Widget SDK includes a `bondi.test` module that makes it easier for you to test a widget that uses the BONDIDeolocation capabilities. This module's `geolocationHeadingAndSpeed` function lets you change the heading and direction programmatically:

```
bondi.test.geolocationHeadingAndSpeed(heading, speed);
```

The following sample code tests for the presence of the `bondi.test` module and then uses the `geolocationHeadingAndSpeed` function to change the heading and speed every two seconds:

```
if (bondi.test) {
    setInterval(function() {
        // Heading is random between 0 and 360. Speed is random between 0 and 10.
        bondi.test.geolocationHeadingAndSpeed(Math.floor(Math.random()*360),
            Math.floor(Math.random()*10));
    }, 2000);
}
```

9.2.3 - Preferences Pages

The settings in `project.xml` are easy to modify as you test your widget. Other BONDIDe-specific settings are unlikely to change after you set up your project, however; you can make these settings on preferences pages.

To view and modify the preferences pages, choose **Window > Preferences** and expand the **Samsung** node.

| Preferences Page | Description | BONDIDe module |
|------------------|---|--------------------------------|
| Applications | Specifies applications that the emulator can launch when the widget calls any of the <code>bondi.applauncher</code> API. | <code>bondi.applauncher</code> |
| Directories | Specifies directories corresponding to the locations your widget might specify in a call to the <code>bondi.filesystem.getDefaultLocation</code> or <code>bondi.filesystem.getRootLocation</code> method. | <code>bondi.filesystem</code> |

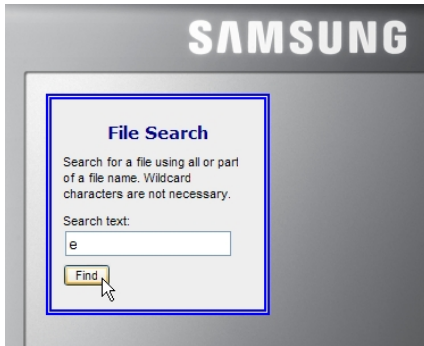
The `bondi.ui` module does not have methods whose properties need to be exposed in the Samsung Mobile Widget SDK. The

SDK supports this module, but you cannot use settings in `project.xml` to change the behavior of such features as a light or a vibrating ringer.

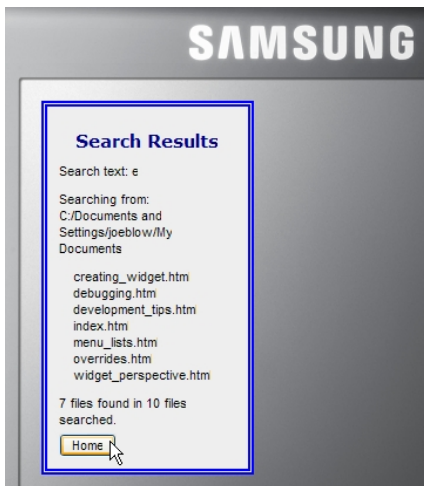
9.3 - Simple BOND! Filesystem Widget

This section of the *Samsung Mobile Widget Development Guide* shows how to implement a simple widget that uses BOND! filesystem API elements. This widget searches file names in the local file system for a given string and reports any matches.

The first page of the widget provides a search entry field:



The second page of the widget displays the search results:



9.3.1 - Setting Up the Project

Begin by creating a new Widget project:

- Choose **File > New > Widget Project**.
- Give your new project a name and description in the **Create a Widget Project** dialog box. Specify a width of 205 and a height of 400. Uncheck the *Access Network* checkbox and press the **Next** button.
- Select the Wave device in the **Select Devices** dialog box. The Wave device is BOND!-enabled.
- Press the **Finish** button. Since this example doesn't use any of the BOND! templates as a starting point, there is no reason to press the **Next** button here.

Now we need to specify which BOND! modules we will be using.

- Double-click `project.xml` in your new project.
- Select the **Features** tab at the bottom of the editor window.
- Select the checkbox labeled `http://bondi.omtp.org/api/filesystem`.
- Save your project to write the change to `project.xml`.

