IBM® Lotus® Symphony™ Developer's Guide

IBM® Lotus® Symphony™ Developer's Guide

Note
Before using this information and the product it supports, read the information in "Part 8. Appendixes" on page 139.

Lotus Symphony V1.0.0 Edition (May 2008)

This edition applies to release V1.0.0 of IBM Lotus Symphony toolkit (license number L-AENR-7DSDUB) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Part 1. About This Publication

Chapter 1. Intended Audience

This guide is intended for Java™ developers who have read the IBM® Lotus® Symphony programming introduction in the Lotus Symphony forum and who need a more in-depth understanding of the Lotus Symphony toolkit to create their own applications. This developer's guide is written to provide quick and easy reference to the different components of the toolkit. For information about Lotus Symphony programming, go to the Website at: http://symphony.lotus.com.

This guide does not include information about general Java programming. For more information on the Java language and Java programming, go to the Website at: http://www.java.sun.com. This guide also does not cover the details of Lotus Symphony API (application programming interface) that are covered in the Javadoc within the toolkit.

Chapter 2. Requirements

The Lotus Symphony Toolkit can be used in the Eclipse 3.2.2 development environment on Microsoft® Windows® XP or Red Hat Enterprise 5.

To build plug-ins, you must have Lotus Symphony installed. Plug-ins created from this toolkit can be deployed into Lotus Symphony on all platform supported by Lotus Symphony.

For detailed information about software requirements for the Lotus Symphony toolkit, see the `readme.txt` file that is included with the toolkit.

Chapter 3. How to Use this Guide

This document is composed of several major parts: product overview, designing Symphony applications, extending IBM Lotus Symphony, IBM Lotus Expeditor and UNO programming, example plug-ins, and troubleshooting and support.

Part 1: Chapter 4 : introduces the main composing of Symphony developer's toolkit and how to begin your Symphony development journey by this toolkit.

Part 2: Product overview

1. Chapter 1: introduces what Lotus Symphony is.
2. Chapter 2: introduces Lotus Symphony architecture outline and the components based.
3. Chapter 3: introduces the programming model on custom Lotus Symphony development.

Part 3: Designing Symphony applications

1. Chapter 1: introduces the Rich Client application.
2. Chapter 2: introduces Lotus Symphony backend service.
3. Chapter 3: introduces two ways to build office applications.
4. Chapter 4: the Lotus Expeditor toolkit for Lotus Symphony application developers.

5. Chapter 5: Lotus Symphony application's package and deploy.
6. Chapter 6: globalization support in Lotus Symphony.
7. Chapter 7: multi-platform of Lotus Symphony application.
8. Chapter 8: developing applications for Lotus Symphony and Lotus Symphony in Lotus Notes.

Part 4: Extending Lotus Symphony

1. Chapter 1: describes step by step how to set custom Lotus Symphony development environment.
2. Chapter 2: introduces how to customizing user Lotus Symphony interface. Such as custom menu, toolbar, launcher item, side shelf, auto recognizer, status bar and preference.
3. Chapter 3: introduces how to use the Lotus Symphony java APIs and extensions in Lotus Symphony toolkit.
4. Chapter 4: introduces how to use UNO APIs to operate three kinds of document model after get it from Lotus Symphony APIs.
5. Chapter 5: describes step by step how to deploy a custom Lotus Symphony application and manage it.

Part 5: Expeditor and UNO programming

1. Chapter 1: introduces developing application on Expeditor platform.
2. Chapter 2: introduces how to use UNO's function in Lotus Symphony development. Such as get global service factory, use import/export function, export document to HTML file and JEPG image.

Part 6: Example plug-ins

1. Chapter 1: describes step by step how to create a hello world plug-in on Lotus Symphony. This sample adds a side shelf to say hello.
2. Chapter 2 demonstrates how to create a simple editor in a view on Lotus Symphony. This sample creates a sample editor on a view.
3. Chapter 3: demonstrates how to operate a spreadsheet on a Lotus Symphony side shelf. This sample shows how to open a spreadsheet by opening two demo files, set and get a cell's value and its address dynamically, how to create a chart and a data pilot.
4. Chapter 4: demonstrate how to manipulate writer document programmatically on a Lotus Symphony side shelf. This sample creates a side shelf for operating a writer document, such as creating sections, creating tables, creating user defined fields.
5. Chapter 5: shows a typical sample application on Lotus Expeditor platform which Lotus Symphony development based. This sample creates a custom perspective and adds three views, an early startup when Lotus Symphony was startup, a status bar and a custom help.
6. Chapter 6: shows how to load document implicitly and export to HTML and JEPG by document type. This sample shows a button for loading documents implicitly; a button for exporting and converting the loading document into HTML file or JEPG image according its type, ODT and ODS into HTML file, ODP into JEPG image array; a simple setter and getter operating to show accessing metadata.

Part 7: Troubleshooting and support

1. Chapter 1 problem and solution about development environment setting up.

2. Chapter 2: problem and solution about Lotus Symphony hang when executing UNO call in Java code.
3. Chapter 3: problem and solution about application does not work when plug-ins deployed
4. Chapter 4: how to get support on Lotus Symphony form.

Part 8: Appendixes

The Appendixes of this developer's guide.

Chapter 4. The Lotus Symphony Toolkit

To access the toolkit, see http://symphony.lotus.com. The Lotus Symphony download page contains links to all the documentation and downloads. You can extract the files for this toolkit on your local system.

4.1 Get started with the toolkit

If you want to get a quick development experience using the Lotus Symphony toolkit, the best way is create a "Hello world" plug-in. To create this plugin, do the following steps:

1. Set up the development environment. Refer to Part 4 Chapter 1: Setting up the integrated development environment in this guide, to set up your development environment.
2. Create a "Hello world" plug-in. Refer to part 6 chapter 1 **Hello world sample plug-in** in this guide.

If you want to experience more, the next best choice is tutorial plug-in sample **DocumentWorkflow** and the tutorial document in the Lotus Symphony toolkit.

4.2 Sample plug-ins

These plug-in samples show how to develop custom plug-ins and applications, how to use the Lotus Symphony APIs and others support functions to add custom UI (user interface elements) and create Lotus Symphony documents. The list of sample plug-ins is as follows:

- `com.ibm.productivity.tools.samples.helloworld`
- `com.ibm.productivity.tools.samples.DocumentWorkflow`
- `com.ibm.productivity.tools.samples.views`
- `com.ibm.productivity.tools.samples.spreadsheet`
- `com.ibm.productivity.tools.samples.writer`
- `com.ibm.productivity.tools.samples.customizing`
- `com.ibm.productivity.tools.samples.convertor`

The examples found in this guide can be run directly from the Lotus Symphony development environment. For instructions on accessing and running the samples, refer to Part 6 Example plug-ins in this guide.

4.3 Related documentation

- Lotus Symphony Java Toolkit Javadoc Reference
- Lotus Symphony Java Toolkit Tutorial
- Lotus Symphony Java Toolkit `readme.txt`

Part 2. Product Overview

Chapter 1. Introduction to Lotus Symphony

Lotus Symphony is a set of applications for creating, editing, and sharing word processing documents, spreadsheets, and presentations. Designed to handle the majority of office tasks, the Lotus Symphony tools support the Open Document Format (ODF), enabling organizations to access, use, and maintain their documents over the long term without worrying about end-of-life uncertainties or ongoing software licensing and royalty fees. By using tools that support ODF, customers are not locked into one particular vendor for their productivity tools. ODF helps provide interoperability and flexibility.

With Lotus Symphony, users create, manage, edit, and import documents in ODF. However, Lotus Symphony tools can also import, edit, and save documents in Microsoft® Office formats or export those documents to ODF for sharing with ODF-compliant applications and solutions.

Lotus Symphony offers more than a simple office application suite. Because it leverages the Eclipse-based product IBM Lotus Expeditor and OpenOffice.org technology, a variety of plug-ins that expand the functionality of Lotus Symphony are available from the Lotus Symphony Web site, and third parties can build additional plug-ins to extend Lotus Symphony.

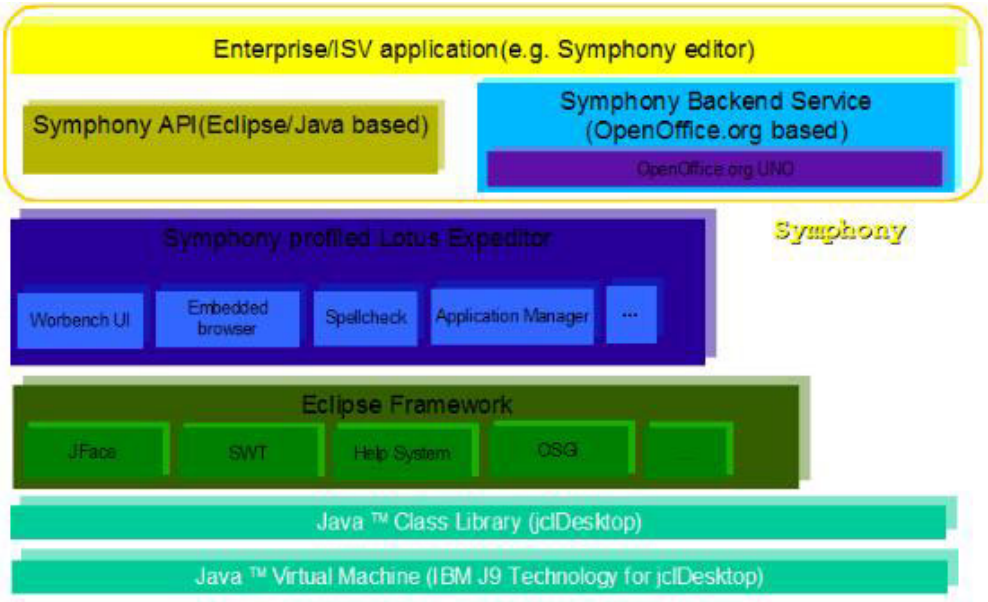
Chapter 2. Lotus Symphony Architecture

Lotus Symphony is derived from OpenOffice.org and it is built on the Eclipse plug-in framework and the Lotus Expeditor rich client platform. In essence, Lotus Symphony is a package of Eclipse plug-ins.

2.1 Overview of Lotus Symphony Architecture

Lotus Symphony wraps the OpenOffice.org application as Eclipse components to provide office document applications.

This picture shows a high-level outline of the Eclipse architecture as Lotus Symphony uses it.



Eclipse is a general-purpose and open-source framework on which you can develop applications. A *plug-in* is the smallest unit of Eclipse Platform function that can be developed and delivered separately. Statically, Lotus Symphony is a set of Eclipse plug-ins that re-packages OpenOffice.org; in runtime, Lotus Symphony re-parent OpenOffice.org window into an Eclipse SWT(Standard Widget Toolkit) control.

You can extend Lotus Symphony by creating plug-ins that extend the Lotus Symphony plug-ins. Your plug-in can access any of the services that are exposed by Lotus Symphony or its underlying platforms, for example, the Lotus Expeditor platform or the Eclipse platform.

2.2 Overview of Eclipse

Eclipse is an integrated development environment. Eclipse offers the *Rich Client Platform (RCP)*, which is required if you want to use the Eclipse graphic toolkit to build stand-alone applications. For more information about Eclipse and RCP, refer to the following resources:

http://www.eclipse.org

http://wiki.eclipse.org/index.php/RCP_FAQ

The following table lists and describes some of the Eclipse platform components that Lotus Symphony uses.

Component	Description
Platform Runtime	Provides the foundational support for plug-ins and for the plug-in registry, a mechanism for declaring extension points, and for extending objects dynamically. The Eclipse runtime uses the standard OSGi framework to define how plug-ins are packaged.
Help	Provides a plug-in with HTML-based online help and search capabilities. Help content is added via user's plug-ins that are recognized at runtime..

Component	Description
JFace	Provides the user interface (UI) framework, working in conjunction with the Standard Widget Toolkit (SWT), for handling many common UI programming tasks.
SWT	Provides access to the UI facilities of the operating systems on which it is implemented. SWT-built applications leverage the UI of the host system more than do other Java toolkits, such as Swing.
Preferences	An Eclipse-managed collection of indexed windows dialog boxes. Plug-ins can add new Preferences pages using an extension.
Workbench	Provides a highly scalable, open-ended, and multi-window environment for managing views, editors, perspectives (task-oriented layouts), actions, wizards, preference pages, and more.
OSGi	Provides Eclipse with the value of OSGi, which includes life cycle management. Lotus Symphony is based on Eclipse 3.2, which is based on OSGi R4.

2.3 Overview of Lotus Expeditor

IBM Lotus Expeditor is a server-managed client solution that extends back-end server services to new users who use a range of client devices spanning desktops, laptops, mobile devices, and specialized devices.

There are several Lotus Expeditor solutions, including Lotus Expeditor for Desktop, Lotus Expeditor for Devices, Lotus Expeditor Toolkit, and Lotus Expeditor Server. The combination of the Lotus Expeditor clients and the Lotus Expeditor server provide the end-to-end services necessary to deliver and manage applications. Lotus Expeditor Toolkit provides a complete, integrated set of tools that allow you to develop, debug, test, package, and deploy client applications. Lotus Symphony is based on Lotus Expeditor for Desktop. In the remaining parts of this document, when Lotus Expeditor is mentioned, it is intended to mean Lotus Expeditor for Desktop.

Lotus Expeditor is an integrated client platform for desktops and laptops that extends the J2EE programming model to clients. The client provides a rich client platform that can operate disconnected from the enterprise such that enterprise applications can operate when the client is online or offline.

The following table lists some of the Expeditor services that Lotus Symphony uses.

Service	Description
Application manager	Enables users to directly install applications and components from standard Eclipse update sites onto managed clients.
Embedded browser	Provides a configurable embedded Web browser.
Spell check	Is used to check misspelled words in document. It is based on the text analyze framework.
Personalities	Defines the framework that the platform uses to determine what perspectives or windows, menus, actions, action bar items, and status line controls are displayed when the application starts.
Application launcher	Is represented in the user interface as a button with a drop-down menu that contains the list of applications available to the user.
Eclipse UI extensions	Common UI extensions provided by the Eclipse platform.

2.3.1 J9 JCL Desktop

Lotus Symphony for Microsoft Windows® and Linux® operating systems uses a compacted, custom Java Runtime Environment known as the J9 Java Class Libraries (JCL) Desktop. While this pared down J9 Java Runtime Environment enables a smaller footprint for the Lotus Symphony client, the J9 does not contain the full number of Java classes included in the standard 1.4.2 or 1.5 Sun JVM. For example, some of the classes not contained in the J9 JVM are AWT and Swing classes, which are used for graphical user interface (GUI) objects in Java applications. These packages are not part of the J9 JVM.

Accordingly, developers might encounter issues when creating plug-ins that reference a class or package (such as AWT or Swing) that is not included in the J9 VM. For this, refer to part 3 chapter 1 in this guide.

2.3.2 The profile of Lotus Expeditor used by Lotus Symphony

Lotus Symphony uses a minimal profile of the Lotus Expeditor platform. The following picture describes the profiled platform. Many components are removed from the Lotus Expeditor platform, such as Web Application Perspective, Portlet Viewer, WSRP, and SSO. The Lotus Symphony profiled lotus Expeditor platform maintains a minimal set of components required by the rich client application model.

2.4 OpenOffice.org

OpenOffice.org is the open source project through which Sun Microsystems has released the technology for the StarOffice Productivity Suite. All of the source code is available under the GNU Lesser General Public License (LGPL).

OpenOffice.org is based on Universal Network Objects (UNO) technology and is the base component technology for OpenOffice.org. You can use and write components that interact across languages, component technologies, computer platforms, and networks. In Lotus Symphony, UNO is available on Linux, and Windows for Java, C++ and OpenOffice.org Basic. UNO is available through the component technology Microsoft COM for many other languages. UNO is used to access Lotus Symphony back-end services, using its application programming interface (API). The OpenOffice.org API is the comprehensive specification that describes the programmable features of OpenOffice.org.

Chapter 3. Lotus Symphony Programming Model

Lotus Symphony is the combination of Eclipse-based Lotus Expeditor and OpenOffice.org. Both of these products provide rich APIs for application integration. In Lotus Symphony, the OpenOffice.org window is re-parented to a SWT control in Eclipse. Most of the user interface items that you can add are provided through Eclipse extension points, such as the menu, toolbar, status bar, and preference page. With this approach, Lotus Symphony provides flexibility for user interface integration with other Eclipse and Lotus Expeditor-based applications.

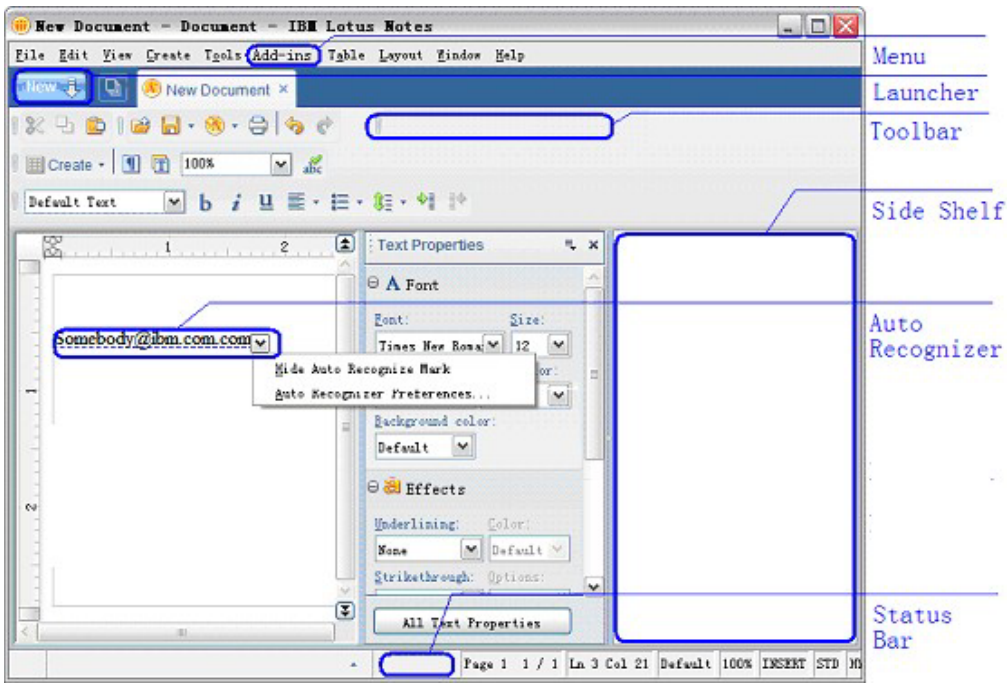
The programming model of Lotus Symphony can be described as:

- User interface integration is based on Eclipse and Lotus Expeditor extension points and plug-in framework.

- The document content level API is based on OpenOffice.org UNO capability.
- The Lotus Symphony API focuses on the integration between OpenOffice.org and Eclipse and Lotus Expeditor.
- The add-in mechanism is based on Lotus Expeditor Application Manager.

In this way, Lotus Symphony inherits the user interface flexibility of Eclipse and Lotus Expeditor and the rich functionality of UNO APIs.

The following screen capture shows the user interface items.



Part 3. Designing Lotus Symphony Applications

This part provides information about planning and designing issues before you can develop Lotus Symphony applications. It describes the recommended approach using the design perspective in the following chapters. For more details about how to develop Lotus Symphony application, refer to Part 4.

Chapter 1. Rich Client Applications

If you want to build a graphical user interface application, the rich client programming model is recommended. The pattern is supported through the rich client application model from Lotus Expeditor. Using Eclipse and Lotus Expeditor, an application can be an aggregation of display components, including menus, toolbars, views, status bars, and side shelves.

You can extend the Lotus Symphony editor by building plug-ins. Most of the user interface components can be added through extension points. For details about how to use the extension points, refer to Part 4.

If you want to access the document model of a loaded document, use the UNO API. There is typical usage provided in Part 5 Chapter 2. You can also find samples in the Lotus Symphony toolkit.

Chapter 2. Lotus Symphony Back-end Service

If you want to build an application without a graphical user interface, you can use the UNO API directly. For example, converting file formats between ODF, PDF, HTML, or MS office format, manipulating documents invisibly, or printing document from file storage without user interaction.

UNO provides language bindings, including Java, C/C++, OLE automation and OpenOffice.org basic. You can also regard the Lotus Symphony editor as a client of the Lotus Symphony back-end service. Lotus Symphony incorporates the display window of OpenOffice.org into a SWT control in the Eclipse environment, so that the user interface of Lotus Symphony is re-designed and re-organized completely. It is also possible for you to re-use the Lotus Symphony back-end service.

The major drawback of UNO is complexity. There is documentation on the Web; you can find the OpenOffice.org software development kit and OpenOffice forum for knowledge and support. The learning curve is still considerable.Use the public APIs provided by Lotus Symphony first. You can get suggestions and help from the Lotus Symphony forum about how to continue if the public APIs are not enough.

Chapter 3. Business Logic

When you want to build an office application, you must decide how to distribute and manage the business logic. You can have two choices here:

- Creating a template which contains the business logic represented by script code
- Creating a separated Eclipse plug-in which contains the business logic

With the first approach, it is easy to create light-weight solutions. You can use OpenOffice.org Basic in Lotus Symphony documents, which is dependent on UNO technology. However, it is hard to manage or extend the scope of business logic.

For enterprise solutions, use the second approach. An Eclipse plug-in is easy to deploy or upgrade in Lotus Symphony. It is also easy to extend the functionality of business logic, for example, accessing data from server. One of the most important concepts of Lotus Expeditor is that you can create a managed client application. It is also applied to your business logic.

Chapter 4. Lotus Expeditor Toolkit for Lotus Symphony Developers

Lotus Expeditor toolkit is the starting point for Lotus Symphony developers and it provides a complete, integrated set of tools that allows you to develop, debug, test, package, and deploy client applications.

There are several programming models defined by the Lotus Expeditor toolkit. For example, the Web application model, the rich client application model, the portal application model and the composite application model. From a developer's perspective, only the rich client application model is provided in Lotus Symphony. For more information, refer to Lotus Expeditor documentation.

In the following sections, are typical issues related to using the Lotus Expeditor toolkit from a design perspective.

4.1. Update from jclDesktop to J2SE

The default Java Runtime Environment (JRE) of Lotus Expeditor is IBM J9 VM with the jclDesktop class libraries, an IBM-optimized subset of Java 5 that offers a smaller footprint and faster class loading than standard JREs. It is also the default virtual machine used by Lotus Expeditor Client for Desktop.

If you need more function, such as Swing, or AWT programming libraries that are provided by the J2SE 5.0 virtual machine, it is possible to upgrade the default VM used by the Lotus Symphony runtime. You can upgrade the VM to J2SE according to the following guide:

http://publib.boulder.ibm.com/infocenter/ledoc/v6r11/index.jsp?topic=/com.ibm.rcp.tools.doc.admin/JVMfeatures.html

or from the Lotus Expeditor's local help content on Eclipse after you finished setting up the Lotus Symphony development environment (refer to Part 4 Chapter 1) by following:

Start up Eclipse > **Help > Help Contents > Assembling and Deploying Lotus Expeditor Applications > Installing and launching the Lotus Expeditor Client >Changing the virtual machine.**

Chapter 5. Packaging and Deployment

Although both UNO and Lotus Expeditor provide packaging and deployment options, the primary approach to package and deploy third-party components is based on the update management functionality of Lotus Expeditor.

5.1 Design and develop components with Lotus Expeditor toolkit

A Lotus Expeditor or Lotus Symphony component contains codes for certain functionality. Additional components can be constructed in a specific structure.

A component can be represented as a plug-in or a bundle. A plug-in is a JAR file with a plug-in manifest file named `plugin.xml`. The plug-in manifest describes the plug-in to the framework and enables a plug-in to consume and provide extensions from and to other plug-ins. A bundle is a JAR file with a bundle manifest file named `MANIFEST.MF`. The bundle manifest describes the bundle to the service framework and enables a bundle to consume and provide packages and services from/to other bundles.

If a component can't provide a complete implementation, fragments can be used to complete or extend a component. For example, to support globalization, the primary component can provide an implementation that contains translatable text in a default language. Fragments can also be used to provide translations for additional languages.

5.2 Group components into features with the Lotus Expeditor toolkit

Lotus Symphony can be regarded as a set of plug-ins and fragments on disk. Components are grouped together into features. A feature is the smallest unit of separately downloadable and installable functionality. A feature is used to organize the structure of the entire product. It contains important information for the Update Manager to identify the dependency between features, and the version of features.

For more details about how to create features step-by-step, refer to Part 4 Chapter 5.

5.3 Package the features into the update site with the Lotus Expeditor toolkit

To make the plug-ins deployable, you are also required to generate an update site. An update site is a set of features with a site.xml. The site.xml file defines root features in the update site. An update site is the smallest unit that can be recognized by the Update Manager.

For more details about how to create an update site, refer to Part 4 Chapter 5.

5.4 Distribute the update site

You can copy the update site into each client for deployment or you can put the update site on a server, and provide the server URL for client deployment.

5.5 Deploy the update site into Lotus Symphony

Deploy update site manually via the user interface from Lotus Symphony.

For more information about how to deploy the update site, refer to Part 4 Chapter 5.

Chapter 6. Globalization

Globalization support in Lotus Symphony is based on International Components for Unicode (ICU) technology provided in Lotus Expeditor platform. ICU4J is a set of Java classes that extend the capabilities provided by the J2SE class libraries in the areas of Unicode and internationalization support. The ICU4J classes enable you to:

- Support multiple locales
- Support bidirectional text layouts
- Create translatable plug-ins

Chapter 7. Cross Platform Considerations

In the development phase, use Windows XP or Red Hat Enterprise 5 as the primary development environment. The component developed can be deployed into all platforms supported by Lotus Symphony. The Java APIs provided by Lotus Symphony or Lotus Expeditor are platform independent. UNO APIs are also designed for cross platform applications. Some functions can be platform dependent, for example, OLE Automation bridge of UNO is only available on Windows operating system.

Chapter 8. Developing Applications for Lotus Symphony and for Lotus Symphony in Lotus Notes

Lotus Symphony is available as a standalone editor product, it is also provided in Lotus Notes client version 8.0. Either it is or it is not. The same code base is used in the two products. You can design applications that work for both products. There are still some issues that you should be aware of in the design phase:

- Lotus Symphony is based on a profiled Lotus Expeditor, which is small and fast, while Notes is based on a different set of functionality of Lotus Expeditor.
- The release cycle for Lotus Symphony and Lotus Notes is different. There might be slight differences, in each release of Notes; it will use some levels of Lotus Symphony code.
- Some functionality is only available in Notes. For example, support of LotusScript® and the composite application editor.

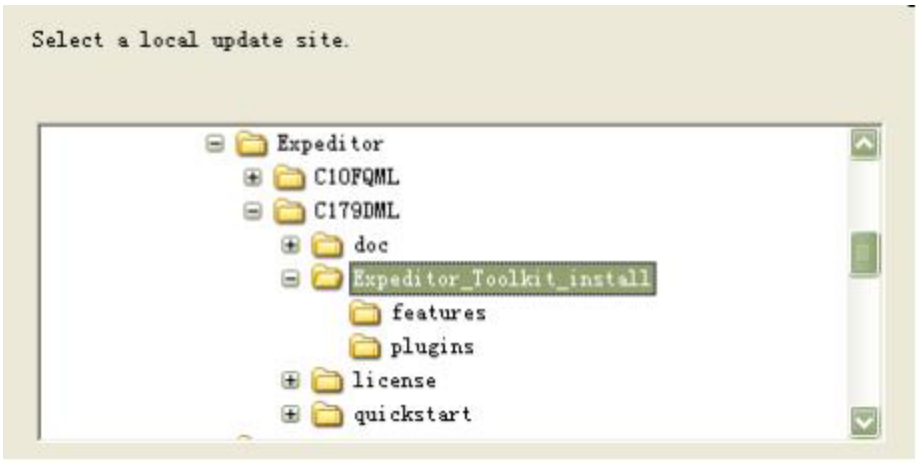
Part 4. Extending Lotus Symphony

Chapter 1. Setting Up the Integrated Development Environment

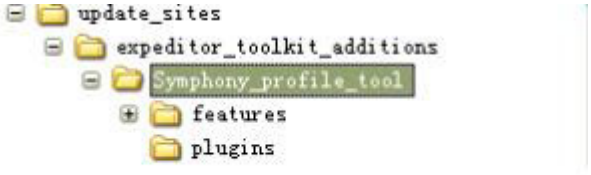
The integrated development environment (IDE) is based on Eclipse 3.2.2 and Lotus Symphony. All the steps in this procedure are for a Windows operating system, but the process on the Linux operating system is similar. If you have any questions during the set up process, refer to Part 7 Troubleshooting and support or get help from the Lotus Symphony forum: http://symphony.lotus.com/software/lotus/symphony/home.jspa.

1. Install Lotus Symphony and Eclipse.
   a. Download Eclipse 3.2.2 SDK for Windows http://archive.eclipse.org/eclipse/downloads/drops/R-3.2.2-200702121330/eclipse-SDK-3.2.2-win32.zip or Linux http://archive.eclipse.org/eclipse/downloads/drops/R-3.2.2-200702121330/eclipse-SDK-3.2.2-linux-gtk.tar.gz and Lotus Symphony from the Lotus Symphony Web site: http://symphony.lotus.com/software/lotus/symphony/home.jspa.
   b. Unpack Eclipse 3.2.2 to a local disk, for example, `D:\eclipse`.
   c. Install Lotus Symphony to a local disk, for example, `D:\IBM\Lotus\Symphony` as `<Symphony installation home>`.
2. Install Lotus Expeditor toolkit with Symphony Support.
   a. Download Expeditor toolkit zip file from the Web site at: http://www14.software.ibm.com/webapp/download/nochargesearch.jsp?q=Lotus+Expeditor+Toolkit and extract it to a local disk.

      Note: recommend 6.1.2 edition
   b. Unzip `Symphony_profile_tool.zip` file which supports Lotus Symphony development on Lotus Expeditor from Lotus Symphony toolkit's `update_sites\expeditor_toolkit_additions` folder to a local disk.
   c. Start the Eclipse IDE.
   d. From the main menu, click **Help > Software Updates > Find and Install**. The Install/Update wizard is displayed.
   e. Select **Search for new features to install** and click **Next**.
   f. In the **Update sites to visit** window, select the **New Local Site** button.
   g. From the **select a local update site** window, select the file that was extracted in **step a**, expand it and select **Expeditor_Toolkit_install**, and then click **OK**.

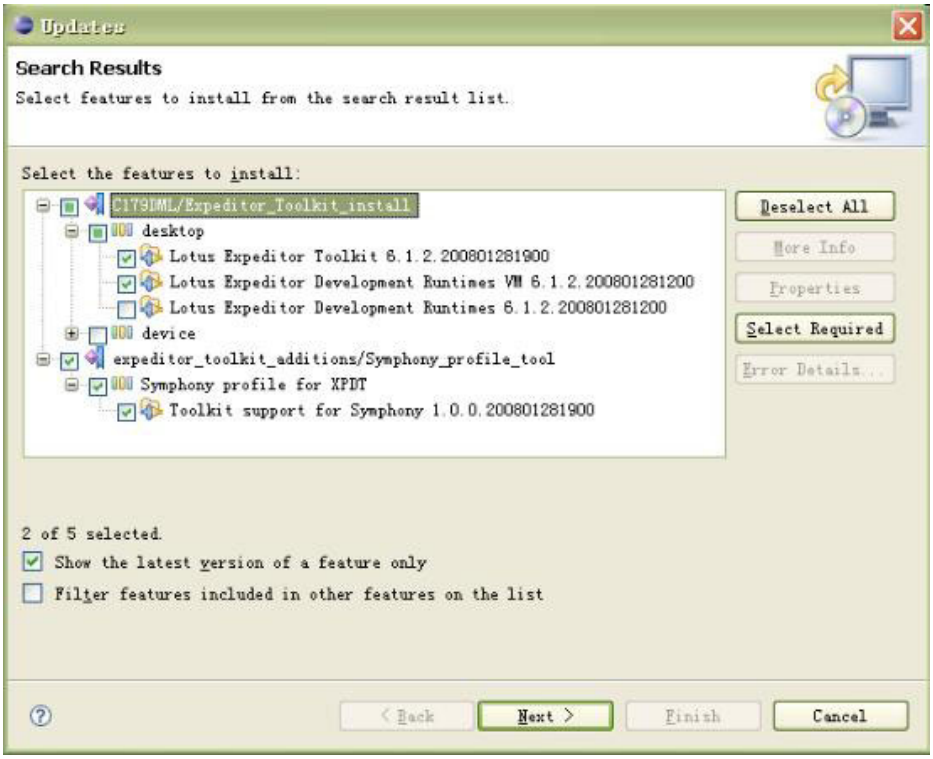


h. Select the **New Local Site** button again and select the extracted file from **step b**, expand it and select **Symphony_profile_tool**, and then click **OK**.



i. Select the check box next to the site name `*/Expeditor_Toolkit_install` and `*/Symphony_profile_tool`, and then click **Finish** (*means the folder name containing the selected folder.).

j. In the search results page, select all of the features available and click **Next**. The following figure shows the minimum needed features for Lotus Symphony development (Because **Symphony_profile_tool** has dependence on **Expeditor_Toolkit_install**, first select the two features under the desktop and then select the features under **Symphony_profile_tool** to avoid an exception).



Note: When selecting Lotus Expeditor's feature on Red Hat or SuSE operating systems, it can throw a `NullPointerException`. This exception doesn't cause problems during installation.

k. In the **Feature License** window, read the licensing information for each feature that you are installing, and if you agree with the license, select to accept the license and click **Next**.

l. Click **Finish** to begin the installation.

m. In the **Feature verification** window, verify that the feature information is correct and click **Install**.

Note: When warned about installing an unsigned feature, click **Install** to continue. This warning does not cause problems during installation.

n. When installation completes, you are prompted to restart your IDE for changes to take effect. Click **Yes** to continue.

Note: Clicking **Apply Changes** does not properly configure the environment.

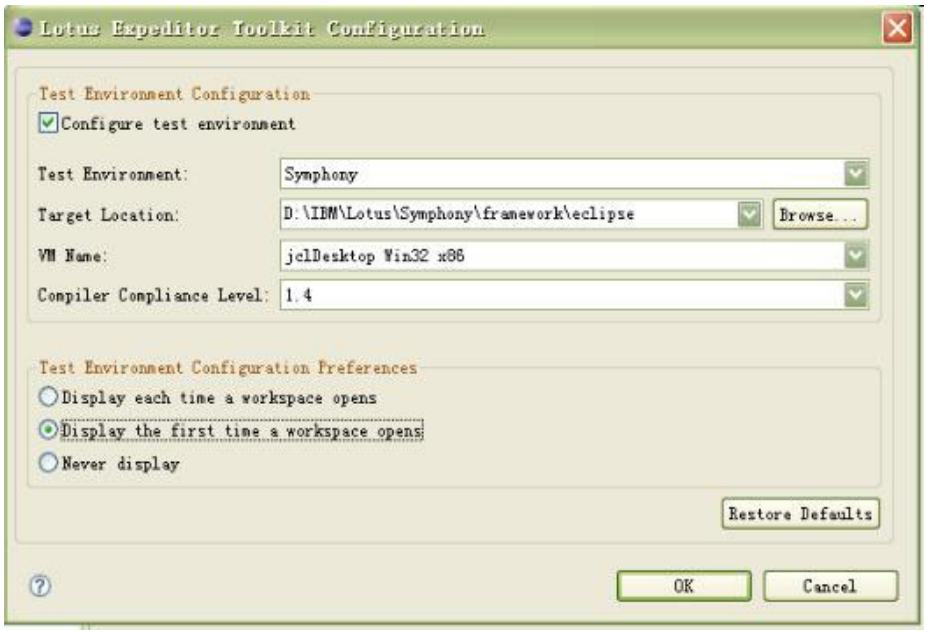
3. Configure Lotus Symphony Support.

After restarting the IDE, you are presented with the Expeditor Toolkit Configuration box. To configure the toolkit for usage with Lotus Symphony, following these steps:

a. Select **Symphony** in the **Test Environment**.

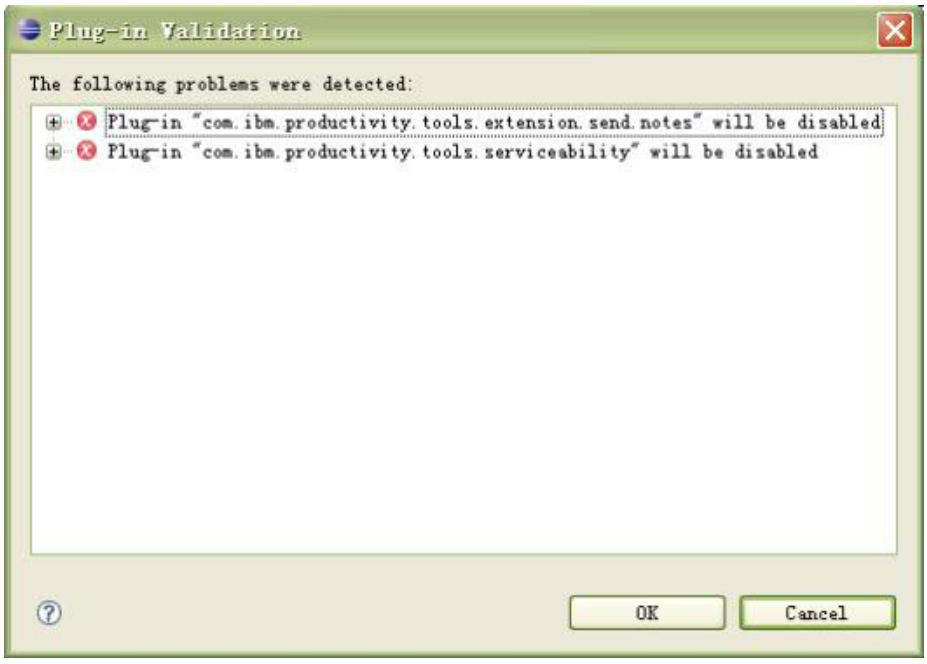
b. Use the **Browse** button to select the **Eclipse** directory of the Lotus Symphony installation location, for example, `<Symphony installation home>\framework\eclipse`.

c. Click **OK**.



4. Create your own project code in this Eclipse environment.
5. Launch Lotus Symphony.
   a. Select **Run > Run** or **Run > Debug**.
   b. Select the **Client Services** launch type and click the new icon or double click **Client Services**, named it `Symphony`.
   c. Click **Run** or **Debug** to start Symphony.

      The build of Lotus Symphony that was tested with the toolkit does not resolve all plug-ins correctly. Therefore, you can see an error similar to the one following. If this error occurs, click **OK** to continue the launch process. You can disable this checking operation by clearing the mark next to **Validate plug-in dependencies...** at the bottom of the **plug-ins** tab of the launcher.



Note: Use Java compiler 1.4 as plug-ins' Java compiler. Java compiler 5.0 might not work correctly.

Note: On Red Hat systems, sometimes a `java.lang.UnsatisfiedLinkError` exception is thrown when launching the Lotus Symphony. Try to fix it with the command similar to the following:

```
ldconfig /opt/ibm/lotus/Symphony/framework/shared/eclipse/plugins/
com.ibm.productivity.tools.base.system.linux_3.0.1.*
```

Chapter 2. Customizing the Lotus Symphony User Interface

The followed examples are all need you build a plug-in project, and then edit the `plugin.xml` file directly by the code provided below. If you are not familiar with how to build a plug-in project, go to Part 6 Example plug-in to see the details.

2.1 Adding a sample menu

Lotus Symphony allows you to add new menus to its main menu. The addition is achieved through the Eclipse extension point: `org.eclipse.ui.actionSets`.

For convenience, menus from third parties should be added under the menu Add-ins. If another third party has defined the menu "Add-ins", you can use it; otherwise, you should define such a menu and use it.

To add a sample menu to the Add-ins menu, perform the following steps:

1. Extend `org.eclipse.ui.actionSets` extension point in the `plugin.xml` file:

```
<extension point="org.eclipse.ui.actionSets">
          <actionSet id="com.ibm.lotus.symphony.example.ui.actionSet"
                    label="example action set"
                    visible="true">
               <menu
                    id="com.ibm.lotus.symphony.addinsmenu"
                    label="Add-ins"
                    path="additions">
                    <separator name="additions"/>
               </menu>
               <action id="com.ibm.lotus.symphony.example.ui.exampleAction"
                    menubarPath="com.ibm.lotus.symphony.addinsmenu/additions"
                    label="Sample Menu"
                    tooltip="Sample Menu Tooltip"
                    class="com.ibm.lotus.symphony.example.ui.ExampleAction"
                    enablesFor="1">
               </action>
         </actionSet>
</extension>
```

   The label property of the action element specifies the name of the menu item or toolbar button label. The menubarPath and toolbarPath properties specify their location in the menu bar and toolbar.

2. Implement the action class:

```
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

public class ExampleAction implements IWorkbenchWindowActionDelegate {
            private IWorkbenchWindow window;

            /*
             * (non-Javadoc)
             *
             * @see org.eclipse.ui.IWorkbenchWindowActionDelegate#dispose()
             */
            public void dispose() {
            }

            /*
             * (non-Javadoc)
             *
             * @see org.eclipse.ui.IWorkbenchWindowActionDelegate#init
            (org.eclipse.ui.IWorkben
             *      chWindow)
             */
            public void init(IWorkbenchWindow window) {
                this.window = window;
            }

            /*
             * (non-Javadoc)
             *
             * @see org.eclipse.ui.IActionDelegate#selectionChanged
            (org.eclipse.jface.action.I
             *      Action, org.eclipse.jface.viewers.ISelection)
             */
            public void selectionChanged(IAction action, ISelection selection) {
            }
```

```
        /*
         * (non-Javadoc)
         *
         * @see org.eclipse.ui.IActionDelegate#run
        (org.eclipse.jface.action.IAction)
         */
        public void run(final IAction action) {
            MessageDialog.openInformation(window.getShell(), "Information",
                    "Menu pressed");
        }
}
```

The action class must implement `IWorkbenchWindowActionDelegate`, or `IWorkbenchWindowPulldownDelegate`, for the action to be shown as a pull-down tool item in the toolbar.

Package

The extension point is provided by Eclipse Rich Client Platform.

See Also

http://publib.boulder.ibm.com/infocenter/wsphelp/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_actionSets.html

Example

The code above results in the following display of the menu:

2.2 Adding a control to the toolbar

Lotus Symphony allows you to add to the main toolbar. You should add your own toolbar group. The addition is achieved through Expeditor extension point: `com.ibm.rcp.ui.controlSets`.

To add items to the Lotus Symphony main toolbar, perform the following steps:

1. Make sure that your plug-in have the following dependencies:
   - `com.ibm.productivity.tools.core`
   - `com.ibm.productivity.tools.ui.toolbar`
   - `com.ibm.rcp.jfaceex`
2. Extend the `com.ibm.rcp.ui.controlSets` extension point in `plugin.xml`:

```
<extension
        point="com.ibm.rcp.ui.controlSets">
     <controlSet
           id="com.ibm.productivity.tools.sample.documentworkflow.controlset"
           label="Sample Control Set"
           preferredWidth="20%"
           visible="false">
        <toolBar
              id="com.ibm.productivity.tools.sample.documentworkflow.toolBar"
              path="BEGIN_GROUP">
        </toolBar>
        <control
              class="com.ibm.productivity.tools.sample.documentworkflow.SampleControl
           "
              id="com.ibm.productivity.tools.sample.documentworkflow.control"
              toolbarPath="com.ibm.productivity.tools.sample.documentworkflow.toolBar
           ">
        </control>
     </controlSet>
</extension>
```

3. Provide a class to define your control:

```
import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.ui.PlatformUI;

import com.ibm.productivity.tools.ui.toolbar.Activator;
import com.ibm.productivity.tools.ui.toolbar.SODCActionContributionItem;

public class SampleControl extends SODCActionContributionItem {

           public IAction createAction() {
                Action action = new Action() {

                     public void run() {
                          MessageDialog.openInformation(PlatformUI.getWorkbench()
                                   .getActiveWorkbenchWindow().getShell(),
           "Information",
                                   "Control pressed");
                     }

                };
                action.setText("Sample");
                action.setToolTipText("Sample");
//              action.setImageDescriptor(Activator.imageDescriptorFromPlugin(
//                       Activator.PLUGIN_ID, "docs/itemCampo.png"));
                return action;
           }
}
```

4. Optional. Define an association in plugin.xml file if you want to associate your toolbar with Lotus Symphony views.

   com.ibm.productivity.tools.ui.toolbar.controlSetSODCAssociations is an extension point defined to associate control sets with Lotus Symphony views so that those associated control sets only display when a Lotus Symphony view is activated. To extend this extension point, in the first place, a control set has been defined.

   The class attribute of control has to be a class that is a sub-class of SODCActionContributionItem, which is defined in bundle com.ibm.productivity.tools.ui.toolbar. More, the visible attribute of the control set has to be set to false.

   To associate this control set with a Lotus Symphony view, define the following extension:

```
<extension
      point="com.ibm.productivity.tools.ui.toolbar.controlSetSODCAssociations">
   <controlSetSODCAssociation>
      <controlSet
            id="com.ibm.productivity.tools.sample.documentworkflow.controlset"
            visible="true">
      </controlSet>
   </controlSetSODCAssociation>
</extension>
```

Here, the visible attribute defines if this control set is displayed by default.

Package

com.ibm.rcp.platform.controlSets are defined in Lotus Expeditor platform.

`com.ibm.productivity.tools.ui.toolbar.controlSetSymphonyAssociations` are defined in `com.ibm.productivity.tools.ui.toolbar` plugin.

See Also

http://publib.boulder.ibm.com/infocenter/wsphelp/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_actionSets.html

Or, from the local help contents on Eclipse by following: Start up Eclipse > **Help menu> Help Centents > Platform Plug-in Developer Guide > Reference > Extension Points Rference > org.eclipse.ui.actionSets**.

Example

The sample code above results in the following display on the toolbar:

2.3 Adding to the launcher button

Lotus Symphony allows you to add its **New** button, which is under the main menu area. The contribution is achieved through the Eclipse extension point: `com.ibm.rcp.ui.launcherSet.`

The extension point `com.ibm.rcp.ui.launcherSet` supports many types of launch items including:

- A URL launch item, which opens a URL.
- A perspective launch item, which opens a perspective.
- A native program launch item, which opens a native program on the system.
- A custom launch item other than a URL, perspective ID or native program.

The following markup adds a new perspective launch item:

```
<extension
         point="com.ibm.rcp.ui.launcherSet">
      <LauncherSet
            id="sym.guide.test.LauncherSet"
            label="Symphony Home Web">

         <urlLaunchItem
           iconUrl="http://www.ibm.com/i/v14/t/us/en/search.gif"
           id=" com.ibm.productivity.tools.sample.tests.googleLauncherItem"
           label="Test URL Launcher Item - Symphony "
           url="http://symphony.lotus.com/"/>

      </LauncherSet>
</extension>
```

Package

The extension point is provided by Lotus Expeditor.

See Also

http://publib.boulder.ibm.com/infocenter/ledoc/v6r11/index.jsp?topic=/com.ibm.rcp.doc.schemas/reference/extension-points/com_ibm_rcp_ui_launcherSet.html

Or from the Lotus Expeditor local help content on Eclipse after you finished setting up the Lotus Symphony development environment (Refer to part 4 chapter 1) : Start up Eclipse > **Help > Help Contents > Developing Applications for Lotus Expeditor > Reference information > Extension points schemas > com.ibm.rcp.ui.launcherSet**.

Example

2.4 Adding a New View in the Shelf View

A sidebar is a stack of shelf views typically located on either the right or left side of the Lotus Symphony user interface. Plug-in developers can add views to a sidebar in the user interface, which is based on the Lotus Expeditor extension point: `com.ibm.rcp.ui.shelfViews`.

Lotus Symphony makes use of the Eclipse **IViewPart interface** to tie each shelf view to the workbench. Each view part has a view site that connects it to the workbench, allowing the view to register any global actions with the site's action bars, including access to its own panel menu, a local toolbar, and the status line. The view can also register any context menus with the site, or register a selection provider to allow the workbench's **ISelectionService** to include the part in its tracking.

To add items to the Lotus Symphony shelf view, perform the following steps:

1. Make sure that your plug-in have the following dependencies:
   - `com.ibm.productivity.tools.ui.views`
   - `com.ibm.productivity.tools.core`
   - `com.ibm.rcp.jfaceex`
   - `com.ibm.rcp.ui`
   - `com.ibm.rcp.swtex`
2. Extend the `com.ibm.rcp.ui.shelfViews` extension point in `plugin.xml`:

```
<extension
        point="com.ibm.rcp.ui.shelfViews">
     <shelfView
          id="com.ibm.productivity.tools.sample.ShelfView"
          page="LEFT"
          region="BOTTOM"
          showTitle="true"
          view="com.ibm.productivity.tools.sample.ShelfView"/>
</extension>
```

3. Add to the `org.eclipse.ui.views` extension point in the `plugin.xml` file for the plug-in, as seen in the following example:

```
<extension
        point="org.eclipse.ui.views">
     <category
          name="Sample Category"
          id="com.ibm.productivity.tools.sample">
     </category>
     <view
          name="Document Sample"
          icon=" "
          category="com.ibm.productivity.tools.sample"
          class="com.ibm.productivity.tools.sample.ShelfView"
          id="com.ibm.productivity.tools.sample.ShelfView">
     </view>
</extension>
```

Make sure that the following attributes are specified:

- The name attribute describes the string to be displayed in the title bar.
- The id attribute is the unique identifier of the view and is used to refer to the view when contributing to the shelfViews extension point.
- The class attribute specifies what class is referenced in this extension.
- The icon attribute describes the icon to be displayed in the top left corner of the title bar. The standard size is 16 x 16 pixels.
- The view should be optimally viewed in a frame approximately 186 pixels wide. The view is also resizable. Make sure that the content can be scrolled (if applicable), and that any toolbars do not get cut off, or have chevrons pointing to more actions.

4. Implement the view class:

```
package com.ibm.productivity.tools.sample;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.part.ViewPart;

public class ShelfView extends ViewPart {

            public void createPartControl(Composite arg0) {
               // TODO Auto-generated method stub

            }

            public void setFocus() {
               // TODO Auto-generated method stub

            }
}
```

Package

The extension point is provided by Lotus Expeditor.

See Also

http://publib.boulder.ibm.com/infocenter/ledoc/v6r11/index.jsp?topic=/com.ibm.rcp.tools.doc.appdev/ui_contributingtosideshelfsidebar.html

Or from the Lotus Expeditor local help content on Eclipse after you finished setting up the Lotus Symphony development environment (refer to part 4 chapter 1) : Start up Eclipse > **Help > Help Contents > Developing Applications for Lotus Expeditor > Developing applications > Developing the application user interface > Using personalities > Contributing to the sidebar**.

Example

2.5 Using the Auto Recognizer

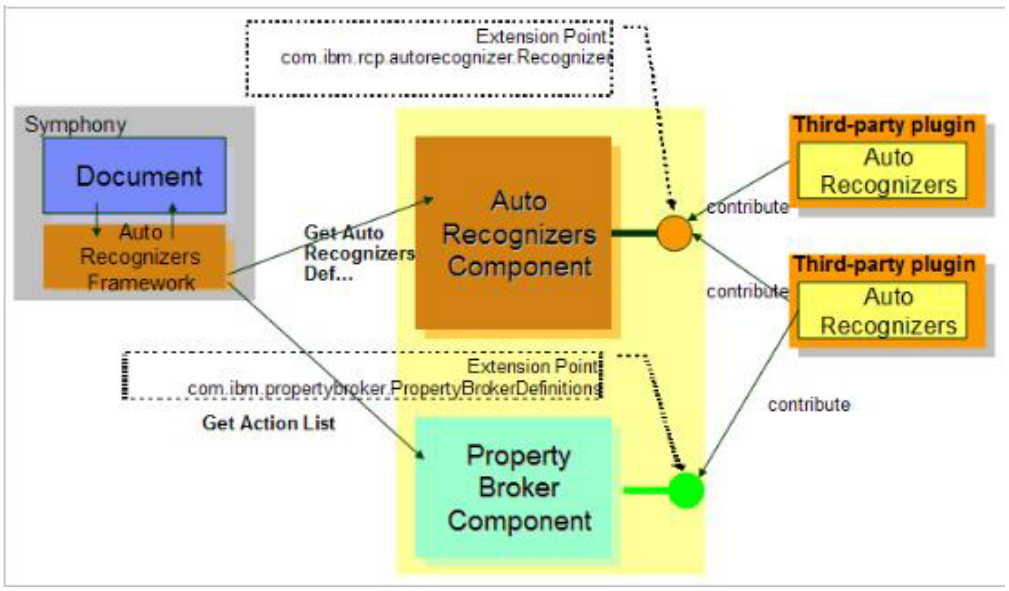
The auto recognizer is a framework to allow users to take actions based on the text that they input in the Lotus Symphony editor. It assists users to do extra operations on the content of a document by underlining items in a special pattern. It provides a gateway to provide further information and activities related to the identified item, specific to users' needs. By using auto recognizer, Lotus Symphony can provide a more collaborative environment.

Note: Auto recognizer is only available in the Writer application and only single word patterns are supported.

Lotus Symphony provides the auto recognizer framework and also the auto recognizer component, `PropertyBroker`, which is inherited from the Lotus Expeditor platform. To use the auto recognizer, you must follow these steps:

1. Add dependencies on the `com.ibm.rcp.autorecognizer` and `com.ibm.rcp.propertybroker` plugins.
2. Implement a detector to define how to detect patterns.
3. Add the action to the `com.ibm.rcp.propertybroker.PropertyBrokerDefinitions` extension point.
4. Add the recognizer to the `com.ibm.rcp.autorecognizer.Recognizer` extension point

The following figure is the overall architecture of the auto recognizer framework.



Adding the Auto Recognizer to the Extension Point

To add the auto recognizer, perform the following steps:

1. Add the com.ibm.rcp.autorecognizer.Recognizer extension point in the `plugin.xml` file:

```
<extension
      point="com.ibm.rcp.autorecognizer.Recognizer">
  <types>
      <define-method id="SampleRecognizer">
        <type
               datatype="SampleType"
               default-name="SampleType"
               multi-segment="true"
               namespace="http://www.ibm.com/wps/c2a"/>
   <custom class="com.ibm.productivity.tools.samples.C2A.recognizer.SampleDetector"/
         >
      </define-method>
  </types>
</extension>
```

2. Implement a SampleDetector class to define how to detect the pattern. Only a single word is detected by the underlying auto recognizer framework in the document:

```
import java.util.ArrayList;

import com.ibm.rcp.autorecognizer.recognizer.DetectResult;
import com.ibm.rcp.autorecognizer.recognizer.IDetect;

public class SampleDetector implements IDetect {

          private String m_Itemlist[] = {"PropertyBroker","AutoRecognizer"};
          public static ArrayList taglist= new ArrayList();

          /* (non-Javadoc)
           * @see com.ibm.rcp.autorecognizer.recognizer.IDetect#detect
          (java.lang.String)
           */
          public DetectResult detect(String word) {
              try {
                   for (int i = 0; i < m_Itemlist.length; i++) {

                        if (m_Itemlist[i].equals(word)) {
                             DetectResult rlt = new DetectResult();
                             rlt.start = 0;
                             rlt.offset = word.length();
                             rlt.value = word;
                             return rlt;
                        }
                   }
              } catch (Exception e) {
                   e.printStackTrace();
              }
              return null;
          }
}
```

Add an Action

To add an action, perform the following steps:

1. Add the `com.ibm.propertybroker.PropertyBrokerDefinitions` extension point in the `plugin.xml` file:

```
  <extension
       point="com.ibm.rcp.propertybroker.PropertyBrokerDefinitions">
      <handler
            class="com.ibm.productivity.tools.samples.C2A.actions.SampleAction"
            file="wsdl/SampleAction.wsdl"
            type="SWT_ACTION"/>
</extension>
```

2. Define the `SampleAction.wsdl` file:

```
<definitions name="Sample_Service"
             targetNamespace="http://www.ibm.com/wps/c2a"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:portlet="http://www.ibm.com/wps/c2a"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://www.ibm.com/wps/c2a"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<types>
             <xsd:schema targetNamespace="http://www.ibm.com/wps/c2a">
                  <xsd:simpleType name="SampleType">
                       <xsd:restriction base="xsd:string">
                       </xsd:restriction>
                  </xsd:simpleType>
                  <xsd:simpleType name="Sample_Status">
                       <xsd:restriction base="xsd:boolean">
                       </xsd:restriction>
                  </xsd:simpleType>
             </xsd:schema>
</types>

<message name="Sample_Keyword">
             <part name="keyword" type="tns:SampleType"/>
</message>

<message name="Sample_Status">
             <part name="sample_status" type="tns:Sample_Status"/>
</message>
```

```
<portType name="Sample_Service">
            <operation name="sample_event">
                <input message="tns:Sample_Keyword"/>
                <output message="tns:Sample_Status"/>
            </operation>
</portType>

<binding name="SampleBinding" type="tns:Sample_Service">

            <portlet:binding/>

            <operation name="sample_event">

                <portlet:action name="SampleAction"
                    type="standard"
                    caption="SampleAction"
                    description="Sample Event"
                    actionNameParameter="ACTION_NAME"/>

                <input>
                    <portlet:param name="keyword" partname="keyword"
            caption="Sample.Event"/>
                </input>

                <output>
                    <portlet:param name="sample_status" partname="sample_status"
            caption="Sample.Status"/>
                </output>

            </operation>
</binding>
</definitions>
```

3. Implement a SampleAction class:

```
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.commands.IHandler;
import org.eclipse.core.commands.IHandlerListener;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;

public class SampleAction implements IHandler {

            public void addHandlerListener(IHandlerListener arg0) {
                //do nothing
            }

            public void dispose() {
                //do nothing
            }
            /**
             * while clicking the context menu, this method will be invoked.
             */
            public Object execute(ExecutionEvent event) throws ExecutionException {
                final PropertyChangeEvent evt = (PropertyChangeEvent) event.getTrigger
            ();
                Display.getDefault().asyncExec(new Runnable() {

                    /* (non-Javadoc)
                     * @see java.lang.Runnable#run()
                     */
```

```
                    public void run() {
                        //open an message box.
                        Display dsp = Display.getCurrent();
                        Shell sh = new Shell(dsp);
                        MessageBox box = new MessageBox(sh, SWT.ICON_INFORMATION);
                        box.setText("Event");
                        box.setMessage("Sample event triggered by: "
                                + evt.getPropertyValue().getValue());
                        box.open();
                    }
                });
                return null;
            }

            public boolean isEnabled() {
                return false;
            }

            public boolean isHandled() {
                return false;
            }

            public void removeHandlerListener(IHandlerListener arg0) {
                //Do nothing
            }
}
```

Package

`com.ibm.rcp.autorecognizer`.

See Also

Property broker extension point in Lotus Expeditor:

http://publib.boulder.ibm.com/infocenter/ledoc/v6r1/index.jsp?topic=/com.ibm.rcp.doc.schemas/reference/extension-points/com_ibm_rcp_propertybroker_PropertyBrokerDefinitions.html

Or from the Expeditor local help content on Eclipse after you finished setting up Lotus Symphony development environment(refer to part 4 chapter 1) by following: Start up Eclipse > **Help > Help Contents > Developing Applications for Lotus Expeditor > Reference information > Extension points schemas > com.ibm.rcp.propertybroker.PropertyBrokerDefinitions**.

Example

In following example, a plug-in defines that **PropertyBroker** and **AutoRecognizer** are two keywords, and a special action (in this example, **SampleAction**) is added to this pattern. When the keywords are found in the document, the words are underlined which indicates that this is a special pattern. If users move the cursor

to the pattern, pull-down button displays and they can click the button to invoke pattern-related actions. The source code for this example is provided above.

2.6 Adding an item to the status bar

Lotus Symphony allows the addition of arbitrarily sophisticated user interface controls to the status bar and the toolbar, through the Lotus Expeditor extension point `com.ibm.rcp.ui.controlSets`.

To add an item into status bar, complete the following steps:

1. Add the `com.ibm.rcp.ui.controlSets` extension point in the `plugin.xml` file:

```
<extension
      point="com.ibm.rcp.ui.controlSets">
   <controlSet
         visible="true"
         id="example.ControlSet">
      <statusLine
            path="BEGIN_GROUP"
            id="example.statusline">
         <groupMarker name="additions"/>
      </statusLine>
      <control
            statusLinePath="example.statusline/additions"
            class="com.ibm.Lotus.Symphony.example.ExampleStatusbarItem"
            id="example.control"/>
   </controlSet>
</extension>
```

   The **statusLine** element defines a marker location for other status line items to be added similarly to the menu element in **actionSet**. The **statusLinePath** property specifies the path in the **statusbar**.

2. Implement the control class:

```
package com.ibm.Lotus.Symphony.example;

import org.eclipse.jface.action.ContributionItem;
import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.CLabel;
import org.eclipse.swt.widgets.Composite;

public class ExampleStatusbarItem extends ContributionItem {
            public void fill(Composite parent) {
                CLabel label = new CLabel(parent, SWT.SHADOW_IN | SWT.LEFT);
                label.setSize(300, 20);
                label.setText("status");
                label.setToolTipText("text");
            }
}
```

The control class must implement `IContributionItem` and implement `fill (Composite parent)`.

Package

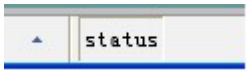
This extension point is provided by Lotus Expeditor.

See Also

http://publib.boulder.ibm.com/infocenter/ledoc/v6r11/index.jsp?topic=/com.ibm.rcp.doc.schemas/reference/extension-points/com_ibm_rcp_ui_controlSets.html

Or from the Lotus Expeditor local help content on Eclipse after you finished setting up Lotus Symphony development environment(refer to part 4 chapter 1) : Start up Eclipse > **Help > Help Contents > Developing Applications for Lotus Expeditor > Reference information > Extension points schemas > com.ibm.rcp.ui.controlSets**.

Example



2.7 Adding a Preferences Page

After a plug-in has added extensions to the Lotus Symphony user interface, preferences page lets users control some of the behaviors of the plug-in through user preferences.

Store plug-in preferences and show them to the user on pages in the Lotus Symphony Preferences window. Plug-in preferences are key value pairs in which the key describes the name of the preference and the value is one of several different types.

The `org.eclipse.ui.preferencePages` extension point lets you add pages to the Lotus Symphony preferences (**File > Preferences**). The preferences window

presents a hierarchical list of user preference entries. Each entry displays a corresponding preference page when selected.

To add a preference page, complete the following steps:

1. Add the `org.eclipse.ui.preferencePages` extension point in the `plugin.xml` file:

```
<extension
      point="org.eclipse.ui.preferencePages">
     <page
           class="com.ibm.lotus.symphony.example.preferences.ExamplePreferencePage"
           id="com.ibm.lotus.symphony.example.preferences.ExamplePreferencePage"
           name="Lotus Symphony Example"
           category="com.ibm.productivity.tools.core.preferences.documenteditors.Docu
mentEditors"/>
</extension>
```

   This markup defines a preference page named ″Lotus Symphony Example″ which is implemented by the class `ExamplePreferencePage`.

2. Add the `org.eclipse.core.runtime.preferences` extension point in the `plugin.xml` file:

```
<extension
      point="org.eclipse.core.runtime.preferences">
           <initializer
           class="com.ibm.lotus.symphony.example.preferences.PreferenceInitializer"/>
</extension>
```

   The extension point `org.eclipse.core.runtime.preferences` lets plug-ins add new preference scopes to the Eclipse preference mechanism and to specify the class to run that initializes the default preference values at runtime.