Finally, we need to specify what directory will be searched by this widget. Select **Window > Preferences** and open the **Samsung > Directories** node. Edit the Documents preference so that it specifies a reasonable target directory on your computer. This example uses `C:\Documents and Settings\user\My Documents`.

9.3.2 - Creating index.html

This widget uses JavaScript to write the HTML dynamically, so the `index.html` file is very simple:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title></title>
        <link rel="stylesheet" type="text/css" href="css/style.css" />
        <script type="text/javascript" src="js/main.js"></script>
    </head>
    <body onload="onFeatureLoad()">
        <div id="liner1">
            </div> <!-- end liner1 -->

        <script type="text/JavaScript">
            function onFeatureLoad() {
                reset();          // entry point in main.js
            }
        </script>
    </body>
</html>
```

The HTML is written inside the `liner1 <div>` tag. When the page loads, the `onFeatureLoad` function is invoked. It calls a `reset` function that is defined inside the `main.js` JavaScript file.

9.3.3 - Creating main.js

Change the `main.js` file inside your project's `js` directory so that it looks like this:

```
var bEmptyFind = new Boolean();
var rootHandle;
var locDefault;

function reset() {
    locDefault = bondi.filesystem.getDefaultLocation("documents");
    rootHandle = bondi.filesystem.resolve(locDefault);
    drawHome();
}

function resultpage(searchtxt) {
    str = "<div id=\"boxhead\">Search Results</div>";
    str += "<p>Search text: " + searchtxt + "</p>";
    str += "<p>Searching from: <br />" + locDefault + "</p>" + "<p>";

    var allFiles = rootHandle.listFiles();
    var filesFound = 0;
    var filesSearched = 0;
    searchtxt = searchtxt.toLowerCase(searchtxt);
    for(var i = 0; i < allFiles.length; i++) {
```



```

        var strName = allFiles[i].name;
        strName = strName.toLowerCase(strName);
        if (strName.indexOf(searchtxt, 0) != -1) {
            str += " " + allFiles[i].name + "<br />";
            filesFound++;
        }
        filesSearched++;
    }
    str += "</p><p>" + filesFound + " files found in " + filesSearched;
    str += " files searched.</p>";

    str += "<form><input class=\"bodytype\" type=\"button\" value=\"Home\"";
    str += "onclick=\"drawHome()\" /></form>";
    document.getElementById("liner1").innerHTML = str;
}

function findresponse(){
    var searchtxt = document.forms[0].elements[0].value;
    if (searchtxt == null || searchtxt == "") {
        if (bEmptyFind == false) { // add this warning only on first occurrence
            var newp = document.createElement("p");
            str = "Please enter all or part of a file name.";
            newp.innerHTML = str;
            document.getElementById("liner1").appendChild(newp);
            bEmptyFind = true;
        }
    } else {
        bEmptyFind = false;
        resultpage(searchtxt);
    }
}

function drawHome() {
    str = "<div id=\"boxhead\">File Search</div>";
    str += "<p>Search for a file using all or part of a file name. ";
    str += "Wildcard characters are not necessary.</p>";
    /* The input type cannot be "submit" in the following form */
    str += "<form name=\"searchform\">";
    str += "<p>Search text: <input type=\"text\" name=\"searchtext\" /><br />";
    str += "<input class=\"bodytype\" type=\"button\" value=\"Find\"";
    str += "onclick=\"findresponse()\" /> </form></p>";
    document.getElementById("liner1").innerHTML = str;
}

```

The code in `main.js` implements four functions:

reset

Specifies the local directory where the search is conducted, gets a handle to that location, and calls the `drawHome` function that draws the widget's home page. This function is called from `index.html`. The parameters you can pass to the `bondi.filesystem.getDefaultLocation` function refer to locations you can specify in a preferences page, as described above, in *BONDI UI Elements in the SDK*.

resultpage

Creates the search-result page. This function searches through a list of file names, attempting to find a match to the user-supplied string. The `rootHandle.listFiles` function here is a BONDI method in the `File` interface. The `resultpage` function is called by the `findresponse` function.

findresponse

Retrieves the text entered by the user and, if the string is not empty, calls the `resultpage` function to report the search result. This function is called by the `drawHome` function when the user clicks the **Find** button.

drawHome

Creates the widget's home page. This function is called by the `reset` function on startup. When the user enters text in the *Search Text* entry field and clicks the **Find** button, this function calls the `findresponse` function.

9.3.4 - Creating style.css

You need to make only a few simple style definitions in your project's `style.css` file:

```
body {
    font-family: sans-serif;
    font-size: 70%;
}

p {
    margin: 10px 0 0 0;
    padding: 0;
}

.bodytype {
    font-family: sans-serif;
    font-size: 100%;
    margin: 5px 0 0 0;
}

#liner1 {
    margin: 10px 0 10px 10px;
    padding: 10px 10px 10px 10px;
    border: 5px double blue;
    background-color: #EEEEEE;
    width: 80%;
}

#boxhead {
    font: bold 15px Verdana, sans-serif;
    color: #000088;
    text-align: center;
    margin: 10px 0 10px 0;
    padding: 0;
}
```

9.3.5 - Unimplemented Features

This widget demonstrates several `bondi.filesystem` methods and the overall architecture of a BOND-Enabled widget, but its simplicity prevents it from being very useful. It would be easy to add UI features such as allowing the user to change the search directory and searching subdirectories, but these features are outside the scope of this tutorial. For information about some of the key `bondi.filesystem` methods you would use to add features like this, see <http://bondi.omtp.org/1.01/apis/filesystem.html#File>.

9.4 - Debugging BOND-Enabled Widgets

Most BOND-Enabled widgets can be debugged using the WebKit debugger that is integrated into the Samsung Mobile Widget SDK. WebKit is used to emulate the Dolphin browser. Launching WebKit's "Web Inspector" integrated debugger is described in section 7.1, *Debugging using WebKit*.

In some exceptional circumstances, it may be necessary to run or debug a BOND-Enabled widget in an external browser. To do this, you

will need to install a plugin that provides support for BOND! emulation on your desktop. Follow these steps to download and install this plugin:

1. Download this file: <http://widget.samsungmobile.com/sdk/npBondiRuntime.dll>
2. Copy the downloaded file into the appropriate plugin directory for your browser.
 1. Firefox normally uses: C:\Program Files\Mozilla Firefox\plugins.
 2. Opera will use the Firefox location shown above if Firefox has been installed. Opera might also use <Opera installation directory>\program\plugins or C:\PFiles\Plugins.
 3. Safari will use the Firefox location if Firefox has been installed. Safari might also use <Safari installation directory>\plugins or C:\PFiles\Plugins.



All contents Copyright © Samsung Electronics Co., Ltd.



SDK Menu Items

This section of the *Samsung Mobile Widget Development Guide* is a reference to the menu items that are specific to the Samsung Mobile Widget SDK. Not every menu item is listed here. Menu items are discussed only when they are new or extended for the Samsung Mobile Widget SDK, when their use could be confusing without some explanation, or when they are used so commonly in widget development that omitting them would appear to be an oversight.

For information about the Eclipse user interface, use the following links into the IDE's integrated help system. These links work only when you are viewing this document inside the IDE, by choosing **Help > Help Contents** and expanding the Samsung Mobile Widget Development Guide node. To return to this page, click the yellow left arrow (↩) immediately above this text.

Eclipse menu items:

- File menu
- Edit menu
- Navigate menu
- Project menu
- Window menu
- Help menu

Eclipse buttons and icons:

- Project Explorer icons
- Editor area marker bar
- Tasks view
- Toolbar buttons

You can find a great deal of useful information about Eclipse in the help file, under **Workbench User Guide**. The preceding links take you to pages in **Workbench User Guide > Reference > User interface information**.

10.0 - Sequential Menu List


This section lists menu items in the order in which you will find them in the user interface.

File Menu

- **New > Widget Project** creates a new widget project. For a description of using this menu item, see *Creating a New Project*.
- **Import** imports an existing widget, creating a new project for it. For a description of using this menu item, see *Importing an Existing Widget*.
- **Properties** gives you information about the selected project. You should select a project name before choosing this menu item. Information that may be useful to widget developers includes:
 - **Resource node** identifies the workspace for the project.

- **Run/Debug Settings** enables you to work with launch configurations for your project. For information about launch configurations, see *Previewing on a Browser*.
- **Validation > HTML Syntax** enables you to configure project-specific HTML validation settings and provides a link to a dialog box where you can configure settings for your entire workspace.