3. Implement the page class.

   The page class must implement the `IWorkbenchPreferencePage` interface.The content of a page is defined by implementing a `createContents` method that creates the SWT controls representing the page content:

```
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.preference.PreferencePage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPreferencePage;

//import sym.guide.test.Activator;

public class ExamplePreferencePage extends PreferencePage implements
                IWorkbenchPreferencePage {
           private Text usrID;

           public ExamplePreferencePage() {
                super();
                setPreferenceStore(Activator.getDefault().getPreferenceStore());
                setDescription("example preference");
           }
           protected Control createContents(Composite parent) {
                Composite composite = new Composite(parent, SWT.NULL);
                composite.setLayout(new GridLayout(2, false));
                Label usrLabel = new Label(composite, SWT.NONE);
                usrLabel.setText("User");
                usrID = new Text(composite, SWT.BORDER|SWT.RIGHT);
                usrID.setLayoutData(new GridData(100, SWT.DEFAULT));
                initializeValues();
                return composite;
           }
```

```
            private void initializeValues() {
                IPreferenceStore store = getPreferenceStore();
                String userID = store.getString("USER_ID");
                usrID.setText(userID);
            }
            protected void performApply() {
                IPreferenceStore store = getPreferenceStore();
                store.setValue("USER_ID", usrID.getText());
            }
            public boolean performOk() {
                performApply();
                return super.performOk();
            }
            protected void performDefaults() {
                IPreferenceStore store = getPreferenceStore();
                usrID.setText(store.getDefaultString("USER_ID"));
            }
           public void init(IWorkbench arg0) {
           }
}
```

4. Implement the page class and initialize class.
   The initialize class is used for preference initialization:

```
package com.ibm.lotus.symphony.example.preferences;

import org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer;
import org.eclipse.jface.preference.IPreferenceStore;
//import sym.guide.test.Activator;

public class PreferenceInitializer extends AbstractPreferenceInitializer {
            /*
             * (non-Javadoc)
             *
             * @see
            org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer#initial
             *      izeDefaultPreferences()
             */
            public void initializeDefaultPreferences() {
                IPreferenceStore store = Activator.getDefault().getPreferenceStore();
                store.setDefault("USER_ID", "tom");
            }
}
```

Note: If you want to contribute the preference page to root node, you can add the following code in `plugin.xml` file. The id is the preference id when you define your preference page. For example, **WebBrowserPreferencePage** is the id for browser component provided within Symphony.

```
<extension
    point="com.ibm.productivity.tools.baseshell.preference">
    <preferenceid id="WebBrowserPreferencePage" />
</extension>
```

Package

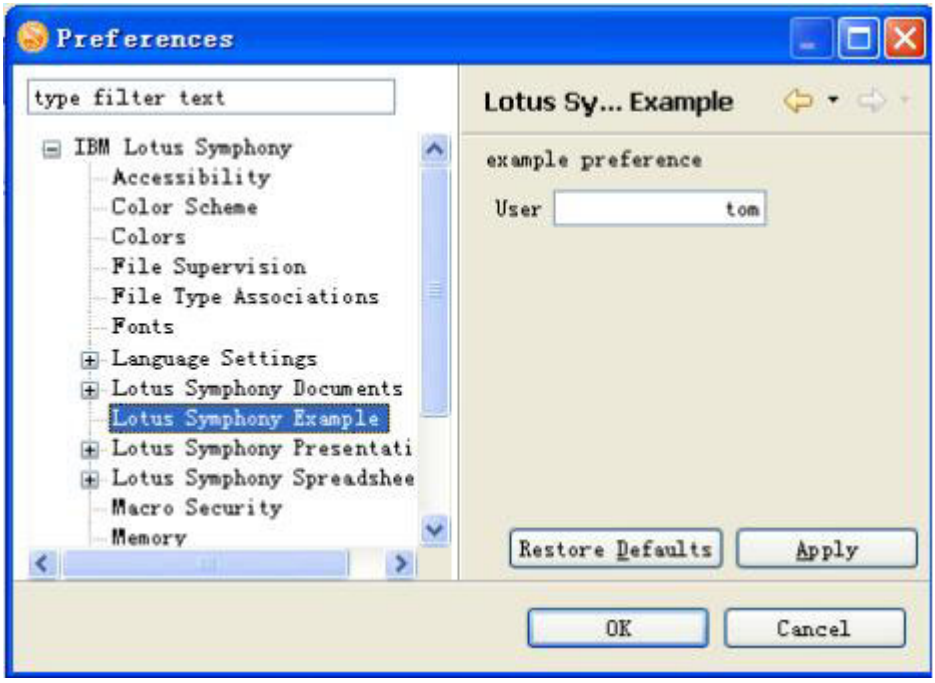
The extension point is provided by Eclipse Rich Client Platform.

See Also

http://publib.boulder.ibm.com/infocenter/wsphelp/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/extension-points/org_eclipse_ui_preferencePages.html

Or from the local help contents on Eclipse by following: Start up Eclipse > **Help menu> Help Contents > Platform Plug-in Developer Guide > Reference > Extension Points Reference > org.eclipse.ui.preferencePages**.

Example



Chapter 3. Lotus Symphony Java APIs and Extension Points

3.1 Selection Service

In Eclipse, the selection service provided by the Eclipse workbench allows efficient linking of different parts within the workbench window. Each workbench window has its own selection service instance. The service keeps track of the selection in the currently active part and propagates selection changes to all registered

listeners. Such selection events occur when the selection in the current part is changed or when a different part is activated. Both can be triggered by user interaction or programmatically.

Each Lotus Symphony view registers the selection provider, so it is possible to monitor if a selection change event occurs.

When opening or creating a document by user interaction or programmatically, the view is opened as an Eclipse ViewPart. The view registers the selection provider to Eclipse workbench window. When an application registers a selection listener, the listener is notified when the selection is changed in the view.

From the user's point of view, a selection is a set of highlighted text or objects in a view. Internally, a selection is a data structure holding the model objects which correspond to the graphical elements selected in the view. Almost all text or objects can be selected in the view for these kinds of applications: writer, spreadsheet, and presentation. The selection can be presented in several ways and you can only get the text content from the selection. It might be possible to present the selection using HTML, ODF, or XML format.

Accessing the Current Selection

The Lotus Symphony workbench keeps track of the currently selected part in the window and the selection within this part. Each view registers it as the selection provider, even if you do not need to propagate its selection now. Your plug-in is ready for future extensions by others.

To access the current selection of current Lotus Symphony view:

```
IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
ISelectionService service = window.getSelectionService();
ISelection selection = service.getSelection();
```

Retrieving Text Content From the Selection

To get the text content from the selection:

```
IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
ISelectionService service = window.getSelectionService();
ISelection selection = service.getSelection();
IAdaptable adaptable = ( IAdaptable )selection;
RichDocumentContentSelection textSel = (RichDocumentContentSelection)
            adaptable.getAdapter(RichDocumentContentSelection.class );
String text = textSel.getPlainText();
```

Tracking Selection Change

Typically views react on selection changes in the Lotus Symphony workbench window, however, it is better to register an `ISelectionListener` to get notified when the window's current selection changes:

```
IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
ISelectionService service = window.getSelectionService();
ISelectionListener listener = new ISelectionListener(){
    public void selectionChanged( IWorkbenchPart part, ISelection selection ){
        //do something…
    }
};
service.addSelectionListener( listener );
```

Removing the Selection Listener

Remove the selection listener when you cannot handle events, such as when your view has been closed. Use the `dispose()` method to remove your listener:

```
public void dispose() {
     IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
     ISelectionService service = window.getSelectionService();
     service.removeSelectionListener( listener );
     super.dispose();
}
```

Package

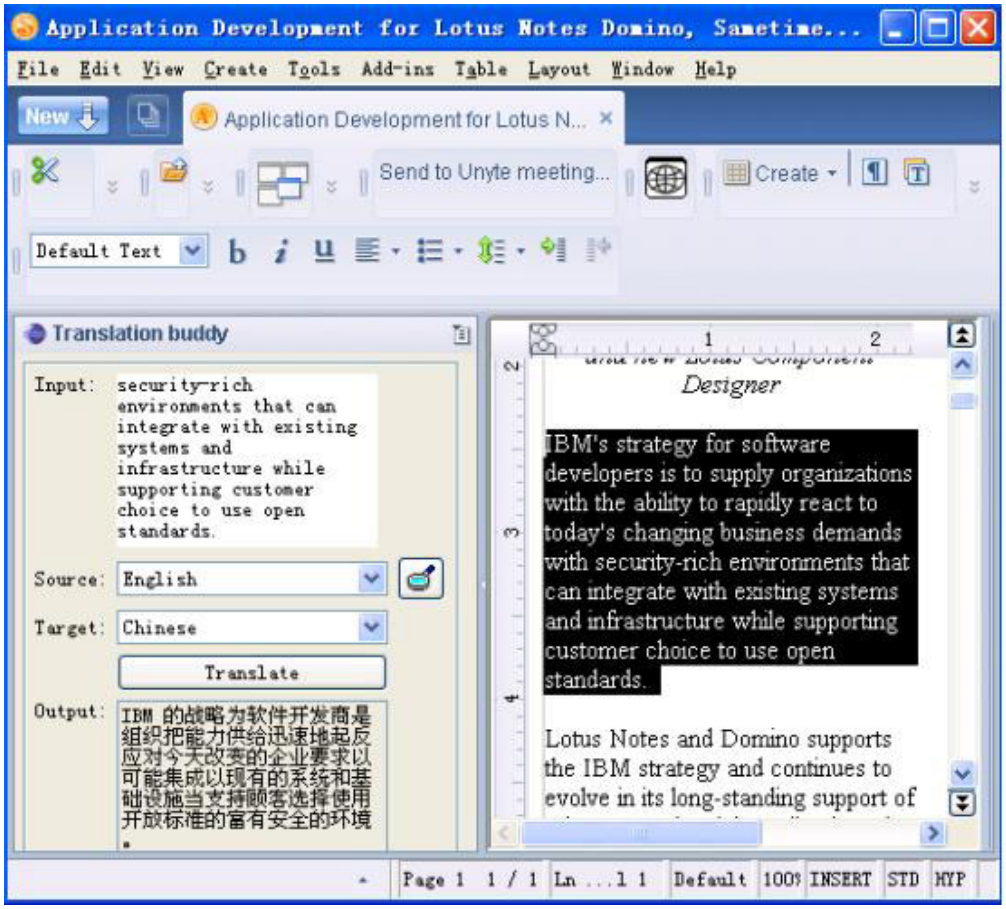
com.ibm.productivity.tools.ui.views

See Also

JavaDoc in Lotus Symphony toolkit.

Example

The sample **Translation buddy** view behaves in this way: whenever the text content selection changes in the Lotus Symphony writer view, the selected text is displayed on the Input area of the view automatically.



3.2 RichDocumentViewFactory

The `RichDocumentViewFactory` class handles the creation, accessing and closing of a rich document view. A listener can be registered through `com.ibm.productivity.tools.ui.views.listener` extension points to monitor the opening and closing of a rich document view. The factory class provides global static methods to handle the rich document views.

The factory class is used to create, open or close rich document view programmatically.

1. Create new document through the user interface or API, for example, click **File->New->Document**.
2. Open the document through the user interface or API.
3. Close the document through the user Interface or API.
4. Get the list of opened views using the API.

Creating a New Rich Document View

Use the following example code to create a new rich document view by specifying whether the document type is writer, spreadsheet or presentation type at creation time:

```
RichDocumentViewFactory.openView( RichDocumentType.DOCUMENT_TYPE);
```

For more information about how to configure the map, see the Javadoc in the Lotus Symphony toolkit.

Opening a Local File in a New Rich Document View

Use the following example code to open a file in a new rich document view:

```
String fileName = .....; //e.g. c:\\temp.odt
RichDocumentViewFactory.openView( fileName, false );
```

You can also specify the configuration map as same as opening new document. In the above code, you set the properties template to close mode. You can also decide whether you want to load the document as a template.

Typically, the document is loaded in a new tab, which depends on the windows and theme settings, of the preference page.

Getting the List of Opened Rich Document Views

Use the following example code to get the list of opened rich document views:

```
RichDocumentView[] views = RichDocumentViewFactory.getViews();
```

All rich document views opened in Lotus Symphony are returned.

Closing a Rich Document View

Use the following code to close a rich document view. The window tab is closed when the view is closed:

```
RichDocumentView view = ….; // get an instance of rich document view
RichDocumentViewFactory.closeView( view );
```

Registering the Listener Using the Extension Point

The `com.ibm.productivity.tools.ui.views.listener` extension point is defined to monitor the status of the `RichDocumentView` instance. If a listener is registered when `RichDocumentView` is created, closed or a document is loaded, then the listener is notified. Currently the following events for rich document views are supported:

- Type_Pre_Document_Open. A rich document is about to be opened in a view.
- Type_Post_Document_Open. A rich document is opened in a view.
- Type_Pre_Document_Close. A rich document is about to be closed in a view.
- Type_Post_Document_Close. A rich document is closed in a view.
- Type_Post_Open. A rich document view is opened.
- Type_Pre_View_Close. A rich document view is about to be closed.
- Type_Post_View_Close. A rich document view is closed.

To use the listener, perform the following steps:

1. Add the `com.ibm.productivity.tools.ui.views.listener` extension point:

```
<extension
      id="SampleListener"
      name="Sample Listener"
      point="com.ibm.productivity.tools.ui.views.listener">
      <listener
         class="com.ibm.productivity.tools.sample.views.SampleListener"
         id="SampleListener"
      />
 </extension>
```

2. Implement a `RichDocumentViewListener` class:

```
public class SampleListener implements RichDocumentViewListener {
       public void handleEvent(RichDocumentViewEvent event) {
           System.out.println(event.getSource());
       }
}
```

   In this example, the `getSource()` event returns the `RichDocumentView` instance which fires the event.

Package

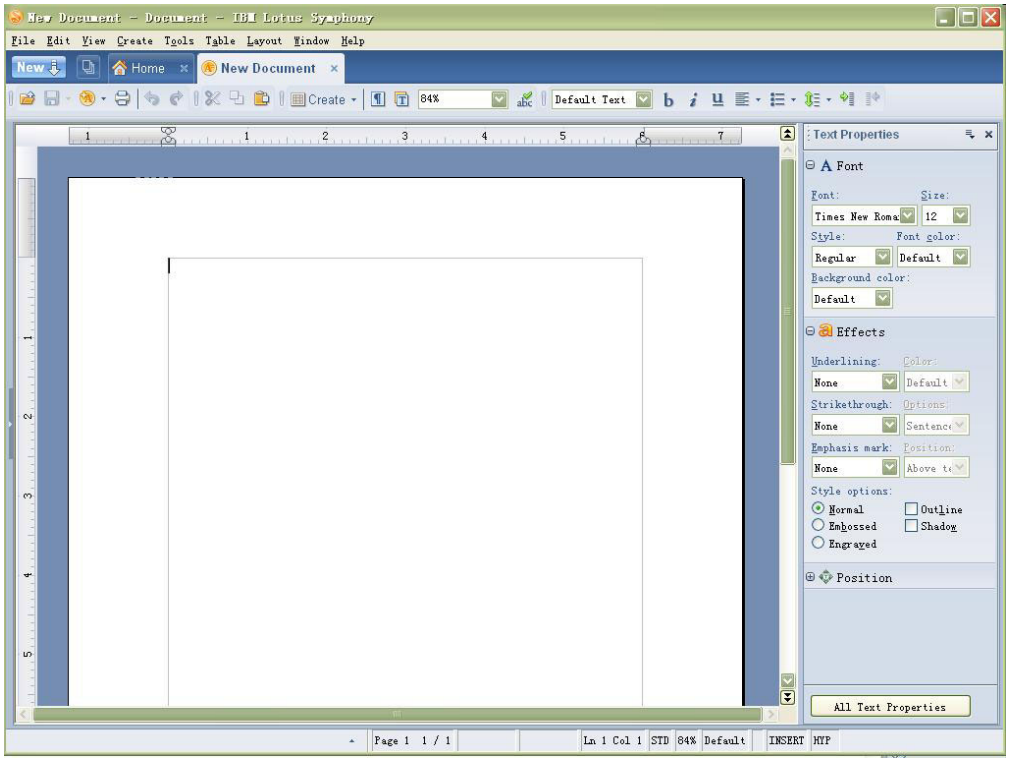
com.ibm.productivity.tools.ui.views.

See also

JavaDoc in Lotus Symphony toolkit.

Example

Typically, when opening or loading a document, the document is opened in a new tab, which depends on the windows and theme settings, in preference page. When closing a document, the tab is closed.



3.3 RichDocumentView

The RichDocumentView provides an interface for all Lotus Symphony view instances and defines common functions on a Lotus Symphony view. The view usually maps to an `Eclipse ViewPart` internally. New user interface items binding to the `ViewPart` are configurable through this interface, like the menu, toolbar, properties side bar and status bar.

Accessing Existing RichDocumentView Instances

You can get or create a `RichDocumentView` instance through `RichDocumentViewFactory` first, then use the APIs defined in **RichDocumentView** to perform the following tasks:

- Open another file in the view.
- Close the document in the view.
- Save the document in the view to another file.
- Add or remove a listener.
- Get the UNO model of the current document.

Using DefaultRichDocumentView Directly

In addition to the `RichDocumentView` interface, a default implementation named `DefaultRichDocumentView` is also provided. The `DefaultRichDocumentView` is an instance of `Eclipse ViewPart` and `RichDocumentView`. You can write a new perspective that aggregates several `Eclipse ViewParts` into one page.

Extending a New View

You can extend the default implementation to define your own view.

The following example code demonstrates how to reuse the `DefaultRichDocumentView`. The sample code implements a `WriterView` which creates a writer document in the `ViewPart`. The ViewPart can be integrated into an Eclipse perspective or displayed by an `IWorkbenchPage`. Refer to Eclipse and Lotus Expeditor programming instructions about how to use it. The complete sample code is also provided in the Lotus Symphony toolkit samples:

```
public class WriterView extends DefaultRichDocumentView {

  /**
   * The constructor.
   */
  public WriterView() {
      super();
    }

  public void createPartControl(Composite parent) {
      super.createPartControl(parent);
      createWriter();
  }

  private void createWriter()
  {
      NewOperation operation =
OperationFactory.createNewOperation(RichDocumentType.DOCUMENT_TYPE );

      operation.execute(this);
  }
}
```

Operations on Rich Documents

The following code example demonstrates how to load a rich document in the rich document view. The `WriterView` is created as above. There are also `SaveOperation`, `SaveAsOperation`, and `CloseOperation` interface provided in the Lotus Symphony Javadoc API. The usages are similar to `LoadOperation`; refer to the Javadoc API for more details:

```
private void loadDocument(){
     LoadOperation operation =
         OperationFactory.createLoadOperation("c:\\text.odt", false);
     this.executeOperation(operation);
 }
```

Monitoring Operations

The following code example demonstrates how to detect that a document is loaded into the rich document view. The `WriterView` is created as above. The example code demonstrates how to add an operation listener into the `ViewPart` when the

ViewPart is created. When a load operation is issued, the monitor is called. The OperationListener is applicable to all default operations and is documented in the Lotus Symphony Javadoc API:

```
public class WriterView extends DefaultRichDocumentView {

      /**
       * The constructor.
       */
       public WriterView() {
          super();
       }
       public void createPartControl(Composite parent) {
            super.createPartControl(parent);
            monitorLoading();
       }

       private void monitorLoading() {
           OperationListener listener = new OperationListener(){

            public void afterExecute(Operation operation, RichDocumentView view) {

                if( operation instanceof LoadOperation ){
                       System.out.println( "document is loaded:"
                               + ( (LoadOperation)operation ).getFileName());
                       Object document = (( LoadOperation )operation).getUNOModel();
                       afterLoading( document);
                }
            }
            public void beforeExecute(Operation operation, RichDocumentView view){
                   if( operation instanceof LoadOperation )
                       System.out.println( "document is about to be loaded:"
                               + ( (LoadOperation)operation ).getFileName());

            }

          };
     this.addOperationListener( listener );
    }
}
```

Chapter 4. Using the UNO API to Access a Document Model

Lotus Symphony Java API is only responsible for managing the Eclipse-based Lotus Symphony view. If you want to access and modify content within the document, use the UNO API, which is inherited from OpenOffice.org.

Accessing the Document Model

In Lotus Symphony, you can use the following code to get the UNO model of the current document:

```
RichDocumentView view = ….;
Object obj = view.getUNOModel();
XModel model = ( XModel )UnoRuntime.queryInterface( XModel.class, obj );
```

After you have the `XModel` object, you can access all UNO APIs within the model. For example, if you want to detect the document type, you can use this example code:

```
public String detectDocumentType(XModel model) {
        String type = "";
        XServiceInfo info = ( XServiceInfo )UnoRuntime.queryInterface
            (XServiceInfo.class, model);
        if( info.supportsService("com.sun.star.text.TextDocument") )
            type = "Writer";
        else if( info.supportsService("com.sun.star.sheet.SpreadsheetDocument") )
            type = "Spreadsheet";
       else if( info.supportsService("com.sun.star.presentation.PresentationDocument")
            )
            type = "Presentation";

    return type;
}
```

Using the Writer Document Model

If the document is a writer document, all UNO APIs can be used with Java. With the UNO API, you can almost do anything you want in the document, for example:

- Navigating objects like text, paragraph, or tables in document.
- Inserting or removing objects.
- Getting or setting the property of objects.
- Getting or setting selections.
- Accessing and modifying document metadata.

Some typical use cases are described in following sections. For more details, refer to the `OpenOffice.org` SDK Developer's Guide.

Setting the Whole Text of a Document

Use the following sample code to change the whole text of a document:

```
public void setWholeTextofDocument( XModel model ){
            XTextDocument xdoc = ( XTextDocument ) UnoRuntime.queryInterface(
                    XTextDocument.class, model);
            XText xdocText= xdoc.getText();
            //simple text insertion
            xdocText.setString ( "The whole text of this document.\n" +
                    "The second line...");
}
```

Inserting a Table in a Document

Use the following sample code to insert a table into the document:

```
public void insertTable( XModel model ) {
            XMultiServiceFactory xDocFactory = (XMultiServiceFactory)
            UnoRuntime.queryInterface(XMultiServiceFactory.class,
                    model);
            XTextDocument xdoc = ( XTextDocument ) UnoRuntime.queryInterface(
                    XTextDocument.class, model);
            XText xdocText= xdoc.getText();

        // Create a new table from the document's factory
            try {
                XTextTable xTable = (XTextTable) UnoRuntime.queryInterface(
                            XTextTable.class, xDocFactory .createInstance(
                                    "com.sun.star.text.TextTable" ) );
                 // Specify that we want the table to have 4 rows and 4 columns
                 xTable.initialize( 4, 4 );

                 // Insert the table into the document
                 xdocText.insertTextContent( xdocText.getStart(), xTable, false);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
}
```

Setting Text in the Current Cursor

Use the following sample code to set content into the current cursor:

```
public void setSelection( XModel model, String content ){
       //the controller of the model
        XController xController = model.getCurrentController();
        // Query TextViewCursor
        XTextViewCursorSupplier xViewCursorSupplier =
            (XTextViewCursorSupplier)UnoRuntime.queryInterface(
                XTextViewCursorSupplier.class, xController);
        //get the view cursor
        XTextViewCursor viewCursor = xViewCursorSupplier.getViewCursor();
        //set the content to the view cursor
        viewCursor.setString( content );

}
```

Using the Spreadsheet Document Model

If the document is a spreadsheet document, all UNO APIs for spreadsheet documents can be used with Java. With the UNO API, you can almost do anything you want in the document, for example:

- Accessing sheets, cells, and cell ranges in the document.
- Modifying content of sheets, cells, or cell ranges.
- Creating charts.
- Using functions.

A typical use case is described in the following section. For more details, refer to the Spreadsheet sample in the Lotus Symphony toolkit samples and `OpenOffice.org` SDK Developer's Guide.

Setting the Content of a Cell

Use the following example code to set the content in column 2 row 3 in the first sheet:

```
public void setCellText( XModel model, String content ){
       //query the sheet document
       XSpreadsheetDocument sheetDocument = ( XSpreadsheetDocument )
             UnoRuntime.queryInterface( XSpreadsheetDocument.class, model );
        XSpreadsheets xSheets = sheetDocument.getSheets();
        XSpreadsheet xSheet = null;
        try {
           com.sun.star.container.XIndexAccess xSheetsIA =
          (com.sun.star.container.XIndexAccess)
               UnoRuntime.queryInterface( com.sun.star.container.XIndexAccess.class,
             xSheets );
            //get the first sheet in the document
            xSheet = (com.sun.star.sheet.XSpreadsheet) UnoRuntime.queryInterface(
                com.sun.star.sheet.XSpreadsheet.class, xSheetsIA.getByIndex( 0 ));

            com.sun.star.table.XCell xCell = null;
            // --- Get cell of column 2 row 3- (column, row) ---
            xCell = xSheet.getCellByPosition( 1, 2 );
            com.sun.star.text.XText xText = (com.sun.star.text.XText)
            UnoRuntime.queryInterface( com.sun.star.text.XText.class, xCell );
            xText.setString( content );
        }
        catch (Exception ex){
            ex.printStackTrace();
        }
}
```

Using the Presentation Document Model

If the document is a presentation document, all UNO APIs for presentation document can be used with Java. With the UNO API, you can almost do anything you want in the document, for example:

- Accessing and modifying pages and shapes in the document.
- Inserting and removing pages or shapes in the document.
- Playing the presentation.

For more details, refer to `OpenOffice.org` SDK Developer's Guide.

Chapter 5. Packaging and Deploying Your Plug-Ins

After you have completed plug-in development, run your code in an installed Lotus Symphony product environment, or distribute your plug-ins to customer's in a Lotus Symphony environment. To achieve it your application needs to be packaged and deployed.

The content below in this chapter illustrates how to package and deploy an application to Lotus Symphony based on the samples which can be found in the Lotus Symphony toolkit.

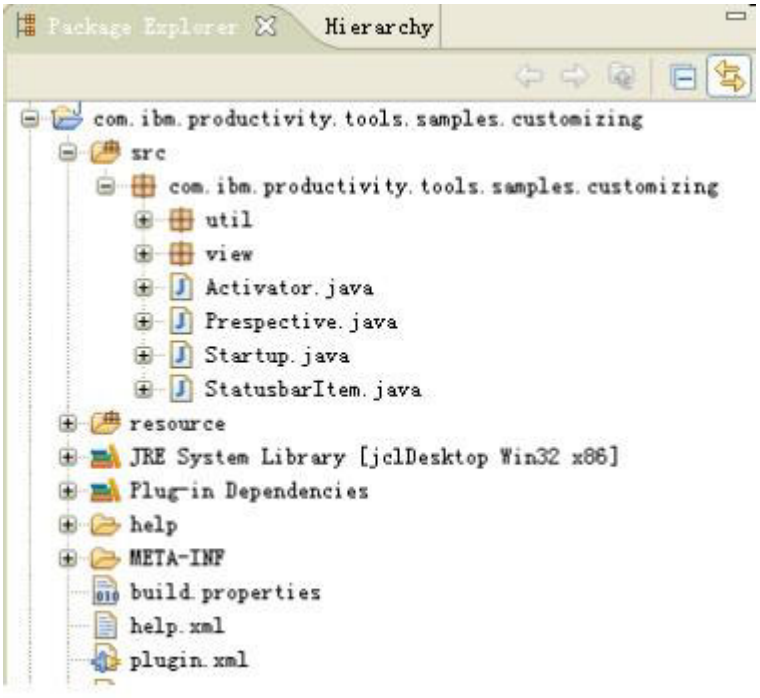
Use following steps to install your custom application into Lotus Symphony:

1. Prepare your custom plug-in for deployment.
2. Create a feature and an Eclipse location update site.
3. Install a custom Lotus Symphony application.
4. Configuration your application.

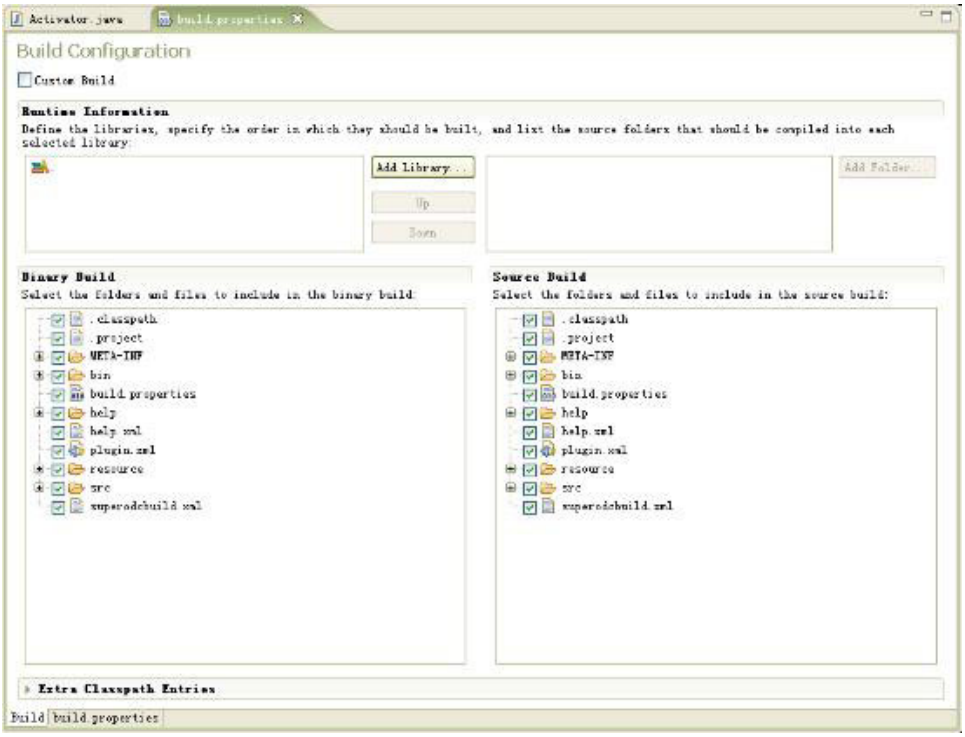
5.1 Prepare Custom Plug-in for Deployment

The following steps prepare the plug-in for deployment:

1. Open Eclipse. Be sure to use the same workspace where you created your plug-ins.



2. Expand your plug-in in the **Package Explorer** perspective.
3. Double-click the `Build.Properties` file.
4. Select the portions of the plug-in that you want to include in the build. For the purposes of this example, all are chosen; however, this might not be necessary in your scenario.



5. Click **File > Save**.

5.2 Create a Feature and an Eclipse Location Update Site

Updates to the client platform are provided in the form of features. Features can contain other features, or a set of related plug-ins. The Update Manager component of the client platform handles the installation of the features, and a user interface is provided to manage the installed features.

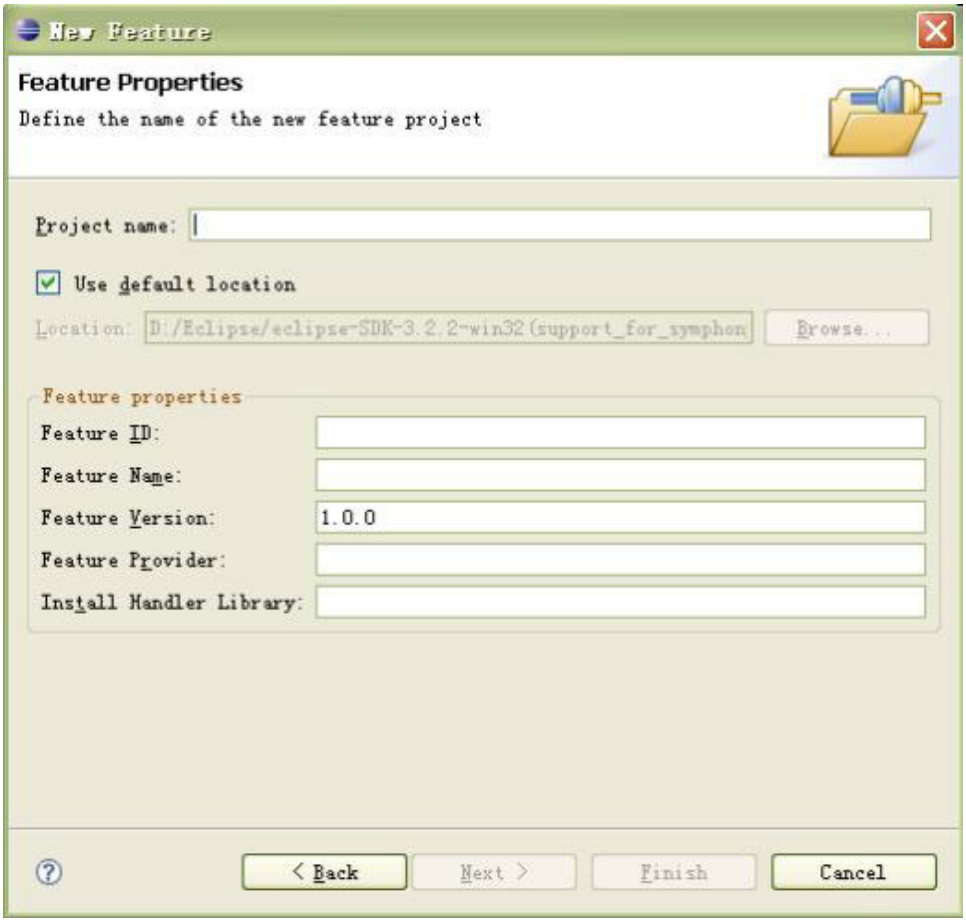
5.2.1 Creating a Feature

A feature contains a manifest that provides basic information about the feature and its contents, including plug-ins and fragments. A feature is deployed and delivered in the form of a JAR file.

Now that your plug-in is ready to be deployed, it needs to be packaged in a manner that is recognized by the Eclipse update Manager. The Eclipse update manager is an Eclipse tool that manages versions and deployment of plug-ins and fragments.

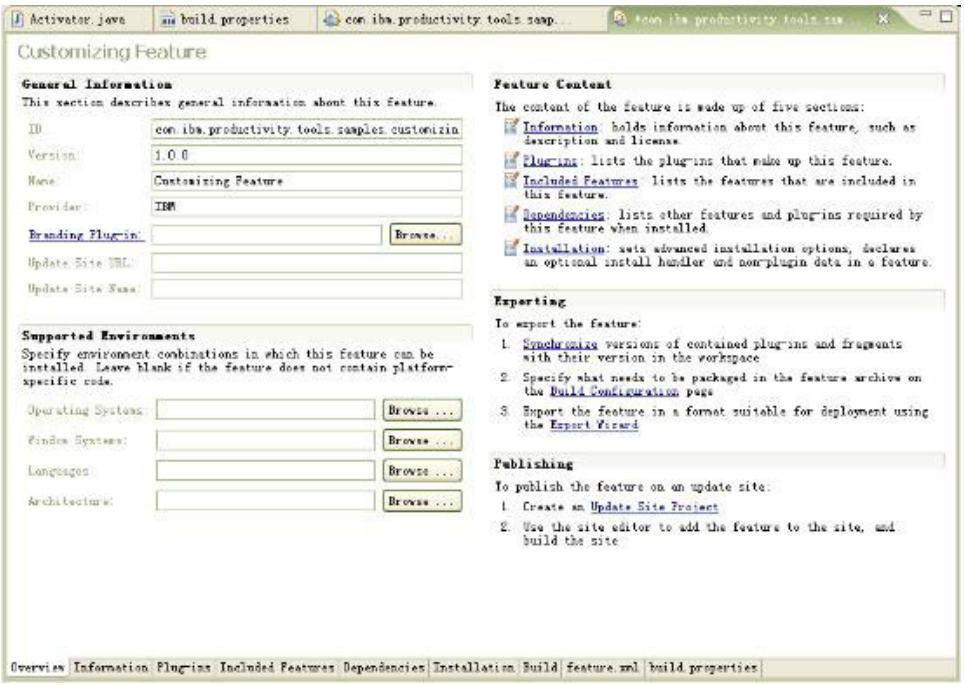
Prior to creating a feature, you should have the plug-ins and fragments that will be contained within the feature

1. Make sure that your plug-in is opened in the workspace you created.
2. From your workspace, select **File > New > Project > Plug-in Development > Feature Project**, as shown in the following figure:



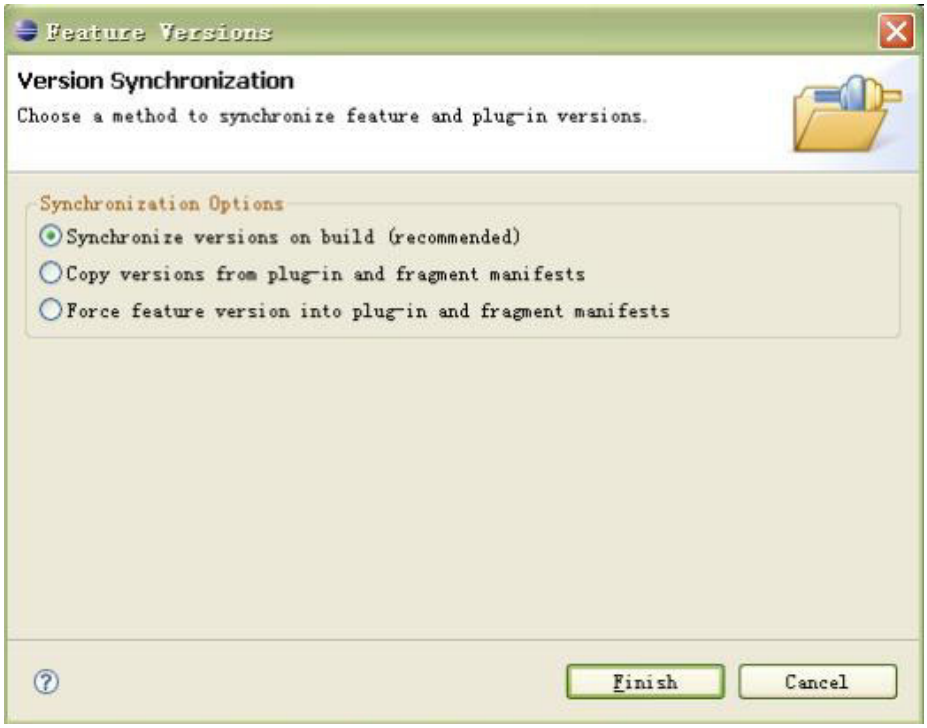
3. On the New Properties page, enter the **Feature ID**, **Feature Name**, and **Feature Version**. The **Feature Provider** and **Install Handler Library** are optional.

4. Click **Next**.
5. On the **Referenced Plug-ins and Fragments** page, select the plug-in that you are making ready for deployment from the list, and then click **Finish**. The wizard now creates your feature package and opens the feature on the Overview tab of the `feature.xml` file. You can always come back to this view (known as the `feature manifest editor`) by double clicking the `feature.xml` file.



6. There are many options in this view. Change the following fields if necessary:
   a. In the **Branding Plug-in** field, click **Browse** field.
   b. Select the plug-in that you want to deploy and click **OK**.
   c. In the **Update Site URL** field, enter the Eclipse update site URL.
   d. In the **Update Site Name** field, enter the site name.
   e. In the **Supported Environments** section, enter operating systems, platform, and language specifications, if these are required by your plug-in. For our example, this section is not necessary.

      Note: This information is used to specify the site that is used to load your feature using Eclipse Update Manager. When Update Manager looks for updates, it will look for sites defined in your update site URL. If you have not created an Eclipse update site yet, you can change this setting later.
7. Click the Information tab.
   a. The **Feature Information**, **Copyright**, **License** and **Sites to Visit** tabs are displayed. Feature information is displayed to the user by the update manager when the feature is selected.
   b. For each of these tabs, you can either enter a URL, if sites already exist, or you can enter the information in the text area for each.
   c. In the **Optional URL** field, enter a URL and name for any other relevant update sites that you have.
8. Click the **Plug-in** tab.
   a. Confirm that your plug-in is listed in the Plug-ins and Fragments window. If it is not, click **Add** and select the plug-in that you want to include, and then click **OK**.
   b. Click **Version**.
   c. Select **Synchronize Versions on Build (recommended)**, as shown in the following figure, and then click **Finish**. This step synchronizs your feature version and plug-in version.



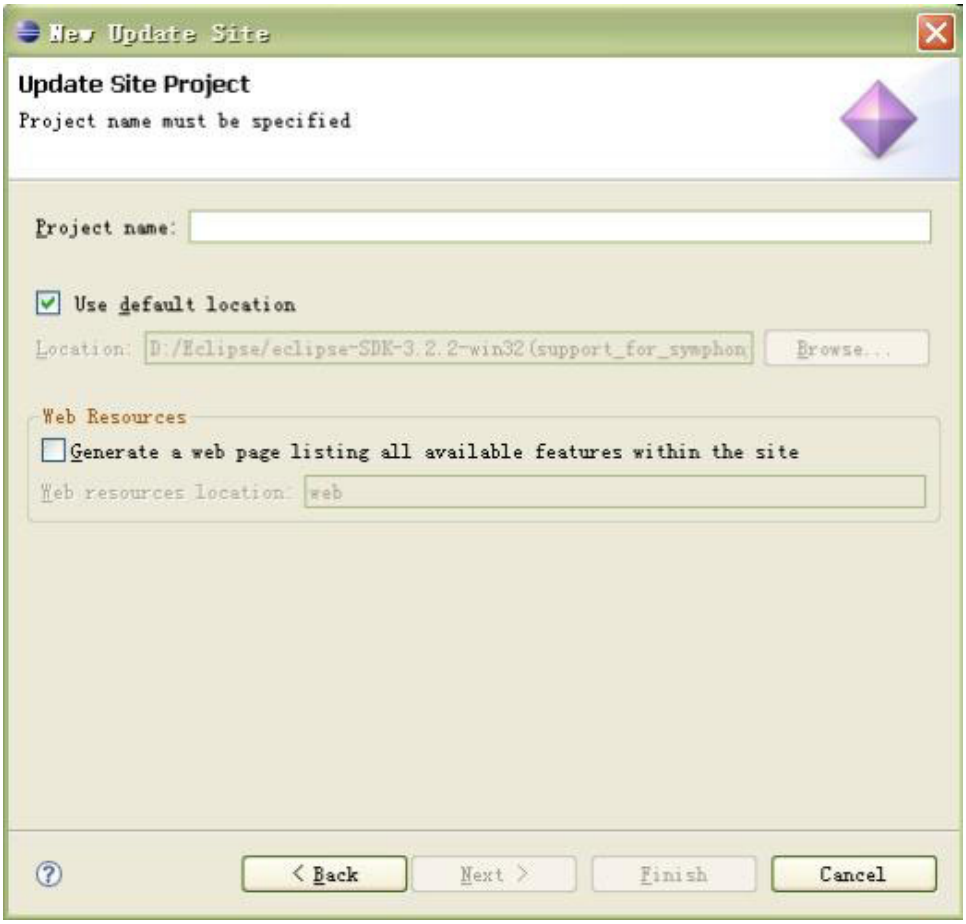
9. Your feature and plug-in are now ready to deploy.

5.2.2 Creating an update site

An update site is the key mechanism to enable installation of the application, which includes the features and plug-ins to be deployed. For more information on update sites, including how to create one, see the **Plug-in Development Environment Guide > Getting Started > Update Sites** section of the PDE Guide.

To create an update site, complete the following steps:

1. Open Eclipse. Be sure to open the workspace where you created your plug-in and feature.
2. Select **File > New > Project > Plug-in Development > Update Site Project**.



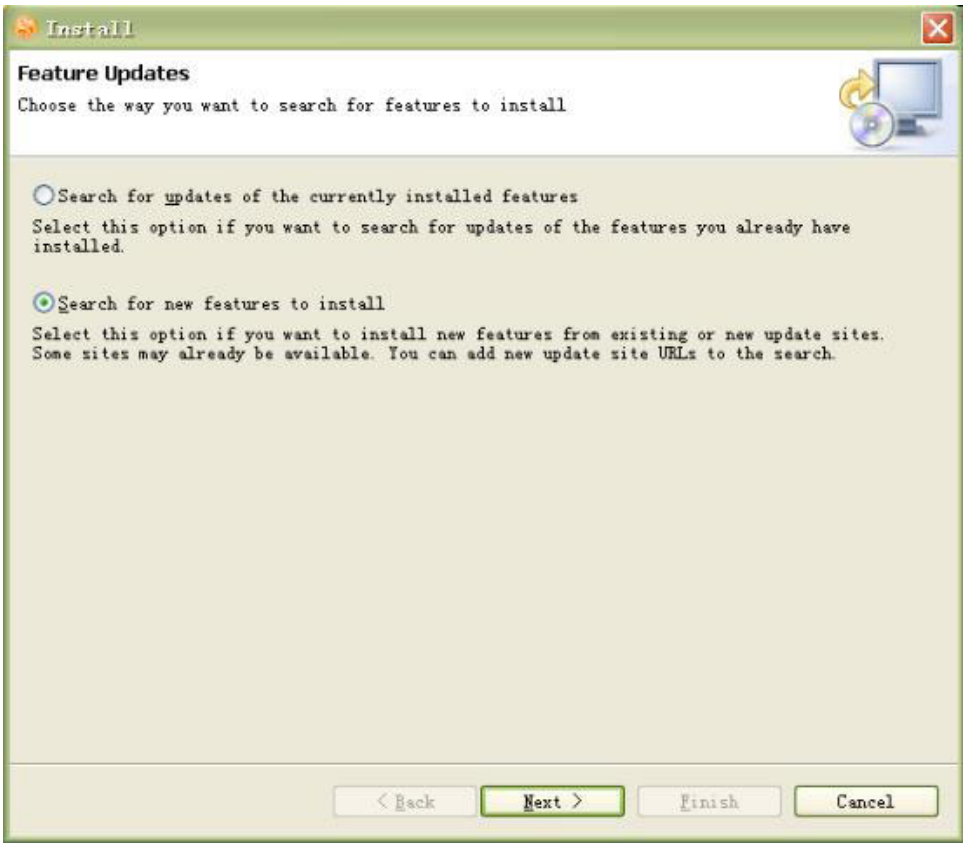
3. The New Update Site wizard has only one page:
   a. Enter a Project name. You should enter the plug-in name and append another word to denote that it is an update site project.
   b. Select **Use the default location**.
   c. Click **Finish**. The wizard creates your update site within your Eclipse workspace.
4. To add your feature(s):
   a. Double-click the **site.xml** file located in the Package Explorer frame. This step opens your site manifest editor in the editor frame (center frame).
   b. To add your new feature, click **Add Feature**. If you are adding more than one feature or plug-in or plan to in the future, you can choose to organize them by category.
   c. Select the feature that you are including in this update site. You can select more than one by holding down the Ctrl key. When you are finished selecting, click **OK**.
5. Click the **Build All** button. This step adds the `/Features` and `/Plug-ins` directories to the `Site` project and populates them with JAR files containing your feature and plug-in files. This step builds your update site locally.
6. Export this update site project to the file system, for example `D:\customizing`.

5.3 Install a Custom Lotus Symphony Application

In this option, customers can deploy applications to an existing Lotus Symphony client in a standard update site installation.

1. Launch Lotus Symphony and select **File > Applications > Install**.

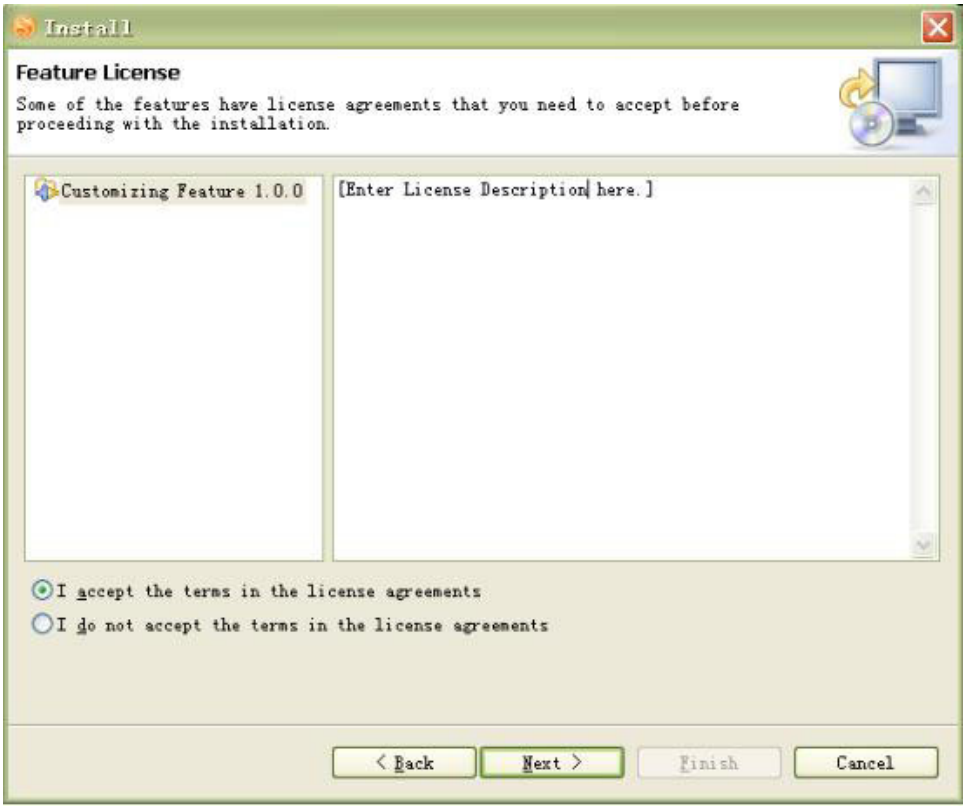
2. Select **Search for new features to install** and click **Next**.



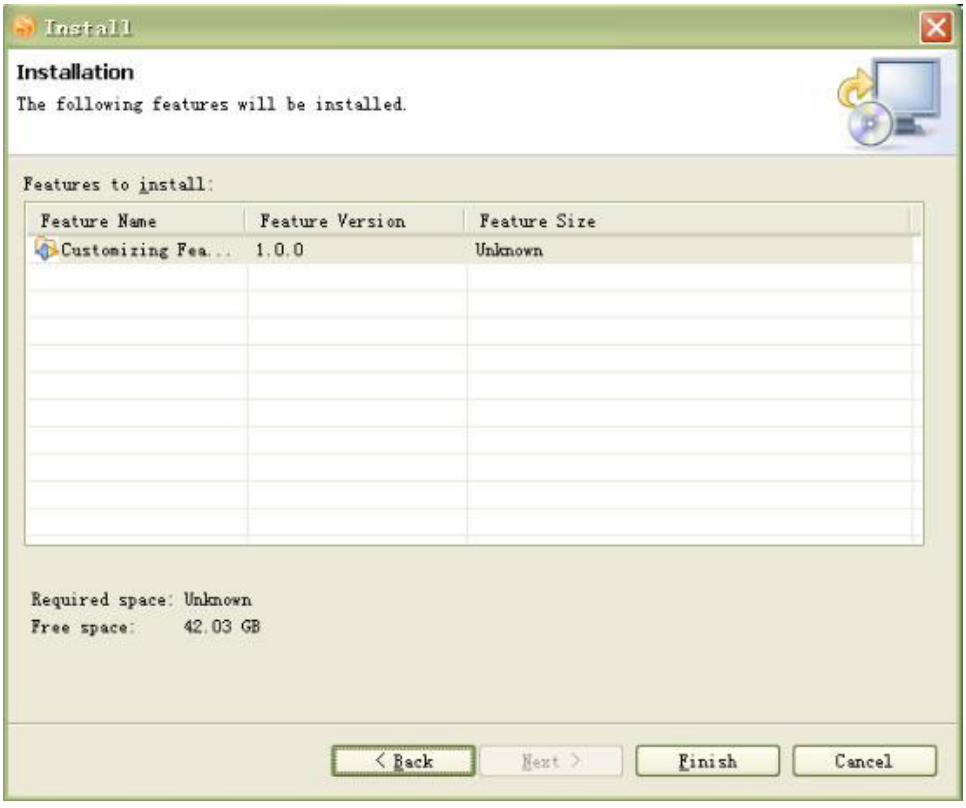
3. Click **Add Folder Location** and select the update site project from the local file system, then click **Finish** button.

4. In Updates window, select the update site, and then click **Next**.

5. To accept **the terms in the license agreements** in the **Install** window, and then click **Next**.



6. Click **Finish** to install the imported feature.



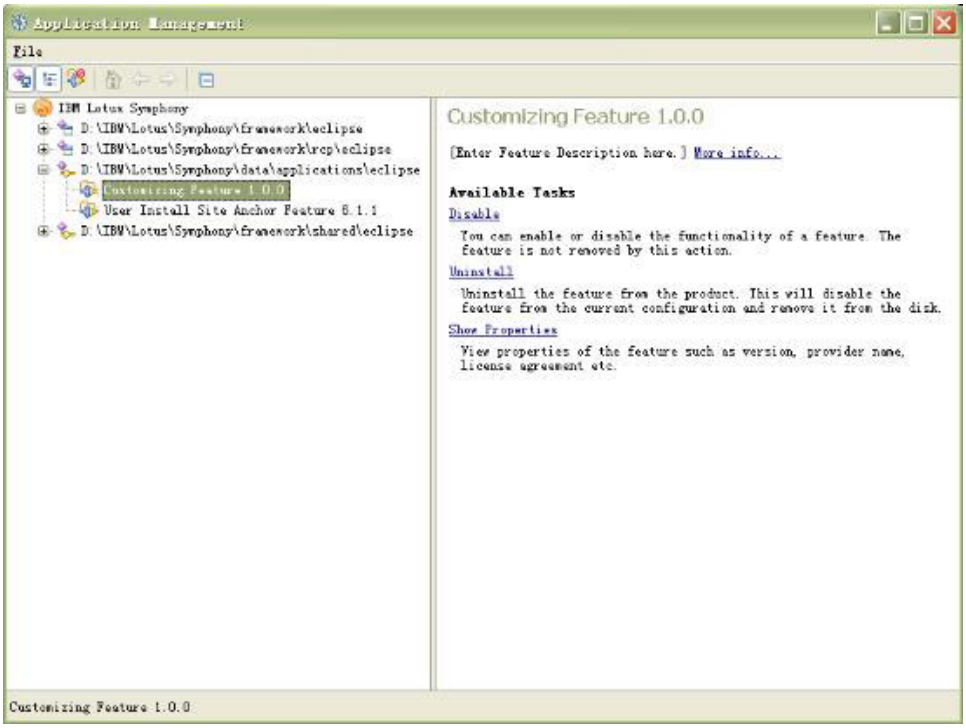
7. After you have finished the installation, **restart** Lotus Symphony to see your application and verify that it was successfully installed.

5.4 Disable or Enable Custom Lotus Symphony Applications

You can view and change the status for any plug-ins that you have installed in Lotus Symphony. To disable custom application plug-ins, do the following steps:

1. Click **File > Application > Application Management**.

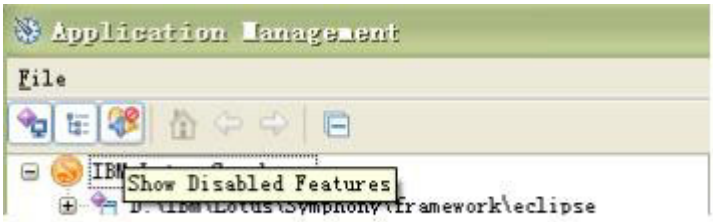
2. In the navigator, click `<Symphony Install Home>\data\ applications\eclipse`, and find the custom application that you want to view and make changes to.



3. Click **Disable** in right pane and accept the restart operation, to disable this application. The application is not removed by this action.

To enable a disabled application, do the following steps:

1. Click **File > Application > Application Management**.
2. Make sure that select the **Show Disable Features** item is selected.



3. Click <Symphony Install Home>\data\applications\eclipse, to find the custom application that you want to view and make changes to.
4. Click **Enable** in right pane and accept the restart operation, to enable the selected application.

5.5 Uninstall Custom Lotus Symphony Application

1. Click **File > Application > Application Management.**
2. Click <Symphony Install Home>\data\applications\eclipse, to find the custom application that you want to view and make changes to.
3. Click **Uninstall** in right pane and accept the restart operation, to uninstall the selected application.

Part 5. Lotus Expeditor and Uno Programming

Chapter 1. Developing Lotus Expeditor Applications

This information focuses on how to extend Lotus Symphony with Lotus Expeditor and Eclipse extension points. After you understand the rich client application model in Lotus Expeditor, you can build rich client applications based on the Lotus Symphony APIs. A large variety of applications can be built with this application model, for example, the Lotus Notes 8 client. Lotus Notes 8 is based on Lotus Expeditor platform and the Lotus Symphony editor is integrated as an office component.

The composite application model is another programming pattern provided by Lotus Expeditor. In this model, multiple applications cooperate by using inter-component communications. With this approach, you can aggregate several loosely coupled views into one perspective. The property broker is used to communicate among different views. The Lotus Symphony editor supports the composite application programming model in Lotus Notes.

1.1. Lotus Expeditor toolkit documentation

To develop plug-ins, use the Lotus Expeditor toolkit as development environment. You can find documentations about Lotus Expeditor from **Help> Help content> Developing applications for Lotus Expeditor**.

Note: The help content is available only after you install the Lotus Expeditor toolkit into the Eclipse development environment.

1.2. Debugging and testing applications

You can use the Lotus Expeditor toolkit's Client Services Launcher to run and debug applications. The Client Services Launcher is very similar to Eclipse plug-in development tools.

For more details refer to the Lotus Expeditor Application Developer's Guide, or you can find the information from Eclipse at **Help > Help content > Developing applications for Lotus Expeditor > Debugging and testing applications**.

1.3. Packaging and deployment for local testing

You might be required to verify your applications in a locally installed instance of Lotus Symphony during the development phase. You will need to export your plug-ins to the local file system, and copy them into your Lotus Symphony installation location.

For details, refer to the Lotus Symphony Application Developers Guide, or you can find the information from Eclipse at **Help > Help content > Developing applications for Lotus Expeditor > Packaging and deploying applications > Deploying projects for local testing**.

1.4. Securing applications and data

Lotus Expeditor is a secure platform that protects your application data. This capability is provided in the com.ibm.rcp.accounts.feature feature, which is known as the account framework in Lotus Expeditor. It is available in a Lotus Symphony package. The account framework provides a mechanism for you to manage account information.

For details, refer to the Lotus Expeditor Application Developers Guide, or you can find the information from Eclipse at **Help > Help content > Developing applications for Lotus Expeditor > Securing applications and data**.

Chapter 2. UNO Programming

2.1 Getting the global service factory

The `com.sun.star.lang.ServiceManager` factory is the main factory in every UNO application. It is the entrance point to the UNO world of Lotus Symphony. The following tasks can be performed from the service manager:

- Instantiate services by their service name
- Enumerate all implementations of a certain service
- Add or remove factories for a certain service at runtime

The service manager is passed to every UNO component during instantiation.

To get the ServiceManager, use the following sample code:

```
public static XMultiServiceFactory getServiceFactory() {
            XConnection conn = ProductivityToolsUtil.getUNOConnection();
            XBridge mBridge;
            try {
                XComponentContext _ctx = com.sun.star.comp.helper.Bootstrap
                        .createInitialComponentContext(null);
                Object x = _ctx.getServiceManager().createInstanceWithContext(
                        "com.sun.star.bridge.BridgeFactory", _ctx);
                XBridgeFactory xBridgeFactory = (XBridgeFactory) UnoRuntime
                        .queryInterface(XBridgeFactory.class, x);

                // create a nameless bridge with no instance provider
                try {
                    mBridge = xBridgeFactory.createBridge("SODC_Bridge", "urp",
                            conn, null);
                } catch (BridgeExistsException beexp) {
                    mBridge = xBridgeFactory.getBridge("SODC_Bridge");
                }
                // get the remote instance
                x = mBridge.getInstance("StarOffice.ServiceManager");

                // Did the remote server export this object?
                if (null == x)
                    return null;
```

```
        // Query the initial object for its main factory interface

        XMultiComponentFactory xOfficeMultiComponentFactory =
        (XMultiComponentFactory) UnoRuntime
               .queryInterface(XMultiComponentFactory.class, x);

        // Retrieve the component context
        // Query on the XPropertySet interface.

        XPropertySet xProperySet = (XPropertySet) UnoRuntime
                .queryInterface(XPropertySet.class,
                        xOfficeMultiComponentFactory);

        // Get the default context from the editor service.

        Object oDefaultContext = null;
        try {
            oDefaultContext = xProperySet
                    .getPropertyValue("DefaultContext");
        } catch (UnknownPropertyException e) {
            e.printStackTrace();
        } catch (WrappedTargetException e) {
            e.printStackTrace();
        }
        if (oDefaultContext == null)
            return null;
        XComponentContext context = (XComponentContext) UnoRuntime
            .queryInterface(XComponentContext.class, oDefaultContext);
        return (XMultiServiceFactory) UnoRuntime.queryInterface(
            XMultiServiceFactory.class, context.getServiceManager());
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
}
```

2.2 Using the import and export functions

The import and export functions are common in all three applications inside Lotus Symphony. For different kinds of document types, there can be a different UNO interfaces to support loading and saving operations.

The following sections detail the common interface used in all three applications and the specific document types that can have special interface support.

Loading new or existing components

The desktop can load new and existing components from a URL. The com.sun.star.frame.XComponentLoader interface has one method to load and instantiate components from a URL into a frame:

```
com.sun.star.lang.XComponent loadComponentFromURL([in] string aURL, [in] string
            aTargetFrameName, [in] long nSearchFlags, [in] sequence<
            com.sun.star.beans.PropertyValue > aArgs );
```

The URL is used to describe which resource should be loaded and in what sequence to load the arguments. For the target frame, pass "_blank" and set the search flags to 0 to open a new frame. In most cases you do not want to reuse an existing frame.

The URL can be of these types: file:, http:, ftp:, or private:. For new documents, a special URL scheme is used. The scheme is private:, followed by factory as the host name. The resource is `swriter` for word processor documents. For example, a new word processor document, uses `private:factory/swriter`.

Storing documents

Documents are stored through their interface `com.sun.star.frame.XStorable`.

```
void storeAsURL( [in] string aURL, sequence< com.sun.star.beans.PropertyValue >
            aArgs)
void storeToURL( [in] string aURL, sequence< com.sun.star.beans.PropertyValue >
            aArgs)
```

The method storeAsUrl() is the exact representation of a **File > Save As** operation, that is, it changes the current document location. In contrast, the method storeToUrl() stores a copy to a new location, but leaves the current document URL untouched.

For exporting purposes, a filter name can be passed to `storeAsURL()` and `storeToURL()` that triggers an export operation to other file formats.

```
/** Store a document, using the MS Word 97/2000/XP Filter */
protected void storeDocComponent(XComponent xDoc, String storeUrl) throws
            java.lang.Exception {
XStorable xStorable = (XStorable)UnoRuntime.queryInterface(XStorable.class, xDoc);
PropertyValue[] storeProps = new PropertyValue[1];
storeProps[0] = new PropertyValue();
storeProps[0].Name = "FilterName";
storeProps[0].Value = "MS Word 97";
xStorable.storeAsURL(storeUrl, storeProps);
}
```

Exporting documents and drawing objects

Writer documents and Spreadsheet documents can be exported as HTML format files. Presentation documents can export drawing objects as graphics through the com.sun.star.drawing.GraphicExportFilter interface. After getting a

GraphicExportFilter from the ServiceManager, use its XExporter interface to inform the filter which page, shape, or shape collection to export.

Functions in this interface include:

```
void setSourceDocument ( [in] com.sun.star.lang.XComponent xDoc)
boolean filter( [in] sequence< com.sun.star.beans.PropertyValue > aDescriptor)
void cancel()
```

The aDescriptor parameter in the filter function holds all the necessary information about the document, such as document title, author, file name, URL, and version. All such properties are organized in a com.sun.star.beans.PropertyValue [] array.