Run Menu

- **Run As > Widget** enables you to test your widget in the device emulator. There are several other ways to do this - by double-clicking the device in the **Device List** in **Project Explorer** view, by using the context menu in **Project Explorer**, or by using the **Run** button in the menu bar (). For a description of using this menu item, see *Testing a Widget Project*.
- **Run Configurations** enables you to create selectable launch environments for your widget project; for instance, you could create a configuration that allowed you to choose to run the widget in Internet Explorer. In the **Run Configurations** dialog box, choose **Widget > New_configuration** to reveal the **Widget Preview Mode** settings. You can use the same alternative ways of doing this as you could for **Run As > Widget** above. For information about launch configurations, see *Previewing on a Browser*.

Widget Menu

- **Add Device** adds a device to the **Device List**. For a description of using this menu item, see section 4.2 - *Selecting Devices*.
- **Package** creates a .WGT file. For a description of using this menu item, see section 4.6 - *Packaging a Widget*.
- **Application Store** causes your default browser to open a window to the Samsung Application Store, where you can upload your new widget. For a description of using this menu item, see section 4.6 - *Packaging a Widget*.
- **Update Project** ensures that the current XML schema for your project is up to date. For a description of using this menu item, see *Version 1.1 update* in section 1.2 - *Release Notes*.

Window Menu

- **Open Perspective > Other** reveals a dialog box that allows you to choose the **Widget** perspective. You can also use the **Open Perspective** icon to do this. For a description, see *Launching the Samsung Mobile Widget SDK*.
- **Show View** enables you to show any views that may be hidden and to choose any views that are not part of the default **Widget** perspective. If you select **Show View > Other** you will be able to choose from many different views that are supplied by the Eclipse IDE.
- **Save Perspective As** enables you to save a customized version of the **Widget** perspective. For more information, see *Customizing a Perspective*.
- **Reset Perspective** resets the **Widget** perspective to its default settings.
- **Preferences** reveals a dialog box where you can change a wide array of different settings for Eclipse.
 - **Install/Update > Available Software Sites** lists the URLs of web sites that might contain Eclipse updates. One of these sites is the Samsung widget update site. You can use this page to modify or test your connection to this site.
 - **Web** reveals a large number of preferences that are interesting to widget developers, including **CSS Files**, **HTML Files**, and **JavaScript**.


Help Menu

- **Help Contents** opens the Eclipse integrated help system. One of the items in the **Contents** pane is *Samsung Mobile Widget Development Guide* - this documentation. You may sometimes find it useful to restrict the scope of your searches to this document; to do this, click the *Search scope* link to the right of the **Search** entry field and define a search scope that is restricted to this guide.

- **Check for Updates** causes Eclipse to check a list of web sites for updates to any of its plug-ins. Any updates to the Samsung Mobile Widget SDK will be downloaded and installed with you choose this menu item.
- **Install New Software** enables you to specify a site from which you would like to download new Eclipse plug-ins. For more information, see *Installing the Samsung Mobile Widget SDK*.
- **About Eclipse** reveals a dialog box with Build identifiers and buttons that display features of individual Eclipse components. One of these buttons is decorated with the widget icon; this will take you to information about the Samsung Mobile Widget SDK.

Clicking the **Installation Details** button at the bottom of the **About Eclipse** dialog box brings up an **Eclipse Installation Details** dialog box, which contains a wealth of information about the plug-ins that comprise the IDE.


Project Explorer View Menu

You gain access to a view menu by clicking the small inverted triangle in the view's icon bar ()

- **Customize View** enables you to specify items you want to hide or reveal in the **Project Explorer** view. For more information, see *Customizing a Perspective*.