Followings are some sample code for exporting function, exporting ODT and ODS files to HTML; ODP to JPEG image files:

1. Get a file's XComponent from a file path.

   When exporting a document to whatever format, first get this file's `com.sun.star.lang.XComponent` object. The following sample code shows how to get the `ServiceManager` as mentioned above:

```
/**
     * get document Xcomponent object.
     *
     * @param   sourceFile  file path
     */
public static XComponent getXComponent(String sourceFile) {

    XMultiServiceFactory xServiceFactory = getServiceFactory();
    XComponent component = null;

    try {
        Object object = xServiceFactory
                .createInstance("com.sun.star.frame.Desktop");

        XComponentLoader loader = (XComponentLoader) UnoRuntime
                .queryInterface(XComponentLoader.class, object);
        PropertyValue[] aArgs = new PropertyValue[1];
        aArgs[0] = new PropertyValue();
        aArgs[0].Name = "Hidden";
        aArgs[0].Value = new Boolean(false);

        String sourceURL = new String("file:///")
                + sourceFile.replace('\\', '/');

        object = loader.loadComponentFromURL(sourceURL, "_blank",
                FrameSearchFlag.CREATE, aArgs);
        component = (XComponent) UnoRuntime.queryInterface(
                XComponent.class, object);

    } catch (Exception e) {
        e.printStackTrace();
    }
        return component;
}
```

2. Convert Lotus Symphony documents (odt, ods) file to a HTML file.

```
/**
     * convert given document format into HTML format.
     *
     * @param   xDocument   document which should be exported
     * @param   filepath    target path for converted document
     */
Public static void convertToHTML(XComponent xDocument, String filepath){
      try {
              XServiceInfo xInfo = (XServiceInfo)UnoRuntime.queryInterface(
                  XServiceInfo.class, xDocument);

             if(xInfo!=null) {
                // Find out possible filter name.
                String sFilter = null;
                if(xInfo.supportsService("com.sun.star.text.TextDocument"))
                    sFilter = new String("HTML (StarWriter)");

                else if(xInfo.supportsService("com.sun.star.text.WebDocument"))
                    sFilter = new String("HTML");

             else if (xInfo.supportsService("com.sun.star.sheet.SpreadsheetDocument"))
                    sFilter = new String("HTML (StarCalc)");

                // Check for existing state of this filter.
                if(sFilter!=null){
                    XMultiServiceFactory xSMGR = ServiceFactory.getServiceFactory();

                    XNameAccess xFilterContainer = (XNameAccess)
UnoRuntime.queryInterface(
                          XNameAccess.class,
                          xSMGR.createInstance
("com.sun.star.document.FilterFactory"));

                    if(xFilterContainer.hasByName(sFilter)==false)
                        sFilter=null;
                }
```

```
             // Use this filter for export.
            if(sFilter!=null) {
                 PropertyValue[] lProperties = new PropertyValue[2];
                 lProperties[0] = new PropertyValue();
                 lProperties[0].Name = "FilterName";
                 lProperties[0].Value = sFilter;
                 lProperties[1] = new PropertyValue();
                 lProperties[1].Name = "Overwrite";
                 lProperties[1].Value = Boolean.TRUE;

                 XStorable xStore = (XStorable)UnoRuntime.queryInterface
              (XStorable.class, xDocument);
                 String sourceURL = new String("file:///") + filepath.replace('\\',
'/');
                 xStore.storeAsURL(sourceURL,lProperties);
             }
          }
       } catch(Exception ex){
          ex.printStackTrace();
       }
}
```

3. Convert current presentation document page as a JPEG image.

```
/**
     * convert given presentation page into a JEPG iamge.
     *
     * @param   xDocument       document which should be exported
     * @param   nPageIndex      the page's index
     * @param   filepath        target path for converted document
     */

public static void exportJPEG(XComponent xComponent, int nPageIndex,
                        String filepath) {
        try {
                XMultiServiceFactory xServiceFactory = ServiceFactory
                     .getServiceFactory();
                Object GraphicExportFilter = xServiceFactory
                     .createInstance("com.sun.star.drawing.GraphicExportFilter;
                XExporter xExporter = (XExporter) UnoRuntime.queryInterface(
                     XExporter.class, GraphicExportFilter);

                PropertyValue aProps[] = new PropertyValue[2];
                aProps[0] = new PropertyValue();
                aProps[0].Name = "MediaType";
                aProps[0].Value = "image/jpeg";

/** some graphics e.g. the Windows Metafile does not have a Media Type, for this case
*aProps[0].Name = "FilterName";  it ispossible to set a FilterName aProps[0].Value =
"WMF";*/
```

```
                java.io.File destFile = new java.io.File(fileName);
                StringBuffer destUrl = new StringBuffer("file:///");
                destUrl.append(destFile.getCanonicalPath().replace('\\',
                  '/'));

                aProps[1] = new PropertyValue();
                aProps[1].Name = "URL";
                aProps[1].Value = destUrl.toString();// args[ 1 ];

                if (nPageIndex < getDrawPageCount(xComponent)&&nPageIndex >=0) {
                        XDrawPage xPage = getDrawPageByIndex(xComponent, nPageIndex);
                        XComponent xComp = (XComponent) UnoRuntime.queryInterface(
                        XComponent.class, xPage);
                        xExporter.setSourceDocument(xComp);
                        XFilter xFilter = (XFilter) UnoRuntime.queryInterface(
                          XFilter.class, xExporter);
                        xFilter.filter(aProps);
                }
        } catch (Exception ex) {
              ex.printStackTrace();
        }
}
```

If you need to specify the exported JPEG image size, add the size information to the filter's property. The code snippet is as following:

```
    PropertyValue aProps[] = new PropertyValue[3];

      aProps[0] = new PropertyValue();
      aProps[0].Name = "URL";
      aProps[0].Value = destUrl.toString();// args[ 1 ];

      aProps[1] = new PropertyValue();
      aProps[1].Name = "FilterName";
      aProps[1].Value = "JPG";

       PropertyValue aFilterData[] = new PropertyValue[2];
      aFilterData[0] = new PropertyValue();
      aFilterData[0].Name = "PixelWidth";
      aFilterData[0].Value = new Integer(width);
      aFilterData[1] = new PropertyValue();
      aFilterData[1].Name = "PixelHeight";
      aFilterData[1].Value = new Integer(heigth);

// use the FilterData hold the export image size infomation
      aProps[2] = new PropertyValue();
      aProps[2].Name = "FilterData";
      aProps[2].Value = aFilterData;
```

Using the print function

Lotus Symphony documents, spreadsheets and presentations all provide the print-related interface `com.sun.star.text.XPagePrintable`, and the print-related properties `com.sun.star.view.PrinterDescriptor` and `com.sun.star.view.PrintOptions`. Specifically, Lotus Symphony documents support printing multiple pages on one page by setting the property `com.sun.star.text.PagePrintSettings`. Lotus Symphony spreadsheets provide access to the addresses of all printable cell ranges by the interface `com.sun.star.sheet.XPrintAreas`. Lotus Symphony presentations have some specific properties to define if the notes and outline view should be printed by `com.sun.star.presentation.DocumentSettings`. For detailed information, refer to the OpenOffice.org SDK.

2.3 Text documents

In the Lotus Symphony Documents API, a text document is a document model that is responsible for managing text contents, through which you can understand how the basic data is organized and represented in the graphical user interface.

You have to work with the model directly, when you want to change it through the Lotus Symphony API to develop applications for your own usage. The model is similar with OpenOffice 1.1, which also has a controller object that is used to manipulate the visual representation of the document in the view areas instead of being used to change a document.

The model is different from the controller, and we discuss the parts of a text document model in the Lotus Symphony API and emphasize some differences between Lotus Symphony documents API and OpenOffice 1.1 Writer API. To the parts that are the same, we provide a reference to OpenOffice 1.1 development guide directly.

The text document model in the Lotus Symphony API has these major architectural areas that are the same as OpenOffice 1.1 API:

- Text (core content)
- Service manager (document internal)
- Draw page
- Text content suppliers (drawing objects)
- Text content suppliers (access content)
- Objects for styling and numbering (document wide)

The text is the core of the text document model. It consists of characters organized in paragraphs and other text contents.

The service manager of the document model is responsible for creating all text contents for the model, except for the paragraphs. And each document model has its own service manager, such as the spreadsheet document model and presentation document model. Almost all of the text contents in a text document can be retrieved from text content suppliers which are provided by the model, except the drawing shapes that can be found on the draw page.

The draw page is floating over the text and it is responsible for drawing contents. Drawing contents can affect the layout of the text around it, such as wrap types.

There are also services that are for document-wide text styles and structures. The style family suppliers are provided to customize document-wide paragraphs, characters, pages and numbering patterns, and suppliers for line and outline numbering.

For more ideas, refer to the Illustration 7.1 Text Document Model of the OpenOffice 1.1 Development Guide.

Word processing

The document model provides the `XTextDocument` interface to work with text through the method `getText()`. It returns a `com.sun.star.text.Text` service that handles text in Lotus Symphony documents. The text service provides interface `XText` and interface `XEnumerationAccess`. `XText` is responsible for editing a text and `XEnumerationAccess` is responsible for iterating over text. This part is almost the same as OpenOffice 1.1 with following exceptions. Developers can refer to section *7.3.1 Text Documents - Working with Text Documents - Word Processing* of OpenOffice 1.1 Development Guide.

- Editing text

  Method `setAttributes()` of `com.sun.star.accessibility.XAccessibleEditableText` might not work because the valid char index range of a character string might be beyond the length of the string.
- Inserting text files

Currently, Lotus Symphony documents does not support this function. Developers can create unexpected issues while using the associated APIs provided by OpenOffice 1.1.

- Auto text

  The auto text function can be used to organize reusable texts, which is the same as OpenOffice 1.1.

Formatting

Lotus Symphony documents formatting is the same as OpenOffice 1.1. Refer to section *7.3.2 Text Documents - Working with Text Documents - Formatting* of the OpenOffice 1.1 Development Guide.

Navigating

There are types of model cursors provided to navigate characters, words, sentences, or paragraphs. The `com.sun.star.text.TextCursor` service is a good example of a model cursor that is based on the interface `com.sun.star.text.XTextCursor`.

The text view cursor enables you to navigate over the document in the view by character, line, screen page, or document page. There is only one text view cursor. The information about the current layout, such as the number of lines and page number must be retrieved at the view cursor. The text view cursor is a `com.sun.star.text.TextViewCursor` service that includes the service `com.sun.star.text.TextLayoutCursor`.

Simultaneously, the text document model provides various suppliers that retrieve all text contents in a document. Refer to section *7.3.3 Text Documents - Working with Text Documents - Navigating* of the OpenOffice 1.1 Development Guide.

Note: In certain scenarios, the interface `com.sun.star.text.XSentenceCursor` might not work when the methods `isStartOfSentence()` or `isEndOfSentence()` are called.

Tables

Lotus Symphony tables are text contents and consist of rows, rows consist of one or more cells, and cells can contain text or rows. It is the same as OpenOffice 1.1 and there is no logical concept for columns. Refer to section *7.3.4 Text Documents - Working with Text Documents - tables* of the OpenOffice 1.1 Development Guide.

Note: Lotus Symphony documents enhanced the table to span pages that might have certain influences when using table-related APIs.

The method `insertByIndex()` of the `com.sun.star.table.XTableColumns` interface might not work because the design considers that inserting a column into a table should not be beyond the column range of the table. This limitation means that after the index number of insertion is beyond the range of the columns, the new column is appended after the last column of the table.

The method `removeByIndex()` of the `com.sun.star.table.XTableColumns` interface might not work because the prior limitation affects the column count of the table, and leads to the failure.

The method autoFormat() of com.sun.star.table.XAutoFormattable might not work when a table is formatted automatically. The auto-format item named "default" and some other auto-format items are selected randomly from the com.sun.star.sheet.TableAutoFormats service. After that, the results of two auto-formats should be checked to determine whether they are the same or not. In certain scenarios, the only one auto-format item named "default" is retrieved from com.sun.star.sheet.TableAutoFormats service, which is the same as the former one.

Text fields

Text fields are text contents that are used to add another level of information to text ranges. Usually their appearance fuses together with the surrounding text, but actually the presented text comes from elsewhere and is generated only while being painted. The types of Lotus Symphony fields are less than OpenOffice 1.1. Lotus Symphony documents field commands only support insertion of the current date, time, page number, total page numbers, and user field. If you use other services described in OpenOffice 1.1 Development Guide, they might create unexpected issues.

Fields are created through the com.sun.star.lang.XMultiServiceFactory and are inserted through the TextContent(). The following text field services are available:

- com.sun.star.text.textfield.DateTime. Show a date or time value.
- com.sun.star.text.textfield.PageCount. Show the number of pages of the document.
- com.sun.star.text.textfield.PageNumber. Show the page number (current, previous, next).
- com.sun.star.text.textfield.User. Variable - User Field. Creates a global document variable and displays it whenever this field occurs in the text. This service depends on com.sun.star.text.FieldMaster.User.

All fields support the interfaces com.sun.star.text.XTextField, com.sun.star.util.XUpdatable, com.sun.star.text.XDependentTextField and the service com.sun.star.text.TextContent. The method getPresentation() of the interface com.sun.star.text.XTextField is used to generate the textual representation of the result of the text field operation, such as a date, time, variable value of user field or TIME (fixed), depending on the Boolean parameter.

The method update() of the interface com.sun.star.util.XUpdatable affects only the following field types:

- Date and time fields are set to the current date and time.
- The ExtendedUser fields that show parts of the user data set for Lotus Symphony, such as the user fields that are set to the current values.
- All other fields ignore calls to update().

It is the same as OpenOffice 1.1 and some of these fields need a field master that provides the data that displays in the field. This requirement applies to the field types User. Refer to the section *7.3.5 Text Documents - Working with Text Documents – Text Fields* of OpenOffice 1.1 Development Guide.

Bookmarks

A bookmark is a kind of text content that marks a position inside of a paragraph or a text selection that supports the com.sun.star.text.TextContent service. The

text document model provides the interface
`com.sun.star.text.XBookmarksSupplier` to retrieve and collect the bookmarks.

Refer to section *7.3.6 Text Documents - Working with Text Documents - Bookmarks* of the OpenOffice 1.1 Development Guide.

Indexes and index marks

Indexes are also a kind of text content that centralize the information which is dispersed over the document. Index marks are another kind of text content which is the same as OpenOffice 1.1.

Refer to section *7.3.7 Text Documents - Working with Text Documents – Indexes and Index Marks* of the OpenOffice 1.1 Development Guide.

Note: Lotus Symphony documents do not feature a bibliographical index. The Table of Contents function of Lotus Symphony documents has been enhanced, which can influence the result of the related APIs.

Reference marks

A reference mark is a kind of text content that is acting as the target for the `com.sun.star.text.textfield.GetReference` text fields. These text fields can show the contents of reference marks in a text document and allow the user to jump to the reference mark.

Refer to section *7.3.8 Text Documents - Working with Text Documents – Reference Marks* of the OpenOffice 1.1 Development Guide.

Note: Lotus Symphony does not support
the `com.sun.star.text.textfield.GetReference` field. You might encounter unexpected issues when using the related APIs.

Footnotes and endnotes

Footnotes and endnotes are a kind of text content that are responsible for providing background information to the users on page footers or at the end of a document. The footnotes and endnotes of Lotus Symphony documents are the same as OpenOffice 1.1. Refer to section *7.3.9 Text Documents - Working with Text Documents – Footnotes and Endnotes* of the OpenOffice 1.1 Development Guide.

Shape objects in text

Shape objects are text contents that act independently of the ordinary text flow. Shape objects can float in front or behind text, and be anchored to paragraphs or characters in the text or page. It is the same as OpenOffice 1.1 and there are two different kinds of shape objects in Lotus Symphony: base frames and drawing shapes. Refer to section *7.3.10 Text Documents - Working with Text Documents – Shape objects in Text* of the OpenOffice 1.1 Development Guide.

Overall document features

Styles

Styles apply document-wide and can differentiate segments in a document that are commonly formatted, and separate this information from the actual formatting. It

is a good way to unify the appearance of a document, and customize the formatting of a document by altering a style, instead of using local format settings after the document has been completed. Styles are sets of attributes that can be applied to text or text contents in a text document in a single step.

Refer to section *7.4.1 Text Documents - Overall Document Features – Styles in Text* of the OpenOffice 1.1 Development Guide.

Line and outline numbering

Line and outline numbering is the same as OpenOffice 1.1 and Lotus Symphony provides automatic numbering for texts. For instance, paragraphs can be numbered or listed with bullets in a hierarchical structure, chapter headings can be numbered and lines can be counted and numbered. Refer to section *7.4.3 Text Documents - Overall Document Features – Line Numbering and Outline Numbering in Text* of the OpenOffice 1.1 Development Guide.

text section

It is the same as OpenOffice 1.1. A text section is a range of complete paragraphs that can have its own format settings and source location. Refer to section *7.4.4 Text Documents - Overall Document Features – Text Sections in Text* of the OpenOffice 1.1 Development Guide.

Page layout

The Lotus Symphony page layout is the same as OpenOffice 1.1. Refer to the section *7.4.5 Text Documents - Overall Document Features –Page Layout* of the OpenOffice 1.1 Development Guide.

Text document controller

The text document controller provides access to the graphical user interface for the model and has knowledge about the current view status in the user interface. Refer to section *7.5 Text Documents - Text Document Controller* of the OpenOffice 1.1 Development Guide.

Text view

Text views is the same as OpenOffice 1.1. Refer to the section *7.5.1 Text Documents - Overall Document Features – Text Document Controller - TextView* of the OpenOffice 1.1 Development Guide.

TextViewCursor

TextViewCursor is the same as OpenOffice 1.1. Refer to the section *7.5.2 Text Documents - Overall Document Features – Text Document Controller - TextViewCursor* of the OpenOffice 1.1 Development Guide.

2.4 Spreadsheets

Spreadsheet documents derive all UNO APIs from OpenOffice.org 1.1.0. The exposed APIs are almost the same as OOo1.1.0. Comparing to OOo1.1.0, functional quality has been improved on the core function, so that the API quality is enhanced accordingly when interfaces remain. Several APIs have been added or changed.

Different spreadsheet elements are presented by different interfaces in different services.

Operations of spreadsheet documents are mainly in those interfaces :

- *com.sun.star.sheet.SpreadDocument*. Whole document
- *com.sun.star.sheet.XSpreadsheet*. Sheet
- *com.sun.star.frame.XStorable*. Document saving and exporting
- *com.sun.star.view.XPrintable*. Document printing
- *com.sun.star.util.XProtectable*. Contains methods to protect and unprotect spreadsheet with a password, and also including text in cells, cell ranges, table rows, and columns

Operations of single cells are in these interface:

- *com.sun.star.sheet.SheetCell*. Used to present cell object
- *com.sun.star.table.CellProperties* . Used to format cells

Operations of cell range are in these interface: The service com.sun.star.sheet.SheetCellranges contains most of the interface of a cell range. A cell range can be named with *com.sun.star.container.XNamed*.

Operations on cell ranges are covered by *com.sun.star.util.XReplaceable(Search, Find and Replace)*, *com.sun.star.table.TableSortDescriptor*(Sort), *com.sun.star.sheet.SheetFilterDescriptor*(Filter), *com.sun.star.sheet.SubtotalDescriptor*(Subtotal functions). The spreadsheet interface *com.sun.star.sheet.XSheetOutline* contains all the methods to control the row and column outlines of a spreadsheet.

User interface refresh:

A spreadsheet document often gets a cell value by invoking an API. Compared to filling in the cell value manually, the API updates cell values more frequently, which can cause the update of a large range of spreadsheet cells because of cross referencing among cells. To resolve this issue, use this method:

interface XCellRange;

void SyncDocument([in] boolean bEnable)

Note: This method is used to resolve the performance issue when changing the values of a number of cells by the UNO API. This method must be called in pairs. When SyncDocument is disabled, only cells that have a value changed are updated in user interface. All of the formulas or charts depending on this cell do not get refreshed until SyncDocument is enabled.

Sample code:

```
/** whether to sync document data and update document status when changing content
  * in cells. SyncDocument(FALSE) and SyncDocument(TRUE) should be called in pairs
  * @param bEnable
  * when bEnable is TRUE, it will sync immediately and set document modified.
  *  when FALSE, some data and UI don't update immediately when changing content in
  *  cells.
 */
 SyncDocument(FALSE); //disable to update some of UI and document data
 for(i=0;i<100;i++)
     setcell(a, i, 1);
 SyncDocument(TRUE); //enable and update immediately.
```

Import external data from a file:

Interface XAreaLinks;

```
insertAtPosition([in] com.sun.stat.Table.CellAddress     aDestPos,
             [in] string                                 aFileName,
             [in] string                             aSourceArea,
             [in] string                             aFilter,
            [in] string                              aFilterOptions,
             [in] boolean                                 bLink);
```

A new parameter, bLink, is added to this method. When bLink == True, the source area is inserted to aDestPos with linkage kept. When bLink == False, only the value is inserted.

Do not use the following UNO APIs because they have not been fully tested:

Interface and methods in service com.sun.star.sheet.DDELinks.

Interface and methods in service com.sun.star.sheet.DatabaseImportDescriptor.

Interface and methods in service com.sun.star.sheet.Scenarios.

Methods in interface com.sun.star.sheet.XSheetAuditing.

Methods to import data from a Web server.

Charts

In Lotus Symphony, charts are always embedded objects inside other Lotus Symphony documents. The chart document UNO API is almost the same as OpenOffice.org 1.1.0. Like the spreadsheet document, enhancements have been added in the Lotus Symphony core function, which improves the API quality.

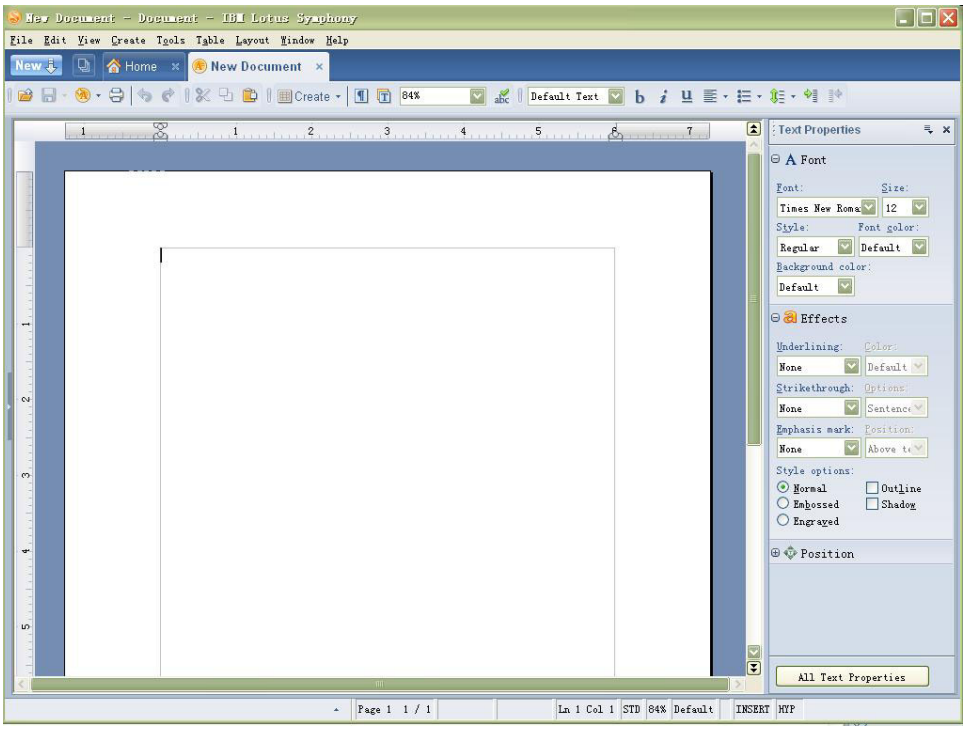
Charts can be added into spreadsheet documents with data in a cell range. In a presentation document or a writer document, a chart can be added as an OLE shape. The Lotus Symphony chart API provides the capability of creating charts, accessing existing charts, and modifying chart properties and elements. Ideally all the operations which can be accomplished with UI can be done by API (refer to OpenOffice.org 1.1.0 Dev guide). Because of core function, the operations are not supported by the APIs with discrete data source in spreadsheet.

Part 6. Sample Plug-ins

First, you need verify the Lotus Symphony development environment on Eclipse as following steps:

1. Set up the Lotus Symphony development environment according Symphony Developer's Guide provided by the toolkit.
2. Click **Run** from the toolbar to launch Lotus Symphony. If the Run option is disabled, select **Run > Run** to open the runtime configuration window. Select **Client Services > Symphony** and then click the **Run** button. If asked whether you want to clear the runtime workspace, select **yes**.

3. When the Lotus Symphony window open, click **File > New > Document** .



This window is the standard Lotus Symphony document editor. In the next section you will add an Eclipse plug-in to the development environment and test that it works.

Select **File > Exit** to close the runtime instance of Lotus Symphony before continuing.

Chapter 1. Hello World Sample Plug-in

1.1 Creating a new plug-in

Launch the Eclipse development environment

1. Click **File > New > Project** .
2. Select **Plug-in Project** , and click **Next** .
3. Type `com.ibm.productivity.tools.samples.helloworld` in the **Project name** field. Click **Next**.
4. Type a descriptive name in the **Plug-in Name** field, for example, **hello world sample**.
5. Click **Finish**.

1.2 Adding the plug-in dependency

The following table lists some of the plug-in dependencies used by the document library, plug-in names are abbreviated:

Plug-in	Description
org.eclipse.core.runtime, org.eclipse.ui	Eclipse core plug-ins
com.ibm.productivity.tools.ui.views com.ibm.productivity.tools.core	Lotus Symphony API plug-in

Perform the following steps to add the plug-in dependency.

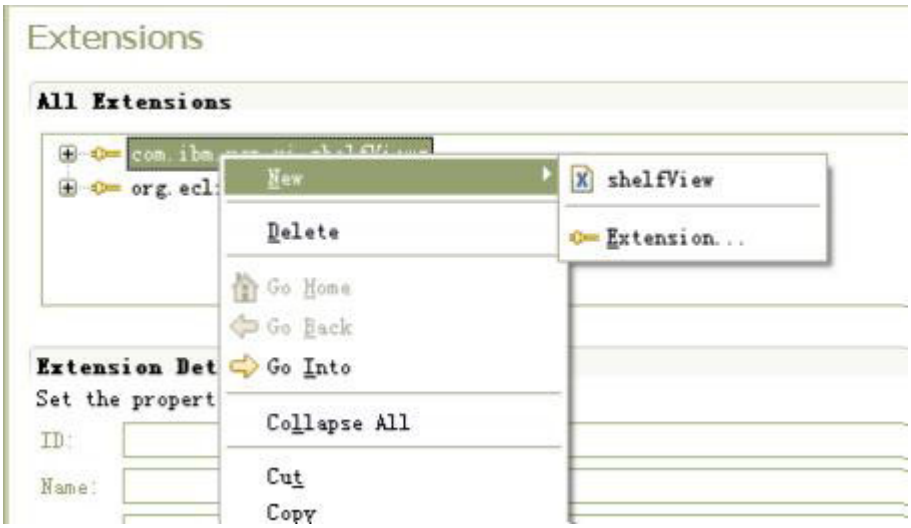
1. Click the Dependencies tab of the Hello world plug-in manifest.
2. Click Add.
3. Add the following plug-ins:
   - com.ibm.productivity.tools.ui.views
   - com.ibm.productivity.tools.core

Note: Add these plug-in dependencies to the `MANIFEST.MF` file, which defines the plug-in. You can see the contents of this file by turning to the Plug-in Manifest Editor's `MANIFEST.MF` tab:

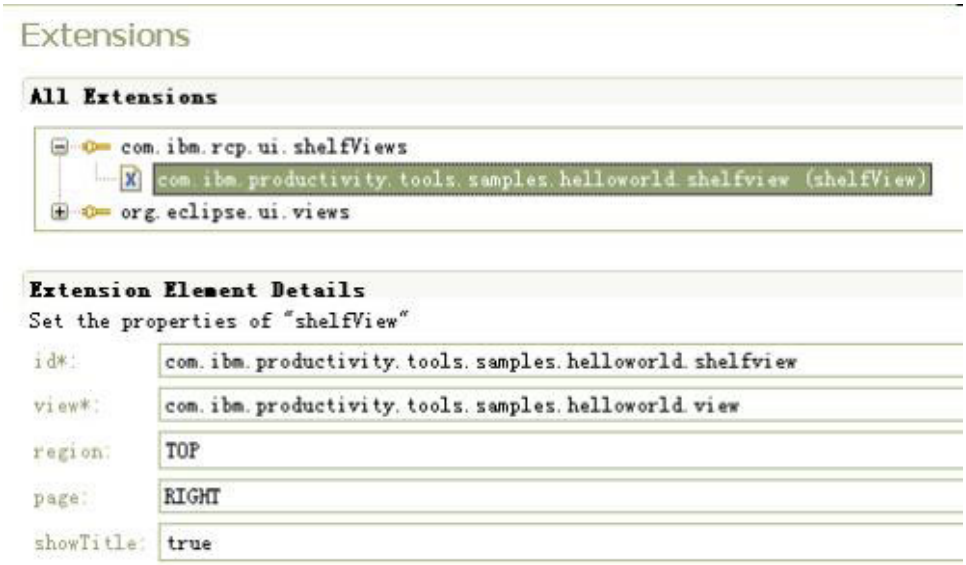
```
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 com.ibm.productivity.tools.ui.views,
 com.ibm.productivity.tools.core
```

1.3 Adding a side shelf element

1. Click the **Extensions** tab.
2. Click **Add**.
3. Add the followings extension:com.ibm.rcp.ui.shelfViews.
4. Click **Finish**.
5. Right-click the added extension and select **New > shelfView**.



Selecting this menu choice adds a shelfview element to the extension declaration. Select the newly added element and note that the Extension Element Details is updated to show the possible attributes. Fill in the fields as shown below.



The asterisk (*) indicates a required attribute. One of particular importance is the class attribute which indicates the Java class that will implement the shelfview's behavior (that is, this class defines what the side shelf area will contain and how it will respond to user events.).

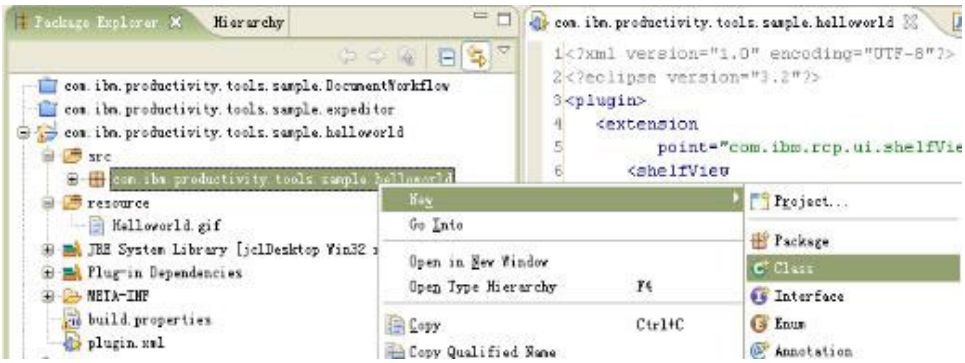
6. Click the **plugin.xml** tab
7. Copy and paste the following into the `plugin.xml` file.

```
<extension
      point="org.eclipse.ui.views">
   <category
         name="Helloworld Category"
         id="com.ibm.productivity.tools.sample">
   </category>

   <view
         name="Hello World"
         icon="resuorce/Helloworld.gif "
         category="com.ibm.productivity.tools.sample"
         class="com.ibm.productivity.tools.samples.helloworld.ShelfView"
         id="com.ibm.productivity.tools.samples.helloworld.view">
   </view>

</extension>
```

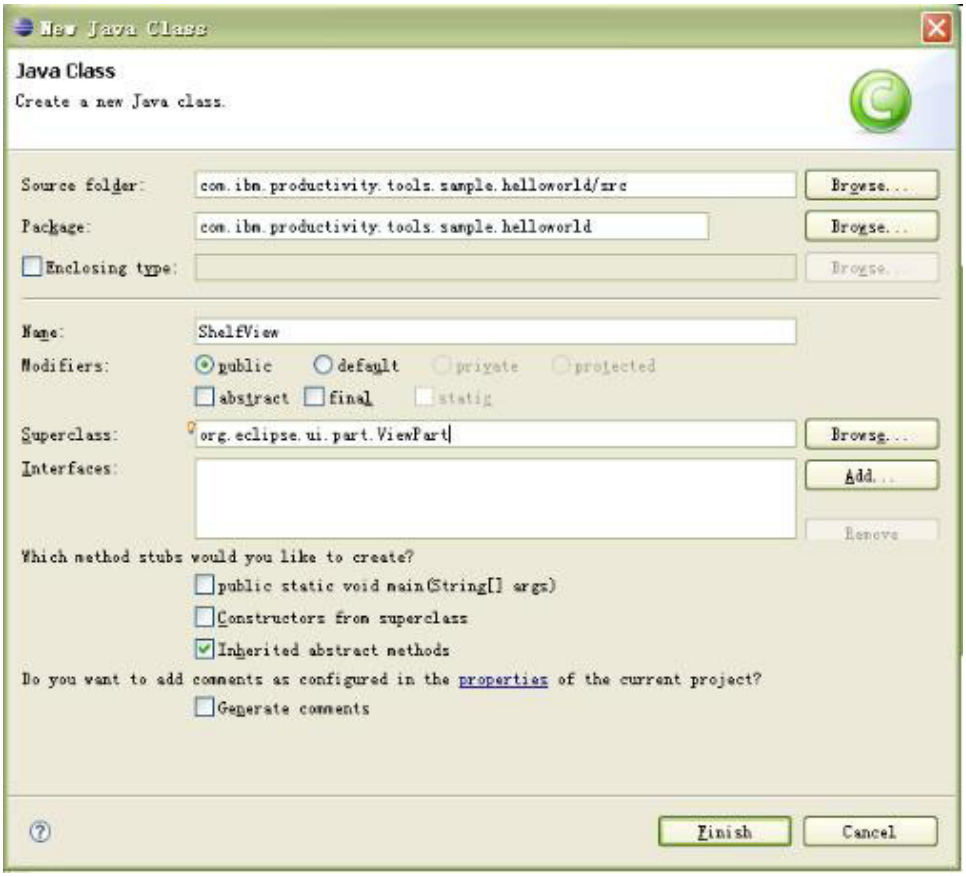
The view attribute of the <shelfView> tag in the com.ibm.rcp.ui.shelfViews extension must match the id attribute of the <view> tag in the org.eclipse.ui.views extension exactly. That is, the side-shelf content is defined by the extensions <view> / <shelfView> pairs.

The prior steps adds a new Eclipse ViewPart to the platform. You can create your plug-in extensions with Manifest Editor or enter the specifications directly in the `plugin.xml` file.

8. Right-click the package com.ibm.productivity.tools.samples.helloworld in **Package Explorer**, and then click **New > Class**.



9. Input the class information as follows.You can click the **Browse** to search the superclass of org.eclipse.ui.part.ViewPart.



A new Eclipse ViewPart named ShelfView is created in the com.ibm.productivity.tools.samples.helloworld package.

1.4 Running the application

1. Check your plug-in.

   Before running the application, take a look at the plugin.xml file and the newly created class.

   The `plugin.xml` file is like the following:

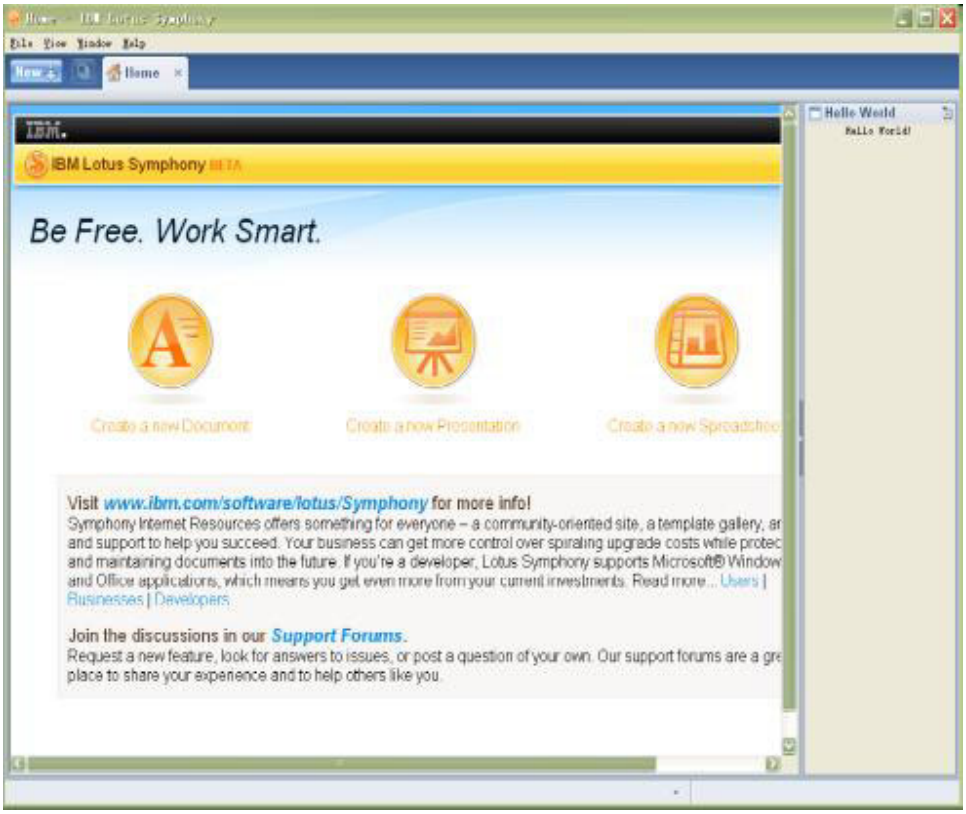
```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
   <extension
         point="com.ibm.rcp.ui.shelfViews">
      <shelfView
            id="com.ibm.productivity.tools.samples.helloworld.shelfview"
            page="RIGHT"
            region="BOTTOM"
            showTitle="true"
            view="com.ibm.productivity.tools.samples.helloworld.view"/>
   </extension>

   <extension
         point="org.eclipse.ui.views">
      <category
            name="Helloworld Category"
            id="com.ibm.productivity.tools.sample">
      </category>

      <view
            name="Hello World"
            icon="resuorce/Helloworld.gif "
            category="com.ibm.productivity.tools.sample"
            class="com.ibm.productivity.tools.samples.helloworld.ShelfView"
            id="com.ibm.productivity.tools.samples.helloworld.view">
      </view>

   </extension>

</plugin>
```

2. Double-click the `ShelfView.java` file in **Package Explorer**, the `ShelfView.java` file looks like the following:

```
package com.ibm.productivity.tools.samples.helloworld;

import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.part.ViewPart;

public class ShelfView extends ViewPart {

    public void createPartControl(Composite parent) {
        parent.setLayout(new FillLayout());
        Label helloLabel = new Label(parent, SWT.CENTER);
        helloLabel.setText("Hello World!");
    }

    public void setFocus() {
    }
}
```

3. Click **Run** from toolbar:

4. Lotus Symphony is launched, your screen should look similar to the following image:



Hint: If the new view does not display, check the console for a message like org.eclipse.ui.PartInitException: Could not create view: XXX and confirm that XXX = the view id. The view attribute of the <shelfView> tag in the com.ibm.rcp.ui.shelfViews extension must match the id attribute of the <view> tag in the org.eclipse.ui.views extension.

Congratulations! You have reserved space in the Lotus Symphony side shelf for your application.

Chapter 2. Editor View Sample Plug-in

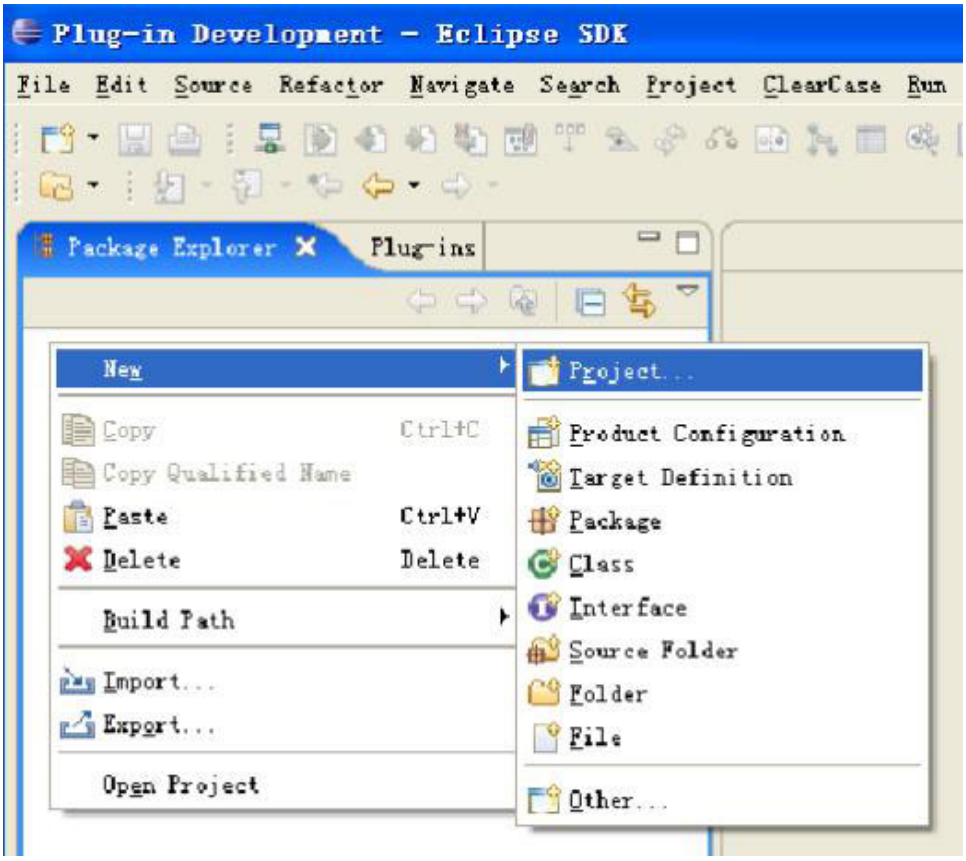
This sample demonstrates how to create a simple editor. When launched from within the new button group, the editor is showed as a view part in a new perspective.

You can find the whole project with all source code from Lotus Symphony toolkit directory (where $symphony_toolkit is the home directory that the API toolkit is installed to):

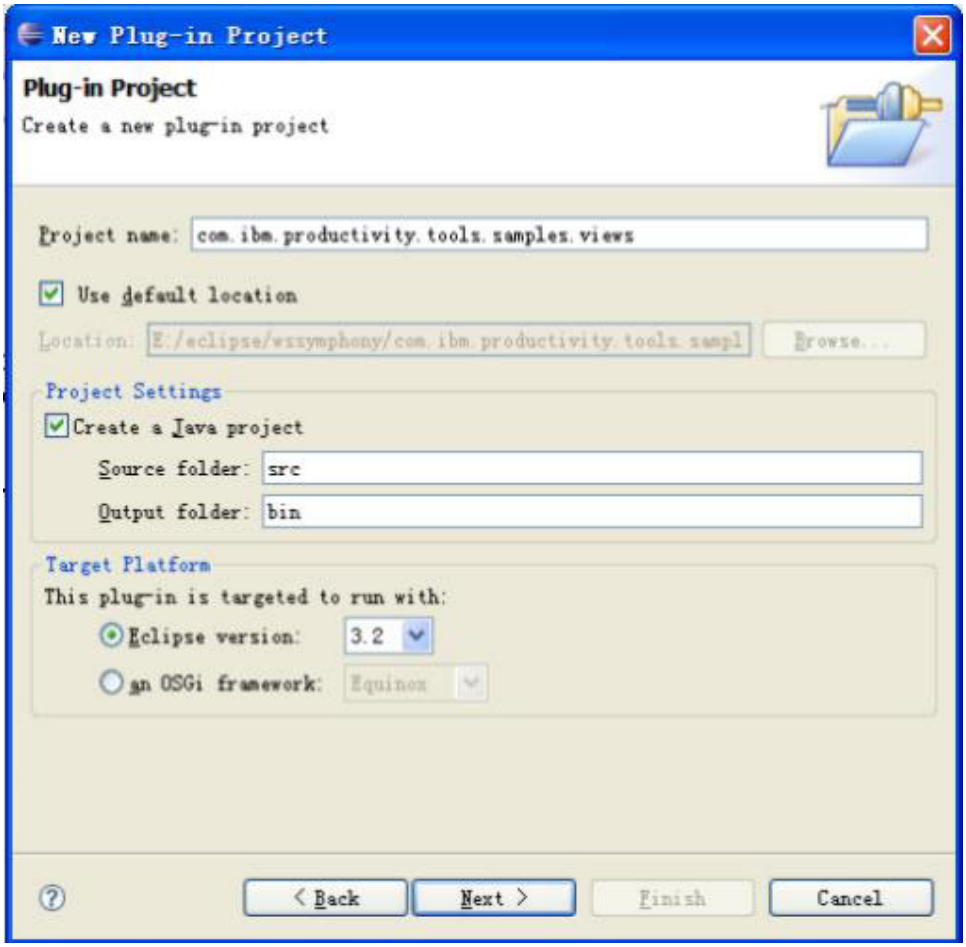
$symphony_toolkit/samples/eclipse/plugins/
com.ibm.productivity.tools.samples.vie ws.

2.1 Creating a plugin

1. Set up the integrated development environment as discussed in Part 4 Chapter 1.
2. Select File > New > **Project** or right-click and select **New > Project**.



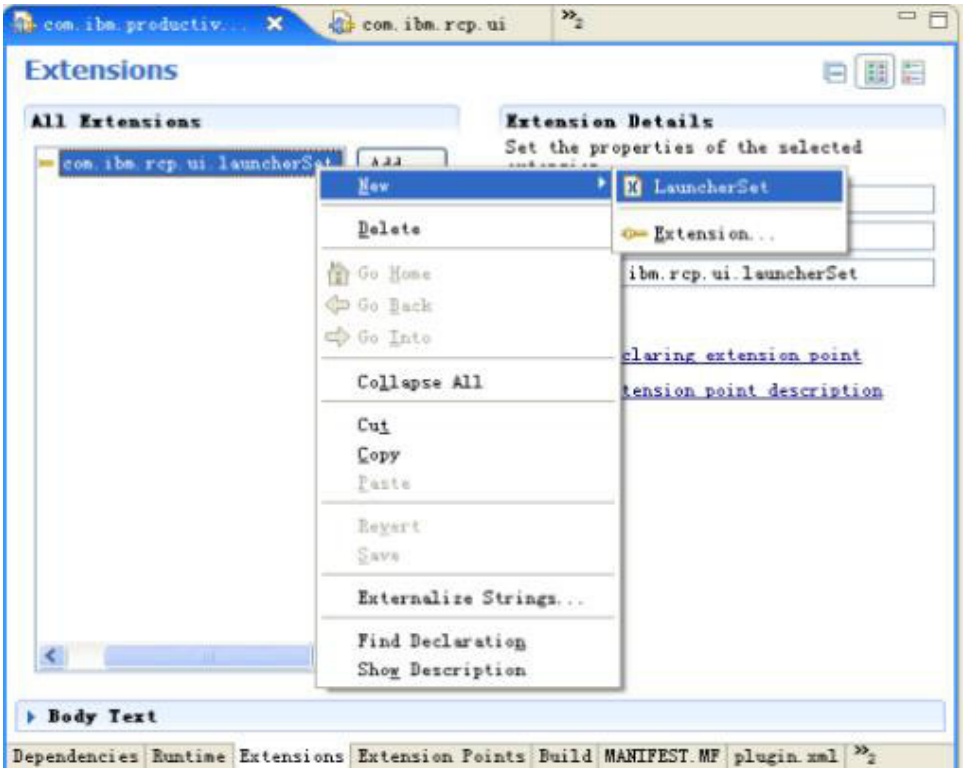
3. Select the Plug-in Project in the Project Category.
4. Click **Next** and type `com.ibm.productivity.tools.samples.views` as the project name. Click **Next**, and then click **Finish** to finish creating the new project.



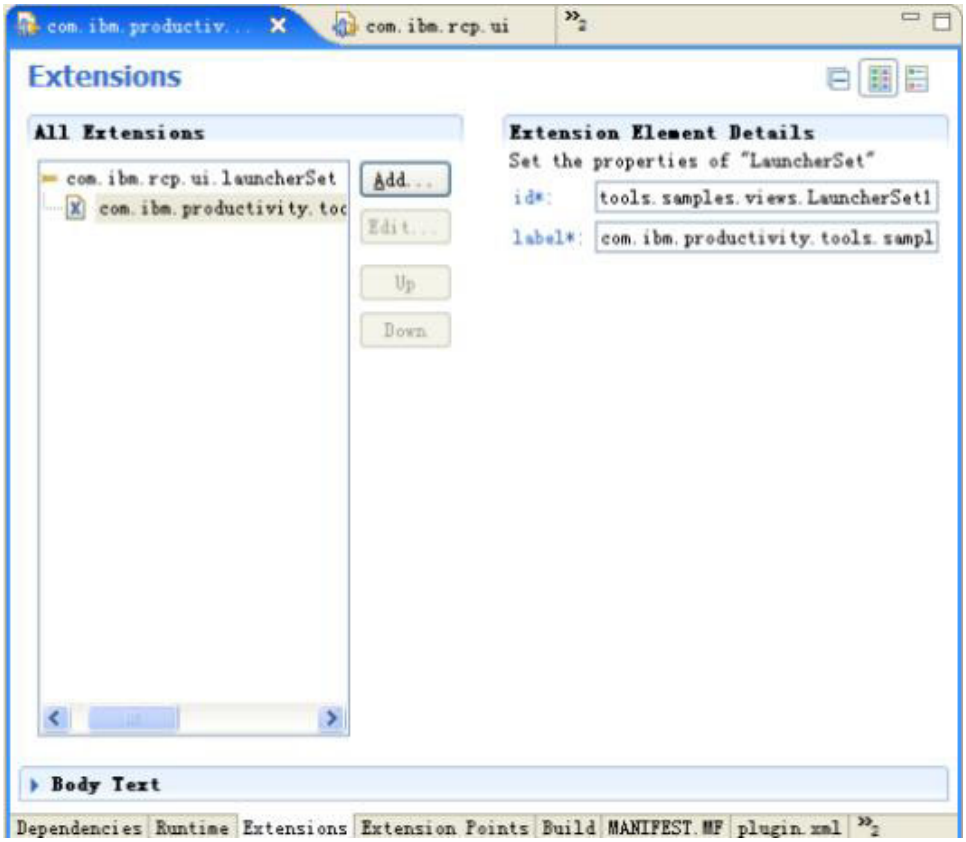
5. Add dependencies. Select the **Dependencies** tab, and click **Add** to add the required plug-ins:
   - `com.ibm.rcp.ui`
   - `com.ibm.productivity.tools.ui.views`

2.2 Creating a new button

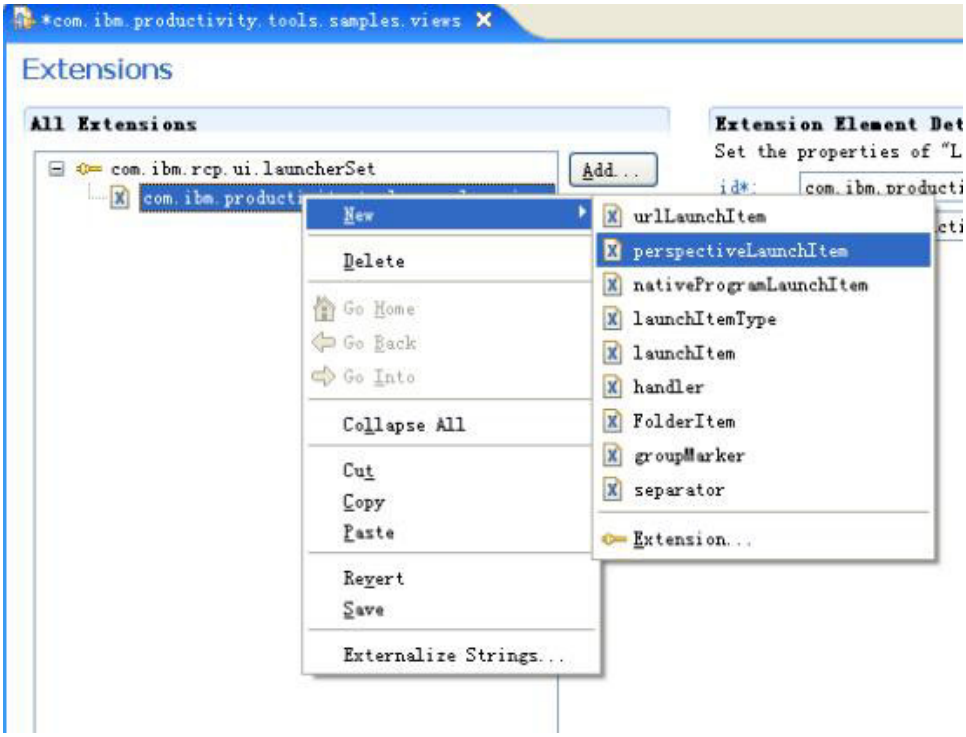
1. Select the **Extensions** tab and click **Add**. In the new extensions window, select **com.ibm.rcp.ui.launcherSet**, and then click **Finish**.
2. In the Extensions page, right-click the added extension and select **New > LauncherSet**.



3. Leave the id and label properties of the LauncherSet unchanged, and save the `plugin.xml` file.



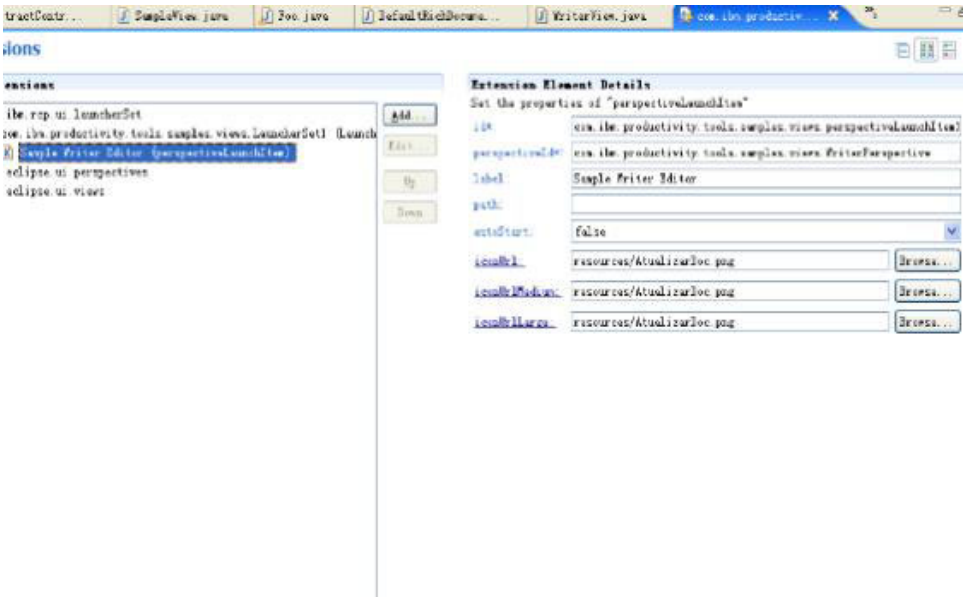
4. Right-click com.ibm.productivity.tools.samples.views.LauncherSet1, and select **New > perspectiveLaunchItem**.



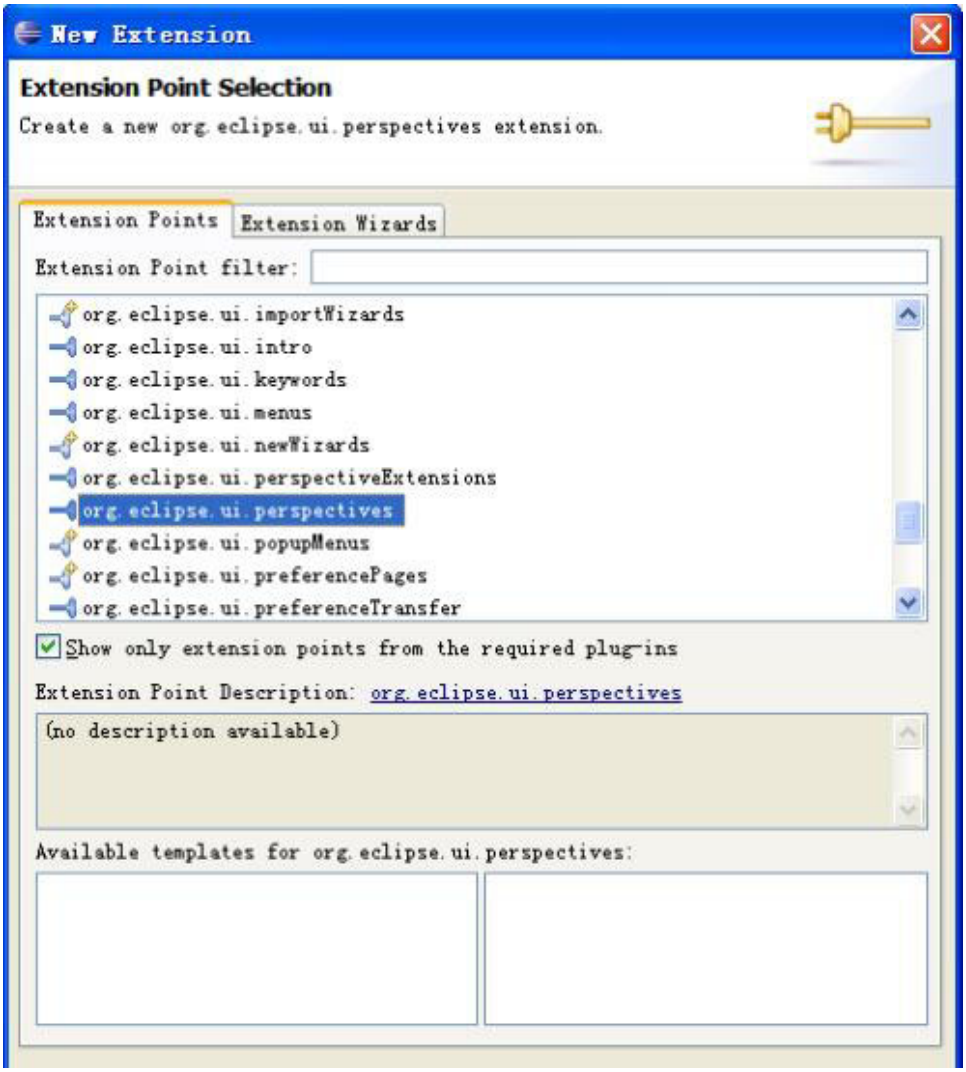
5. Set the properties of perspectiveLaunchItem as shown in the following sample code:

```
<extension
      point="com.ibm.rcp.ui.launcherSet">
   <LauncherSet
         id="com.ibm.productivity.tools.samples.views.LauncherSet1"
         label="%simpleeditor2.launcherSet1">
      <perspectiveLaunchItem
            autoStart="false"
            iconUrl="resources/AtualizarDoc.png"
            iconUrlLarge="resources/AtualizarDoc.png"
            iconUrlMedium="resources/AtualizarDoc.png"
            id="com.ibm.productivity.tools.samples.views.perspectiveLaunchItem1"
            label="%sample.editor.spreadsheet"
            perspectiveId="com.ibm.productivity.tools.samples.views.WriterPerspective
           ">
      </perspectiveLaunchItem>
   </LauncherSet>
</extension>
```

Make sure that the **perspectiveID** is `com.ibm.productivity.tools.samples.views.WriterPerspective`, and then save the `plugin.xml` file.



6. Add an extension at extension point org.eclipse.ui.perspective.



7. Select the **plugin.xml** tab. Change the extension declaration of the added perspectives extension point as shown in the following sample code:

```
<extension point="org.eclipse.ui.perspectives">
    <perspective
        class =
"com.ibm.productivity.tools.samples.views.WriterPerspective"
        name = "Sample Writer Editor"
        id = "com.ibm.productivity.tools.samples.views.WriterPerspective"
    />
</extension>
```

8. Create a Java class named `com.ibm.productivity.tools.samples.views.WriterPerspective`:

```
package com.ibm.productivity.tools.samples.views;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

/**
 * Perspective class of writer editor sample
  */
public class WriterPerspective implements IPerspectiveFactory {

        public static final String PERSPECTIVE_ID =
        "com.ibm.productivity.tools.samples.views.WriterPerspective";
    /* (non-Javadoc)
     * @see org.eclipse.ui.IPerspectiveFactory#createInitialLayout
        (org.eclipse.ui.IPageLayout)
     */
    public void createInitialLayout( IPageLayout layout ) {
        //set editor area to invisible so that our view can show maximized.
        layout.setEditorAreaVisible(false);

        //add our writer view to this perspective
        layout.addView(WriterView.VIEW_ID, IPageLayout.LEFT, 1f, layout.getEditorArea
            ());
    }
}
```

2.3 Creating an editor view part

1. Select the **Extensions** tab and click **Add**.
2. Add new extensions org.eclipse.ui.views, then click **Finish**.

3. Select the **plugin.xml** tab, and add the markup as shown in the following sample code:

```
<extension
      point="org.eclipse.ui.views">
     <view
            allowMultiple="true"
            class="com.ibm.productivity.tools.samples.views.WriterView"
            id="com.ibm.productivity.tools.samples.views.WriterView"
            name="Writer View" />
</extension>
```

4. Create a view class. Select **New > Class** to create a new Java class for the view. Set the Class arguments as shown below:

   Package: `com.ibm.productivity.tools.samples.views`

   Name: `WriterView`

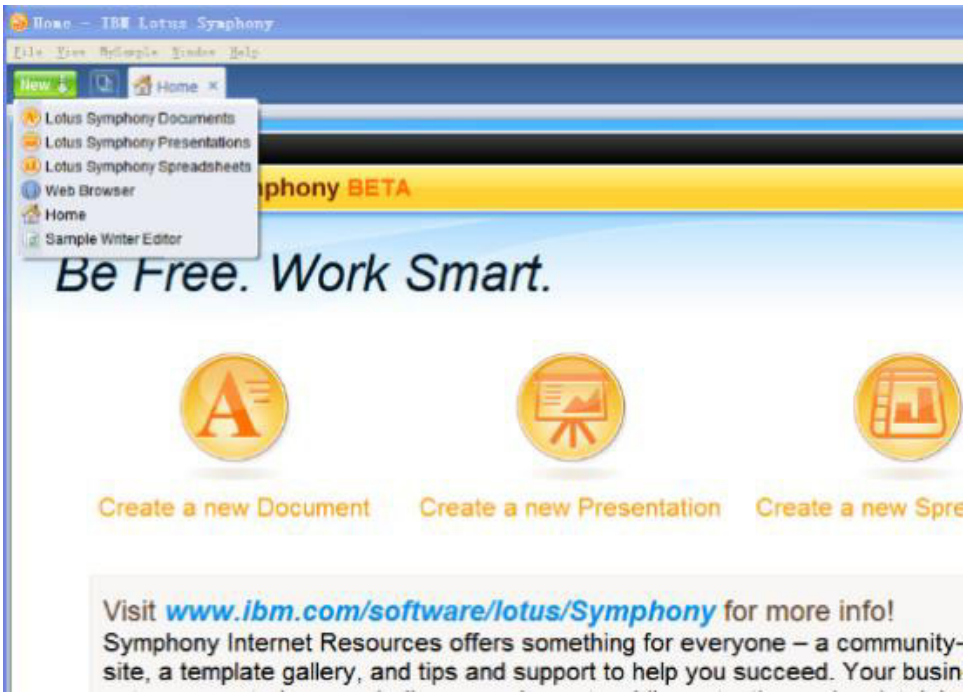
   Superclass:
   `com.ibm.productivity.tools.ui.views.DefaultRichDocumentView`

   and then click **Finish**.

5. Implement the WriterView class as shown in the following sample code:

```
package com.ibm.productivity.tools.samples.views;

import org.eclipse.swt.widgets.Composite;

import com.ibm.productivity.tools.ui.views.DefaultRichDocumentView;
import com.ibm.productivity.tools.ui.views.RichDocumentType;
import com.ibm.productivity.tools.ui.views.operations.NewOperation;
import com.ibm.productivity.tools.ui.views.operations.OperationFactory;

public class WriterView extends DefaultRichDocumentView {

            public static final String VIEW_ID =
            "com.ibm.productivity.tools.samples.views.WriterView";

            public WriterView() {
                super();
            }

            public void createPartControl(Composite parent) {
                // must call super to create part control
                super.createPartControl(parent);

                NewOperation operation = OperationFactory.createNewOperation
            (RichDocumentType.DOCUMENT_TYPE );
                this.executeOperation( operation );
            }
}
```

The following figure shows the result of creating an editor viewpart:



Chapter 3. Spreadsheet sample plug-in

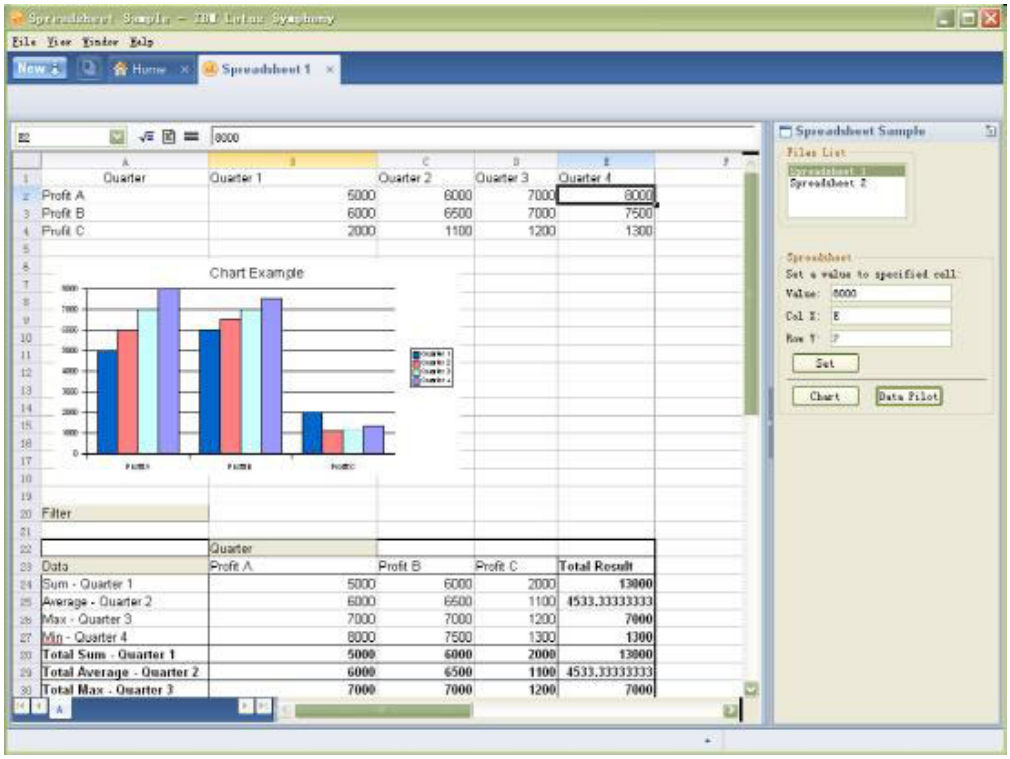
This chapter shows how to begin to add a customized Lotus Symphony spreadsheet UI plug-in and operate the Spreadsheet on a Lotus Symphony side shelf.