- **Link with Editor** is a persistent setting that causes the name of the file you are working on in the **Source Editor** view to be highlighted in **Project Explorer**.

Project Explorer Context Menu

You gain access to a view's context menu by right-clicking in the view. The context menu changes depending on what item is selected when you click the right mouse button. Some of the following menu items appear only when a file of a particular type is selected.

- **New > Project** opens a **New Project** dialog box, which you can use to create a new widget project. Inside this dialog box, you can expand the Samsung node, select **Widget Project**, click the **Next** button, and complete the creation process as described in *Creating a New Project*.
- **Add Device** adds a device to the **Device List**. For a description of using this menu item, see section 4.2 - *Selecting Devices*.
- **Package** creates a **.WGT** file. For a description of using this menu item, see section 4.6 - *Packaging a Widget*.
- **Application Store** causes your default browser to open a window to the Samsung Application Store, where you can upload your new widget. For a description of using this menu item, see section 4.6 - *Packaging a Widget*.
- **Import** opens an **Import** dialog box. For information about importing an existing widget into the IDE, see *Importing an Existing Widget*.
- **Run As > Widget** enables you to test your widget in the device emulator. There are several other ways to do this - by double-clicking the device in the **Device List** in **Project Explorer** view, by using the context menu in **Project Explorer**, or by using the **Run** button in the menu bar ()
- **Properties** gives you information about the selected project. For more about this menu item, see *File Menu*, above.
- **Open With > Web Page Editor** - *available only when you have selected an HTML file*. This source editor may help visualize your work as you produce a widget. It includes a visual preview, a palette of user-interface elements that you can drag into your project, and Design and Preview modes. For more about this menu item, see *Using Open With Web Page Editor*.

10.1 - Menu Index

This is an alphabetical index of the menu items that are described in the preceding section.

- About Eclipse See *Help Menu*.
- Add Device See *Selecting Devices*.
- Application Store See *Packaging a Widget*.
- Available Software Sites, Preferences > Install/Update See *Window Menu, Preferences*.
- Check for Updates See *Help Menu*.
- Configurations, Run See *Run Menu, Run Configurations*.
- Customize View See *Project Explorer View Menu*.
- Eclipse, About See *Help Menu, About Eclipse*.
- Eclipse, Installation Details See *Help Menu, About Eclipse*.
- Editor, Link with See *Project Explorer View Menu, Link with Editor*.
- Help Contents See *Help Menu*.
- HTML Syntax, Validation See *File Menu, Properties*.
- Import See *File Menu* and *Project Explorer Context Menu*.
- Install/Update > Available Software Sites, Preferences See *Window Menu, Preferences*.
- Install New Software See *Help Menu*.
- Link with Editor See *Project Explorer View Menu*.
- New > Project See *Project Explorer Context Menu*.
- New > Widget Project See *File Menu*.
- Open Perspective > Other See *Window Menu*.
- Open With > Web Page Editor See *Project Explorer Context Menu*.
- Package See *Packaging a Widget*.
- Perspective, Open > Other See *Window Menu*.
- Perspective, Reset See *Window Menu, Reset Perspective*.
- Perspective, Save As See *Window Menu, Save Perspective As*.
- Preferences See *Window Menu*.
- Project, New See *Project Explorer Context Menu, New Project*.
- Project, New Widget See *File Menu*.
- Properties See *File Menu* and *Project Explorer Context Menu*.
- Reset Perspective See *Window Menu*.
- Run/Debug Settings See *File Menu, Properties*.
- Run As > Widget See *Run Menu* and *Project Explorer Context Menu*.
- Run Configurations See *Run Menu*.
- Save Perspective As See *Window Menu*.
- Show View See *Window Menu*.
- Software, Install New See *Help Menu, Install New Software*.
- Update Project See *Version 1.1 Update* in *Release Notes*.
- Updates, Check for See *Help Menu, Check for Updates*.
- Validation See *File Menu, Properties*.
- View, Customize See *Project Explorer View Menu, Customize View*.
- View, Show See *Window Menu, Show View*.
- Web, Preferences See *Window Menu, Preferences*.
- Widget, Run As See *Run Menu, Run As* and *Project Explorer Context Menu*.

- Widget Project, New See *File Menu*.



All contents Copyright © Samsung Electronics Co., Ltd.