Note: All sample code used within this chapter can be found in the Lotus Symphony Toolkit, such as $symphony_toolkit/samples/eclipse/plugins/com.ibm.productivity.tools.samples.spreadsheet. You can get this toolkit from the site: http://symphony.lotus.com.

In the spreadsheet sample plug-in, it shows how to:

1. Add a customized shelf view.
2. Open a spreadsheet and get the model of this document.
3. Insert data into the spreadsheet.
4. Get the current selected cell's value and its address dynamically.
5. Create a chart of this sheet.
6. Create a data pilot of this sheet.

The following figure shows this sample plug-in's overview image.



3.1 Introduction to the scenario

When you want to import data from a database or from files into a spreadsheet, first, you need open the spreadsheet and get its model for operating before you can insert data into it. So open and insert a data into a sheet is a basic operator for operating a spreadsheet. Then you might need to create a chart for this sheet to make a overview of this sheet's data. Or you might need to set focus and do analysis on this sheet, in which case you should use the data pilot.

3.2 Preview of the result

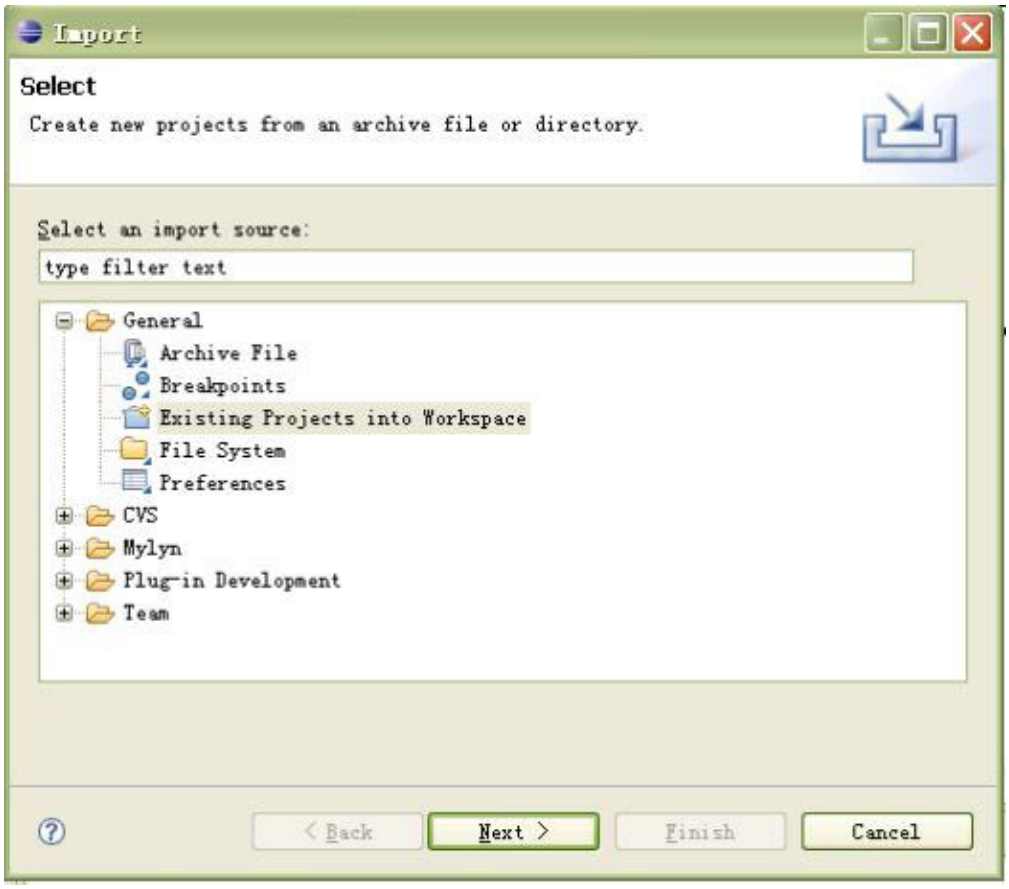
According to the scenario above, this plug-in first creates a shelf view, then adds a list view on the side shelf to show the spreadsheet file list. Then it adds three text fields and a button for setting data to a specified cell. It adds a button for creating a chart and a button for creating a data pilot. When you select a cell in the sheet, you get the value of this cell and its address on the side shelf dynamically.

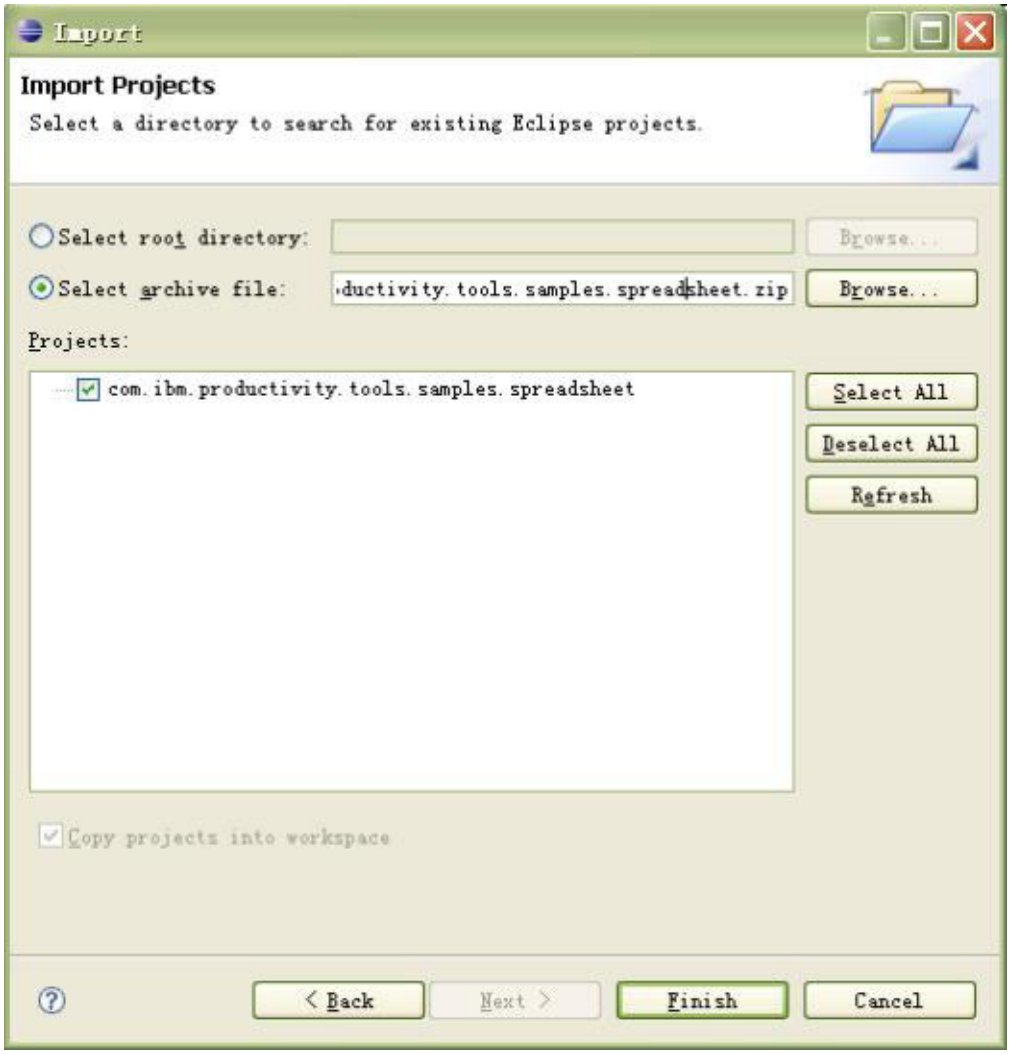
3.3 Prepare your development environment

Refer to Part 4 chapter 1: Setting up the integrated development environment, which shows how to prepare your Lotus Symphony development environment step by step.

3.4 Deploying the sample

If you already have this plug-in, you can import it into Eclipse from an existing project by using the Eclipse import function. Otherwise, the following sections show you how to build this plug-in.





3.5 Creating the sample

Creating a new plug-in

1. Launch the Eclipse development environment.
2. Click **File > New > Project**.
3. Select **Plug-in Project**, and click **Next**.
4. Type `com.ibm.productivity.tools.samples.spreadsheet` in the **Project name** field. Click **Next**.
5. Type a descriptive name in the **Plug-in Name** field, for example, Spreadsheet sample.
6. Click **Finish**.

Adding the plug-in dependency

The following table lists some of the plug-in dependencies used by the document library . The plug-in names are abbreviated.

Plug-in	Description
org.eclipse.core.runtime org.eclipse.ui	Eclipse core plug-ins
com.ibm.productivity.tools.ui.views com.ibm.productivity.tools.core	Lotus Symphony API plug-ins

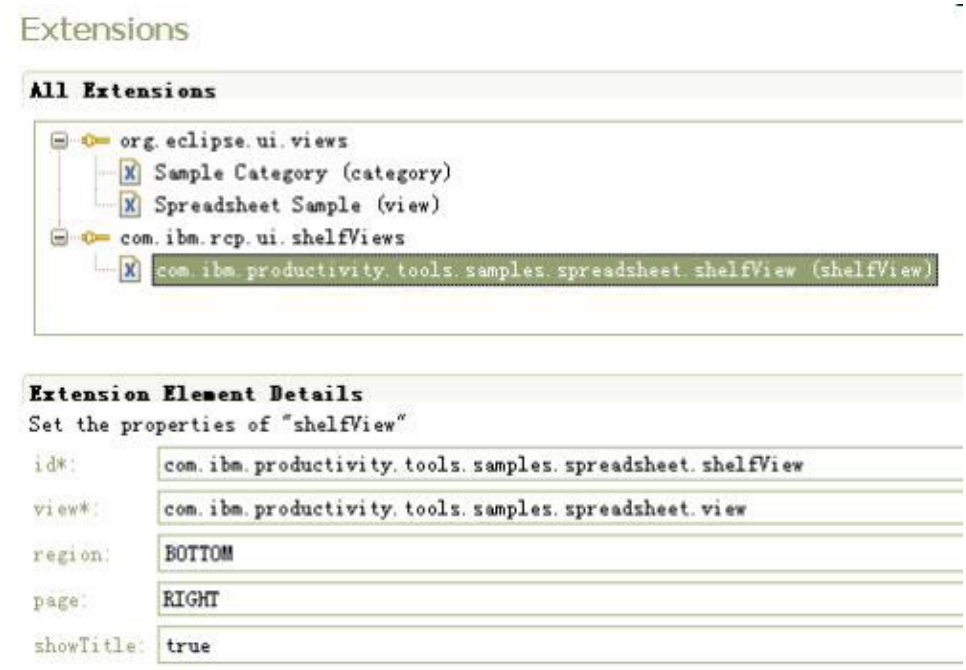
Perform the following steps to add the plug-in dependency.

1. Click the **Dependencies** tab of the spreadsheet sample plug-in manifest.
2. Click **Add**.
3. Add the following plug-ins:
   - com.ibm.productivity.tools.ui.views
   - com.ibm.productivity.tools.core

Adding an element to the side shelf

1. Click the **Extensions** tab.
2. Click **Add**.
3. Add the following extension: com.ibm.rcp.ui.shelfViews.
4. Click **Finish**.
5. Right-click the added extension and select **New > shelfView**.
6. Click the **plugin.xml** tab.
7. Copy and paste the following sample code into the **plugin.xml**.

```
<plugin>
<extension
       point="org.eclipse.ui.views">
    <category
          name="Sample Category"
          id="com.ibm.productivity.tools.sample">
    </category>
    <view
          name="Spreadsheet Sample"
          icon="resources/spreadsheetview.gif"
          category="com.ibm.productivity.tools.sample"
          class="com.ibm.productivity.tools.samples.spreadsheet.ui.ShelfView"
          id="com.ibm.productivity.tools.samples.spreadsheet.view">
    </view>

 </extension>
 <extension
       point="com.ibm.rcp.ui.shelfViews">
    <shelfView
          id="com.ibm.productivity.tools.samples.spreadsheet.shelfView"
          page="RIGHT"
          region="BOTTOM"
          showTitle="true"
          view="com.ibm.productivity.tools.samples.spreadsheet.view"/>
 </extension>
</plugin>
```

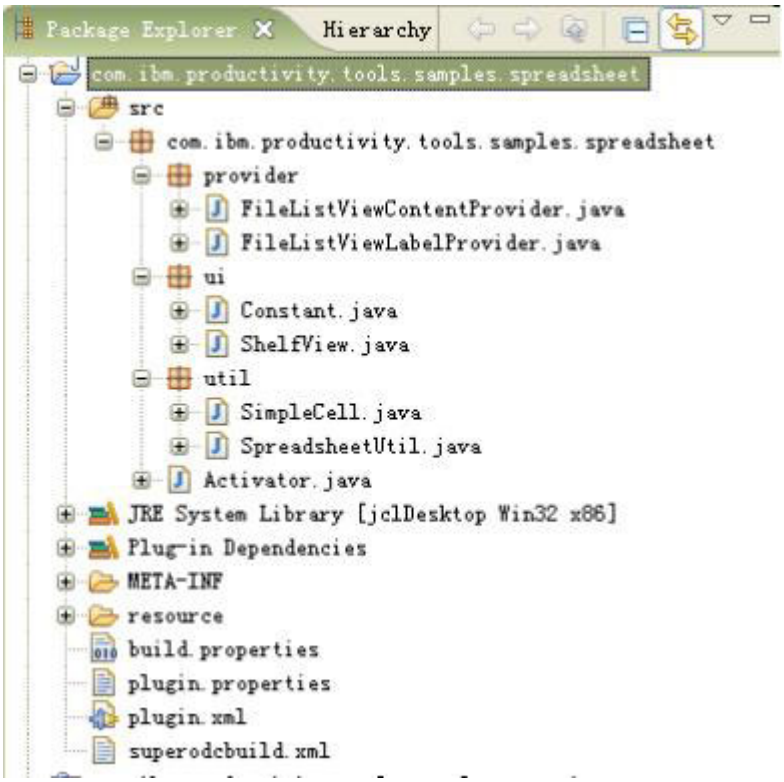


8. Create a folder named ui and a class named `ShelfView` which extends org.eclipse.ui.part.ViewPart under this folder. The main method in this class is

shown in the following sample code:

```
public void createPartControl(Composite aParent) {
      ......
      drawFileListGroup();
      drawTableGroup();
      addListener();
}
```

The method drawFileListGroup() creates a ListViewer to show the file lists for this sample and opens the files in this list when you double-click the file name. The method drawTableGroup() creates three text and a set button for setting the specified cell value. It also creates a button named Chart to create a chart for this sheet, and a button named Data Pilot to create a datapilot sample for this sheet. There are also assistant classes for the class ShelfView, for the details see the sample code.



3.6 Core code demonstration

The following section shows core code snippets for the function. For details, refer to the sample code.

1. Add a side shelf to the Lotus Symphony.

   Refer to the section **Adding a side shelf element** of Chapter 1 Hello world sample plug-in .

2. Open a spreadsheet file and get this sheet's model.

```
// the parameter url is this sheet file's url.
RichDocumentView view = RichDocumentViewFactory.openView(url, false);
Object model = view.getUNOModel();
```

3. Set a value in a cell:

Wherever you get data, setting a value in a cell is a basic operation. First get the sheet's model, then get the cell by specifying the position and setting the value in this cell.

```
(XSpreadsheetDocument)UnoRuntime.
  queryInterface(XSpreadsheetDocument.class, model ).getSheets();

 XIndexAccess xSheetsIA =(XIndexAccess)UnoRuntime.queryInterface(
    XIndexAccess.class, xSheets);
//get the first sheet in the document
 xSheet = (XSpreadsheet) UnoRuntime.queryInterface(
   XSpreadsheet.class, xSheetsIA.getByIndex( 0 ));

 oCell = xSheet.getCellByPosition(x , y );
 oCell.setValue(value);
```

4. Create a chart for this sheet.

   First get the chart object of this sheet by specifying the range, which decides the cells' data in this chart, then set this chart's properties, such as specifying this chart as a 3D chart or a pie chart.

```
XTableChart chart = (XTableChartsSupplier)UnoRuntime.queryInterface(
          XTableChartsSupplier.class, xSheet).getchart();

XEmbeddedObjectSupplier oEOS = (XEmbeddedObjectSupplier)UnoRuntime.
             queryInterface(XEmbeddedObjectSupplier.class, chart);
```

5. Create a data pilot for this sheet:

   First, set source range for this data pilot, and then set properties field for this data pilot.

```
XDataPilotTablesSupplier xDPSupp = (XdataPilotTablesSupplier)
 UnoRuntime.queryInterface(XDataPilotTablesSupplier.class, xSheet);

 XDataPilotTables xDPTables = xDPSupp.getDataPilotTables();
 XDataPilotDescriptor xDPDesc = xDPTables.createDataPilotDescriptor();
XIndexAccess xFields = xDPDesc.getDataPilotFields();
Object aFieldObj;
XPropertySet xFieldProp;
// use first column as column field
aFieldObj = xFields.getByIndex(0);
xFieldProp = (XPropertySet)
UnoRuntime.queryInterface(XPropertySet.class, aFieldObj);
xFieldProp.setPropertyValue ("Orientation",
     DataPilotFieldOrientation.COLUMN);
```

3.7 Extending the sample

Next, you can add a mapping table of this sheet in the side shelf and you can add more functions to operating a spreadsheet, such as loading, saving, and closing a sheet.You can also export this sheet file as a HTML file.

Chapter 4. Writer Sample Plug-in

This chapter provides method and instructions to create a UI plug-in used to demonstrate how to manipulate a writer document programmatically. The sample plug-in presents the following abilities provided by the Lotus Symphony API:

1. Loading documents
2. Adding a shelf view
3. Getting the UNO model of document
4. Creating sections
5. Creating tables
6. Creating user-defined fields

4.1 Introduction to the scenario

For the purpose of understanding the characteristic of various object types of a writer document, we chose creating a Getting Things Done (GTD) document for the development scenario.

GTD is a time management method for productivity success and increased focus. It has the following concepts:

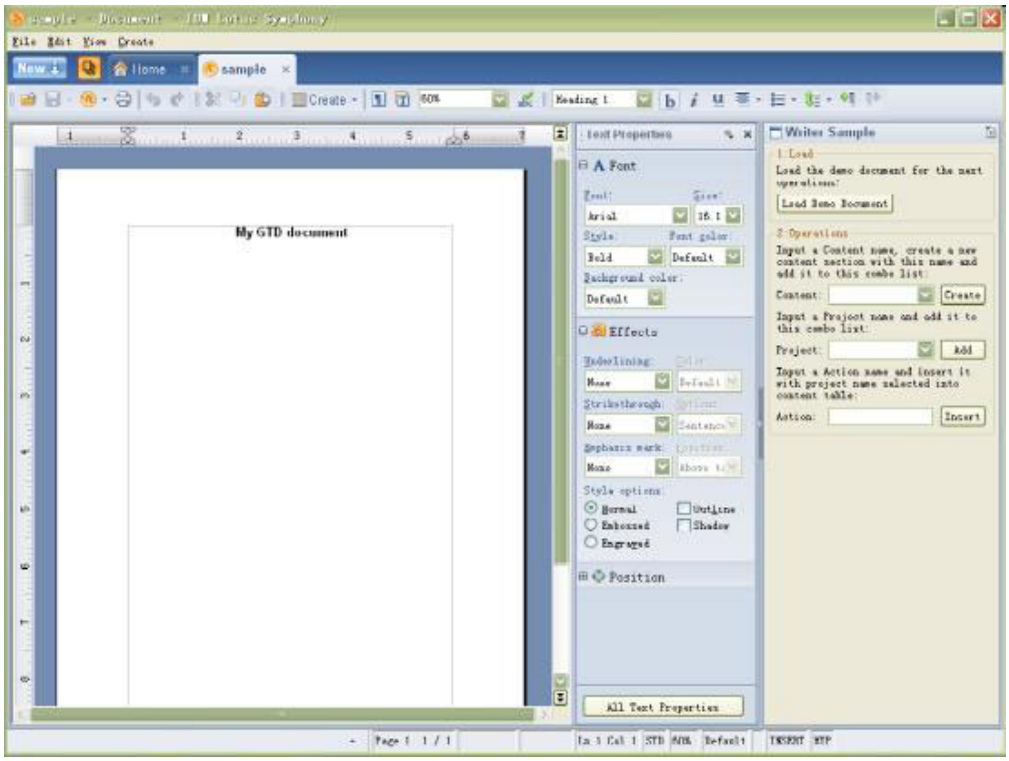
1. Context. A context refers to locations or situations, such as home, computer, work and errands, that are suitable for doing a certain kind of to-dos.
2. Project. A project can be, for example, Repaint bedroom or Review report.
3. Action. An action is a to-do item.

A GTD document is represented as a list of contexts. Each context has a name to identify it and contains a table for actions. Each action refers to a project. Users can manipulate the document in the following ways:

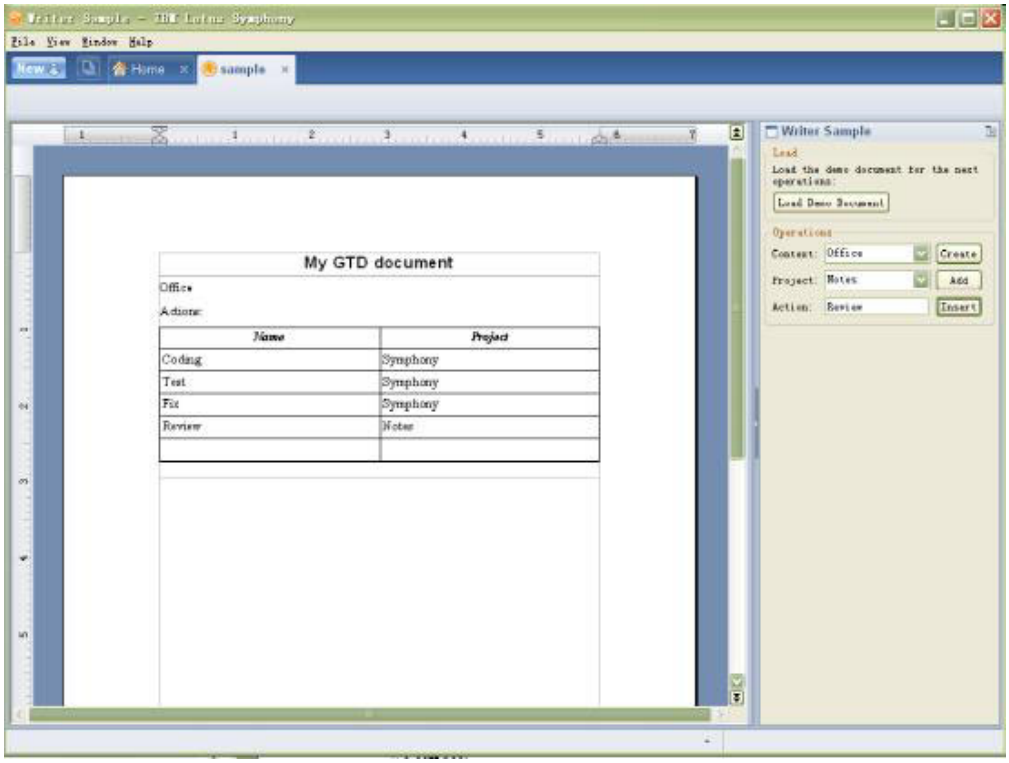
1. Adding contexts
2. Adding projects
3. Adding actions

4.2. Preview of the result

The UI of the plug-in to manipulate the writer document is as follows:



The plug-in creates a GTD document like this:



From this figure we can see that:

1. Each context is represented with a section element of the writer document. In the section, the first line is the name of the context.
2. Actions are represented with a text table element of the writer document. Every row is an action. The second column is its associated project.
3. Each project is represented with a user-defined field, which you can see by double-clicking on a project. A project can be referenced by multiple actions in multiple contexts, and using user-defined fields allows us to change a project name easily.

You can learn the following tasks from this plug-in:

1. Getting the UNO model of a writer document.
2. How to create sections in a writer document and then inserting other types of elements such as text, tables into them.
3. Text Tables: how to create them and then inserting content into their cells.
4. User-defined fields: How to create user-defined fields and insert them into the document.

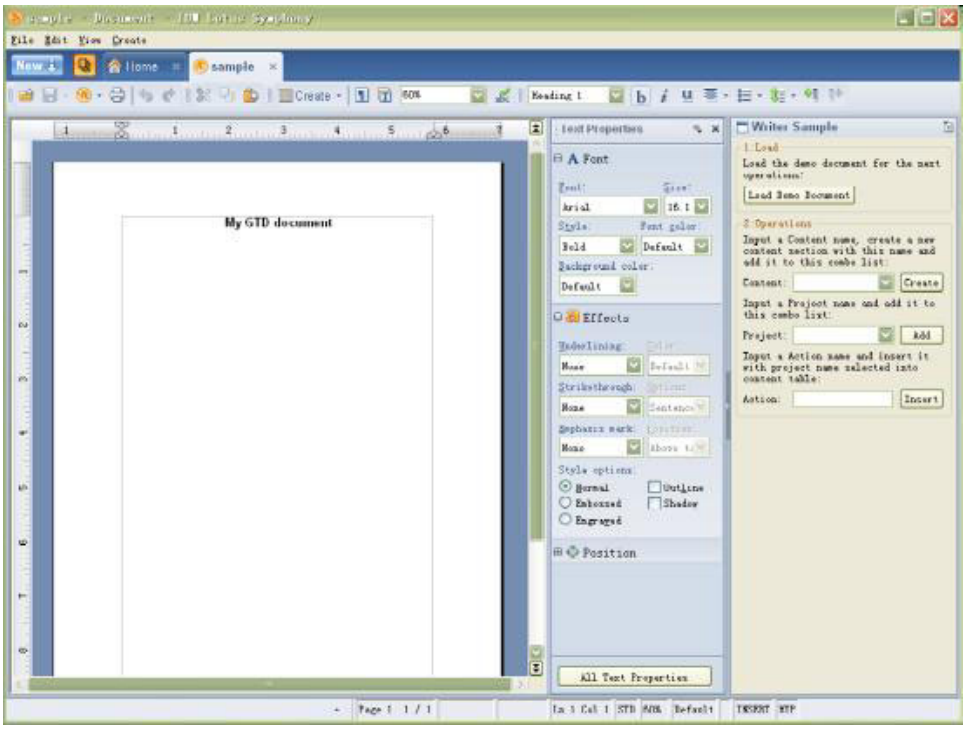
4.3 Deploying the sample

The standard deployment approach described in the developer guide applies to this sample. Refer to Part 4 Chapter 5:Packaging and deploying your plug-ins.

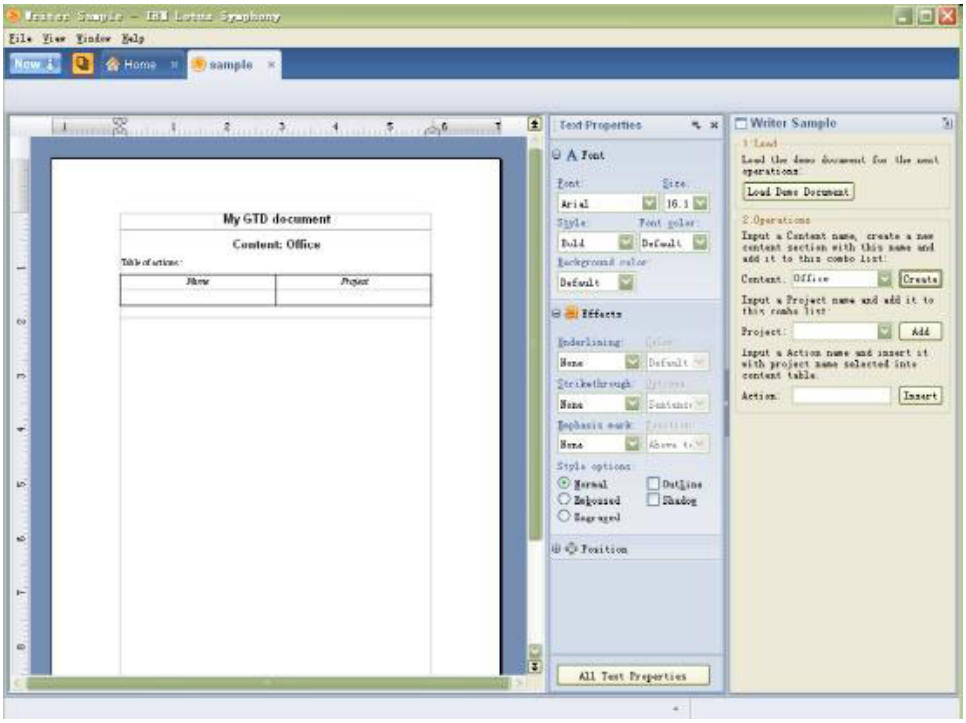
4.4 Using the sample

Launch Lotus Symphony after this plug-in is deployed. You can see a sidebar on the right of the window. The steps to create a GTD document are as follows:

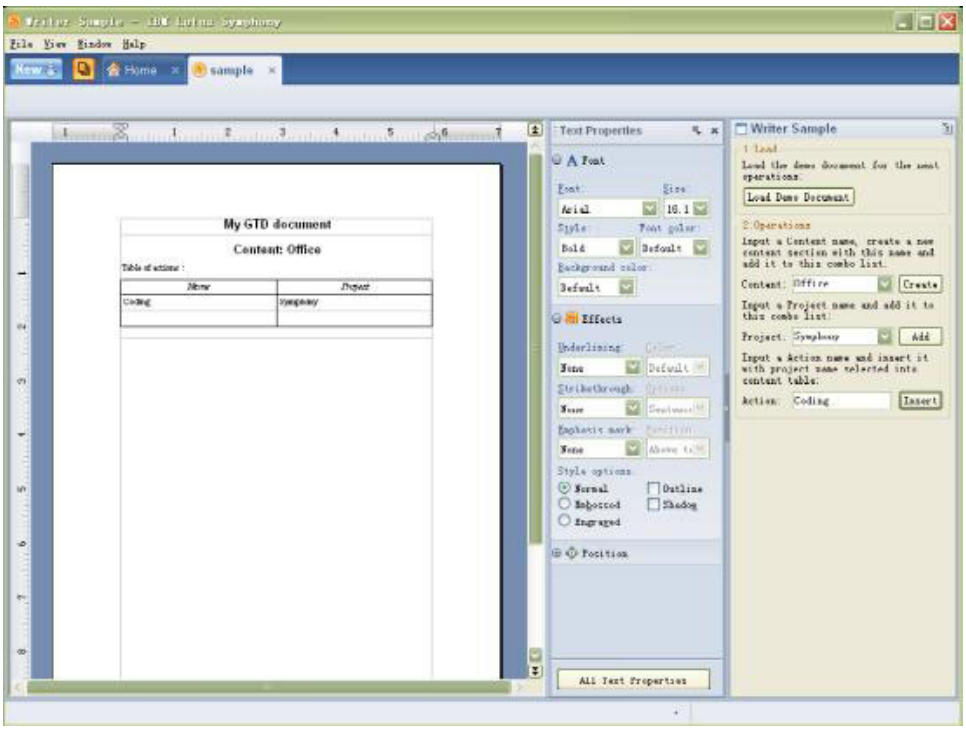
1. Click **Load Demo Document**. An empty GTD document is opened in a new page.



2. Input a name in the **Context** field, such as Office, then click **Add**. An empty context is inserted into the end of the document:



3. Input a name in the **project** field, such as `Lotus Symphony`, then click **Add**. A project is created and displays in the field.
4. Input a name in the **action** field, such as Coding, select a project previously created, and then click **Add**. An action is appended into the last row of the table.



5. You can create more contexts, projects, and actions.

4.5 Building the sample

Prepare your development environment

Refer to Part 4 chapter 1: Setting up the integrated development environment, which shows how to prepare your Lotus Symphony development environment step by step.

Creating the sample

1. Create an empty plug-in named `com.ibm.productivity.tools.samples.writer` .
2. Open `MANIFEST.MF` file. On the **Dependencies** tab, add the following dependent plug-ins:
   - com.ibm.productivity.tools.ui.views
   - com.ibm.rcp.ui
   - com.ibm.productivity.tools.core
3. On the **Extensions** tab, add an extension on the extension point com.ibm.rcp.ui.shelfViews. Change the part of the `plugin.xml` file corresponding to the extension with:

```
<extension
      point="com.ibm.rcp.ui.shelfViews">
   <shelfView
         id="com.ibm.productivity.tools.samples.writer.shelfViewl"
         page="RIGHT"
         region="TOP"
         showTitle="true"
         view="com.ibm.productivity.tools.samples.writer.demoView"/>
</extension>
```

4. Add a view extension by appending the following sample code in the `plugin.xml` file:

```
<extension
      point="org.eclipse.ui.views">
   <view
        id="com.ibm.productivity.tools.samples.writer.demoView"
        name="Writer Sample"
        category="com.ibm.productivity.tools.samples"
        class="com.ibm.productivity.tools.samples.writer.DemoView">
   </view>
</extension>
```

5. Create a class com.ibm.productivity.tools.samples.writer.DemoView, override the createPartControl method to create the controls shown on the plug-in UI. Then add listeners to handle user events. The following sample code snippets are the main methods of the class:

```
/**
 * create this action with the specified project.
 * @param name action name
 * @param projectField
 */
protected void createAction(String name, XPropertySet projectField) {
    String tableKey = contentsCombo.getCombo().getText();
    XTextTable table = (XTextTable) xTables.get(tableKey);

    int lastRow = table.getRows().getCount();
    XCell xCell = table.getCellByName("A" + Integer.toString(lastRow));
    XText cellText = (XText) UnoRuntime.queryInterface(XText.class, xCell);
    cellText.setString(name);
    xCell = table.getCellByName("B" + Integer.toString(lastRow));
    cellText = (XText) UnoRuntime.queryInterface(XText.class, xCell);

    Object oUserField;
    try {
        oUserField = factory
                .createInstance("com.sun.star.text.TextField.User");
        XDependentTextField xUserField = (XDependentTextField) UnoRuntime
                .queryInterface(XDependentTextField.class, oUserField);
        xUserField.attachTextFieldMaster(projectField);

        cellText.insertTextContent(cellText.getStart(), xUserField, false);
        table.getRows().insertByIndex(table.getRows().getCount(), 1);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
/**
     * create a project with the name
     * @param id project's ID
     * @param name
     * @return
     */
    private XPropertySet createProject(String id, Object name) {
        try {
            Object oUserFieldMaster = factory
                    .createInstance("com.sun.star.text.FieldMaster.User");
            XPropertySet xUserFieldMaster = (XPropertySet) UnoRuntime
                    .queryInterface(XPropertySet.class, oUserFieldMaster);
            // Set the name and value of the FieldMaster
            xUserFieldMaster.setPropertyValue("Name", id);
            xUserFieldMaster.setPropertyValue("Content", name);
            projects.add(xUserFieldMaster);
            return xUserFieldMaster;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

```
    /**
     * create this action into the content table
     * @param xSection the section
     * @param name the action's name
     */
    protected void createActionTable(XTextSection xSection, String name) {
        try {
            Object oTable = factory
                    .createInstance("com.sun.star.text.TextTable");
            XTextTable xTable = (XTextTable) UnoRuntime.queryInterface
(XTextTable.class,
                    oTable);
            xTables.put(name, xTable);
            xTextDoc.getText().insertTextContent(xSection.getAnchor().getEnd(),
                    xTable, false);
            XText xCellText = (XText) UnoRuntime.queryInterface(XText.class,
                    xTable.getCellByName("A1"));
            xCellText.setString("Name");
            xCellText = (XText) UnoRuntime.queryInterface(XText.class, xTable
                    .getCellByName("B1"));
            xCellText.setString("Project");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

```
/**
 * create a content with the name
 * @param name the content's name
 * @return a XTextSection object
 */
protected XTextSection createContent(String name) {
    try {
        Object oSection = factory
                .createInstance("com.sun.star.text.TextSection");
        XTextSection xSection = (XTextSection) UnoRuntime.queryInterface(
                XTextSection.class, oSection);
        XNamed xNamed = (XNamed) UnoRuntime.queryInterface(XNamed.class,
                oSection);
        xNamed.setName(name);
        xTextDoc.getText().insertTextContent(xTextDoc.getText().getEnd(),
                xSection, false);
        xTextDoc.getText().insertString(xSection.getAnchor().getStart(),
                "Content: " + name, false);
        xTextDoc.getText().insertControlCharacter(
                xSection.getAnchor().getEnd(),
                ControlCharacter.PARAGRAPH_BREAK, false);
        xTextDoc.getText().insertString(xSection.getAnchor().getEnd(),
                "Table of actions :", false);
        createActionTable(xSection, name);
        return xSection;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

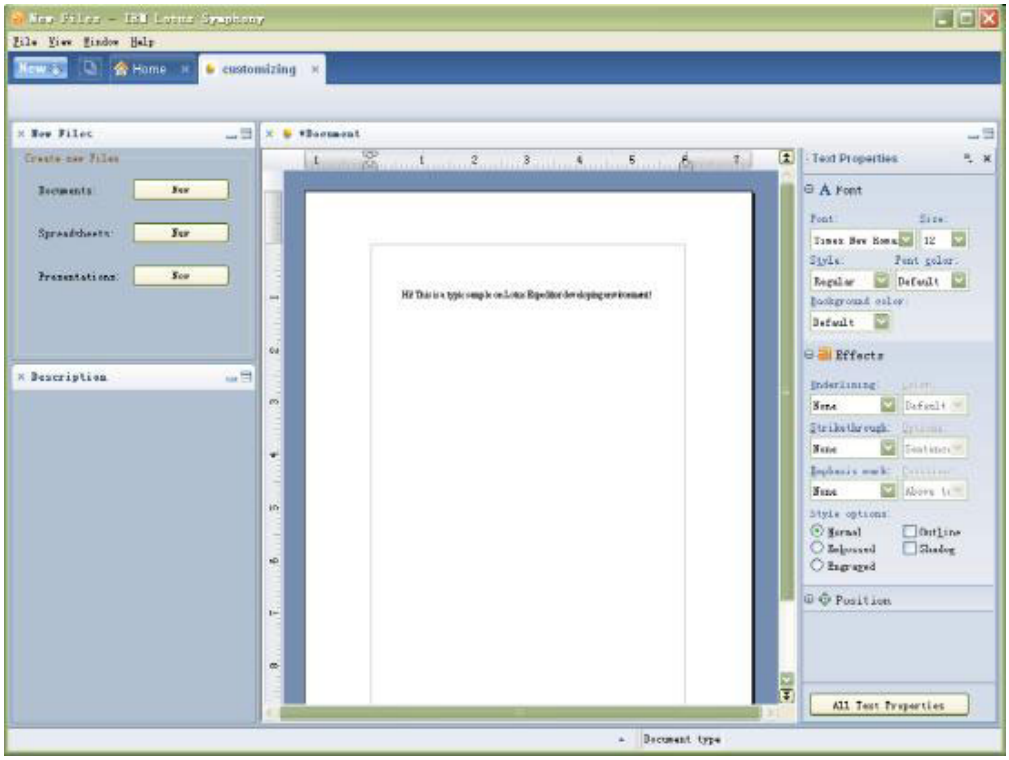
6. Debug and test the sample Eclipse plug-in.
7. Package and deploy the sample plug-in. Refer to Part 4 Chapter 5:Packaging and deploying your plug-ins.

Chapter 5. Customizing a Sample Plug-in

In this sample plug-in,it shows :

1. the ways to use the Lotus Expeditor launch item.
2. a custom perspective with custom views and Symphony views.
3. how to create new Lotus Symphony documents of three kinds repeatedly.
4. how to add a status bar to show the new documents' type.
5. a custom early startup when Lotus Symphony starts up.
6. a custom help document.

The following figure shows this sample plug-in's overview image.



Note: All sample code used within this chapter can be found in the Lotus Symphony development Toolkit, such as `$symphony_sdk/samples/eclipse/plugins/com.ibm.productivity.tools.samples.customizing`. You can get the toolkit from the site: http://symphony.lotus.com.

5.1 Introduction to the scenario

On the Lotus Expeditor platform and in the Lotus Symphony development environment, you might need custom views and Lotus Symphony views at same time. You might need to use a custom view to operate a Lotus Symphony view. You might need other typical Eclipse application and Lotus Expeditor such as a status bar, an early startup, a custom spell checker, or a custom help document.

5.2 Preview of the result

This plug-in first creates a perspective, and then adds three views on this perspective. One view is used for new buttons which creates three new Lotus Symphony documents, the other one view is used to show description, and the third view is used to show multiple Lotus Symphony document views. Then you will add an early startup which is invoked when Lotus Symphony starts up, a status bar, and a sample help topic.

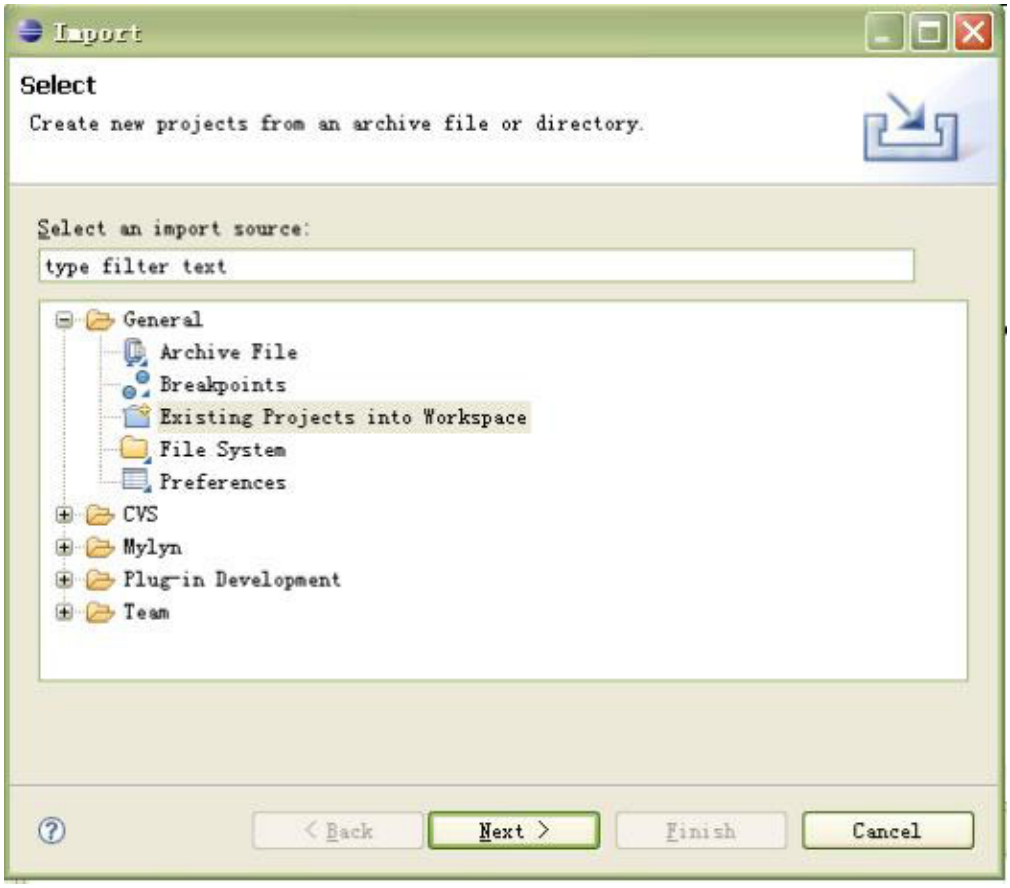
5.3 Prepare development environment

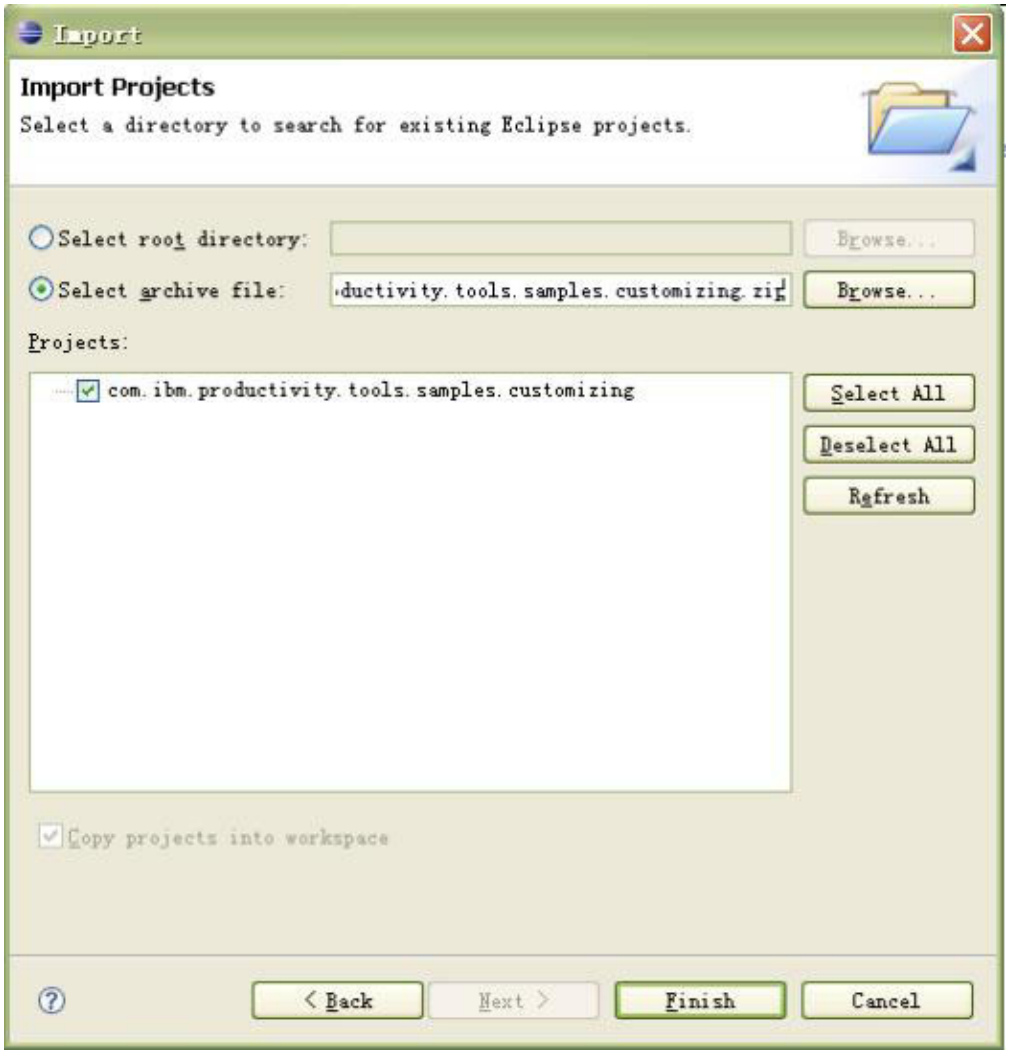
Refer to Part 4 chapter 1: Setting up the integrated development environment, which shows how to prepare your Lotus Symphony development environment step by step.

5.4 Deploying the sample

If you already have this plug-in, you can import it into Eclipse from an existing project using the Eclipse import function. Otherwise, the following sections show

you how to build this plug-in.





5.5 Creating the sample

Create a new plug-in

1. Launch the Eclipse development environment.
2. Click **File > New > Project**.
3. Select **Plug-in Project**, and click **Next**.
4. Type `com.ibm.productivity.tools.samples.customizing` in the **Project name** field. Click **Next**.
5. Type a descriptive name in the **Plug-in Name** field, for example `Customizing sample`.
6. Click **Finish**.

Add the plug-in dependency

The following table lists some of the plug-in dependencies used by the document library. The plug-in names are abbreviated:

Plug-in	Description
org.eclipse.core.runtime org.eclipse.ui	Eclipse core plug-ins
com.ibm.productivity.tools.ui.views com.ibm.productivity.tools.core	Lotus Symphony API plug-in

Perform the following steps to add the plug-in dependency.

1. Click the **Dependencies** tab of the Customizing sample plug-in manifest.
2. Click **Add**.
3. Add the following plug-ins:
   - com.ibm.productivity.tools.ui.views
   - com.ibm.productivity.tools.core
   - com.ibm.rcp.textanalyzer

Adding a perspective and views

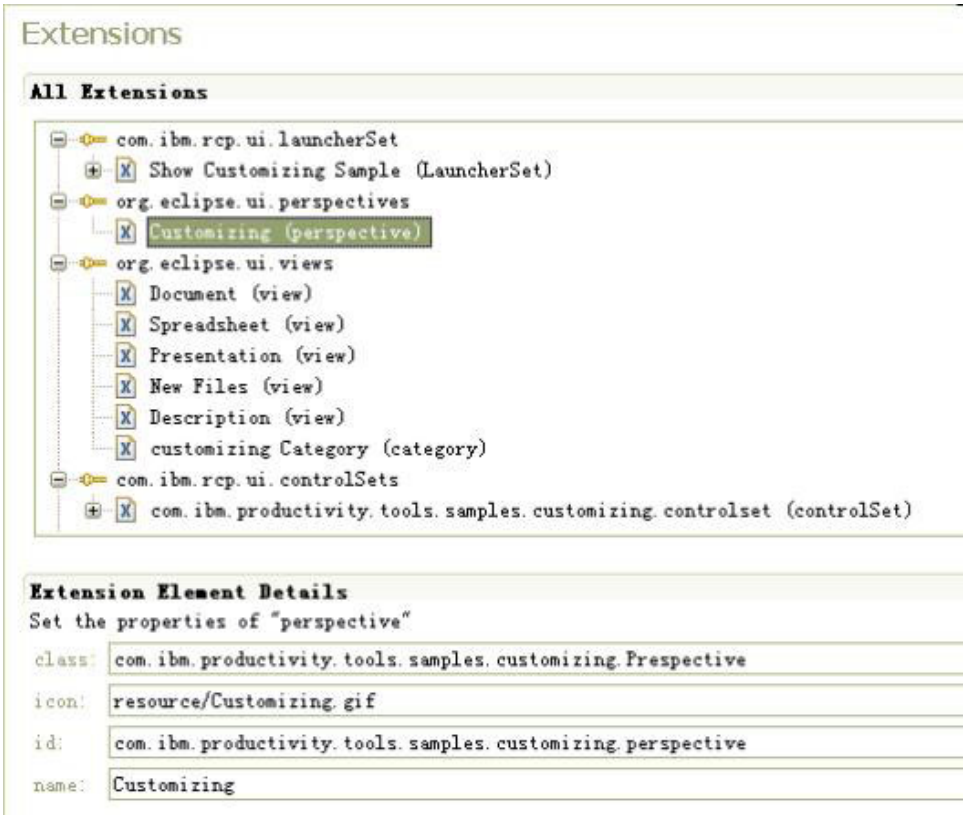
1. Click the **Extensions** tab.
2. Click **Add**.
3. Add the following extension: org.eclipse.ui.perspectives.
4. Click **Finish**.
5. Right-click the added extension and select **New > perspective**.
6. Click the **plugin.xml** tab.
7. Copy and paste the following sample code into the `plugin.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

   <extension
      point="com.ibm.rcp.ui.launcherSet">
   <LauncherSet
         id="com.ibm.productivity.tools.samples.views.LauncherSet"
         label="Show Customizing Sample">
      <perspectiveLaunchItem
            autoStart="false"
            iconUrl="resource/Customizing.gif"
            id="com.ibm.productivity.tools.samples.views.perspectiveLaunchItem"
            label="Show Customizing Sample"
            perspectiveId="com.ibm.productivity.tools.samples.customizing.perspective
           ">
      </perspectiveLaunchItem>
   </LauncherSet>
</extension>

  <extension
        point="org.eclipse.ui.perspectives">
        <perspective
                   class =
          "com.ibm.productivity.tools.samples.customizing.Prespective"
                   icon="resource/Customizing.gif"
                   name = "customizing"
                   id = "com.ibm.productivity.tools.samples.customizing.perspective"

                />
  </extension>
```

```
<extension
        point="org.eclipse.ui.views">
        <view
           category="com.ibm.productivity.tools.samples.customizing"
           allowMultiple="true"
           class="com.ibm.productivity.tools.samples.customizing.view.WriterView"
           id="com.ibm.productivity.tools.samples.customizing.writerview"
           icon="resource/Customizing.gif"
           name="Document">
     </view>

     <view
           category="com.ibm.productivity.tools.samples.customizing"
           allowMultiple="true"
           class="com.ibm.productivity.tools.samples.customizing.view.SpreadsheetView
           "
           id="com.ibm.productivity.tools.samples.customizing.spreadsheetview"
           icon="resource/Customizing.gif"
           name="Spreadsheet">
     </view>

     <view
           category="com.ibm.productivity.tools.samples.customizing"
           allowMultiple="true"
           class="com.ibm.productivity.tools.samples.customizing.view.PresentationVie
           w"
           id="com.ibm.productivity.tools.samples.customizing.presentationview"
           icon="resource/Customizing.gif"
           name="Presentation">
     </view>
```

```
<view
           category="com.ibm.productivity.tools.samples.customizing"
           allowMultiple="true"
           class="com.ibm.productivity.tools.samples.customizing.view.OpenFilesView"
           id="com.ibm.productivity.tools.samples.customizing.openfilesview"
           icon="resource/Openfiles.gif"
           name="New Files">
     </view>

     <view
           category="com.ibm.productivity.tools.samples.customizing"
           allowMultiple="true"
           class="com.ibm.productivity.tools.samples.customizing.view.DescriptionView
           "
           id="com.ibm.productivity.tools.samples.customizing.descriptionview"
           icon="resource/Openfiles.gif"
           name="Description">
     </view>

     <category
           id="com.ibm.productivity.tools.samples.customizing"
           name="customizing Category">
     </category>

  </extension>
```

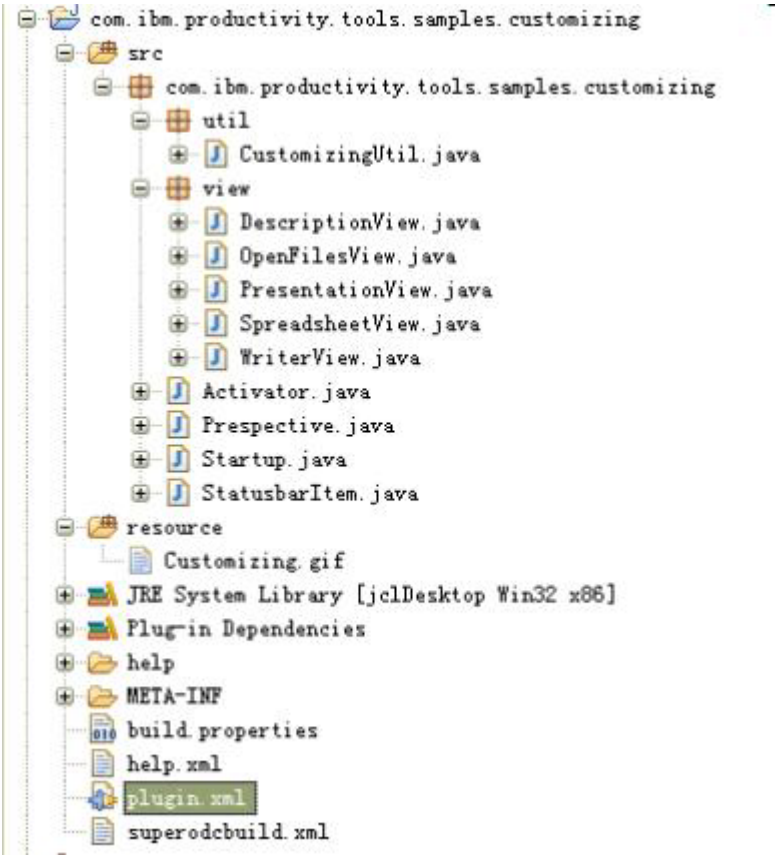
```
   <extension
         point="com.ibm.rcp.ui.controlSets">
      <controlSet
            visible="true"
            id="com.ibm.productivity.tools.samples.customizing.controlset">
         <statusLine
               path="BEGIN_GROUP"
               id="com.ibm.productivity.tools.samples.customizing.statusline">
            <groupMarker name="additions"/>
         </statusLine>
         <control
            statusLinePath="com.ibm.productivity.tools.samples.customizing.statusline
            /additions"
               class="com.ibm.productivity.tools.samples.customizing.StatusbarItem"
               id="com.ibm.productivity.tools.samples.customizing.control"/>
       </controlSet>
   </extension>

   <extension point="org.eclipse.ui.startup">
     <startup class="com.ibm.productivity.tools.samples.customizing.Startup"/>
   </extension>

   <extension point="org.eclipse.help.toc">
    <toc file="help.xml" primary="true"/>
 </extension>

</plugin>
```



8. Create a class named Prespective which implements IPerspectiveFactory. The main method in this class is shown in the following sample code.

```
public void createInitialLayout( IPageLayout layout ) {
            //set editor area to invisible so that the view shows maximized.
        layout.setEditorAreaVisible(false);

        //add the expeditor view to this perspective
        layout.addView(OpenFilesView.VIEW_ID, IPageLayout.LEFT, 0.25f,
            layout.getEditorArea());
        layout.addView(DescriptionView.VIEW_ID, IPageLayout.BOTTOM, 0.4f,
            OpenFilesView.VIEW_ID);
        layout.addView(WriterView.VIEW_ID, IPageLayout.RIGHT, 0.75f,
            layout.getEditorArea());
    }
```

The method createInitialLayout () specifies the layout of the views on the page.



5.6 Core code demonstration

The Following section shows the core code snippet for the function. For more details, refer to the sample code.

1. Add a launcher item to launch a perspective.

   First, add the extension point com.ibm.rcp.ui.launcherSet, then add a new perspectiveLaunchItem and set this item's perspectiveId attribute value as the perspective's id which will be launched.

2. Add a custom view and Lotus Symphony views.

   To add a custom view, refer to the Eclipse org.eclipse.ui.views extension point reference. For Lotus Symphony views, refer to Part 5 Section 2.3 Chapter 2.

3. Add a status bar.

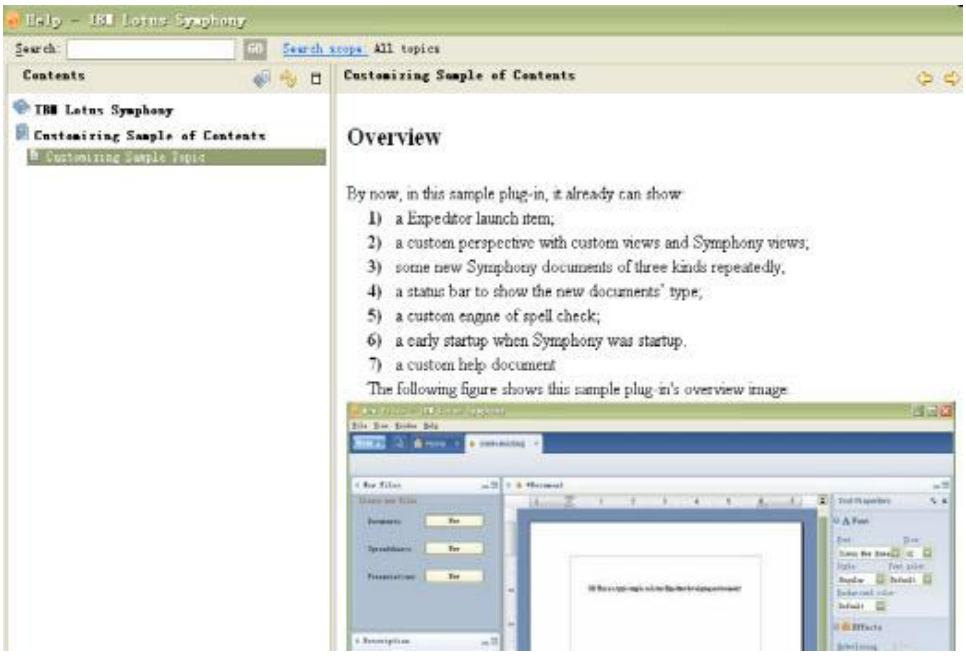
   Refer to Part 4 Section 2.6 Chapter 2.

4. Add a custom early startup.

   First, add the extension point org.eclipse.ui.startup, then create a class named StartUp which implements org.eclipse.ui.IStartup.

```
<extension point="org.eclipse.ui.startup">
     <startup
       class="com.ibm.productivity.tools.samples.customizing.Startup"/>
   </extension>
```

5. Add a custom help topic:



First, add the extension point org.eclipse.help.toc, then specify the toc file which defines the custom help file, as shown in the following sample code:

```
<extension point="org.eclipse.help.toc">
   <toc file="help.xml" primary="true"/>
</extension>
```

The following sampe code shows the content of the toc file.

```
<?xml version="1.0" encoding="UTF-8"?>
<?NLS TYPE="org.eclipse.help.toc"?>

<toc label="Customizing Sample of Contents">
    <topic
       label="Customizing Sample Topic"  href="help/help.htm">
    </topic>
</toc>
```

5.7 Extending the sample

Next, you can add a custom dictionary for spell check. You can add activities which are assigned a name and description that provide information about the activity.

Chapter 6. Convertor Sample Plug-in

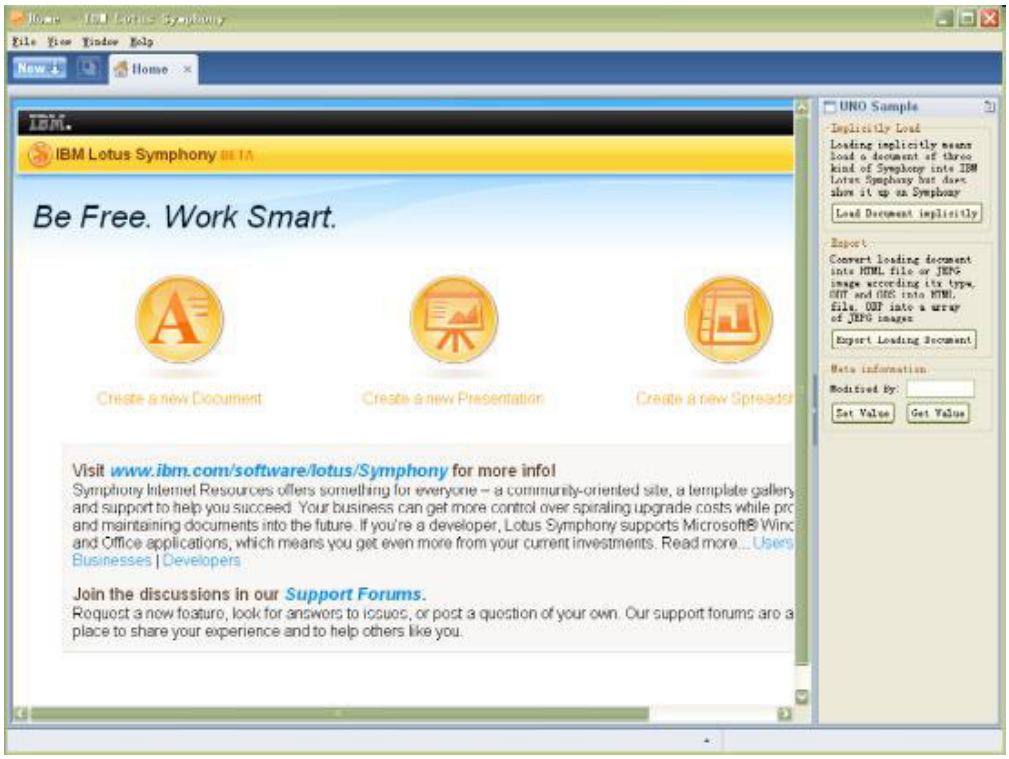
This plug-in sample shows a typical application of loading three kinds of Lotus Symphony documents implicitly, which means loading a document into Lotus Symphony but does not show it up on Lotus Symphony. The sample will export this loaded document into HTML or JPEG format according its type. A sample operation of accessing the meta-data of the document, to set and get a name will modify this document.

Note: All sample code used within this chapter can be found in the Symphony developing Toolkit, such as $symphony_sdk/samples/eclipse/plugins/com.ibm.productivity.tools.samples.convertor. You can get this Toolkit from the site: http://symphony.lotus.com.

In this sample plug-in, it shows how to create:

1. A simple side shelf.
2. A button for loading documents implicitly.
3. A button for exporting and converting the loaded document into an HTML file or JPEG image according its type: ODT and ODS into the HTML file or ODP into JPEG image array.
4. A simple set and get operation to show how to access metadata.

The following figure shows this sample plug-in's overview image.



6.1 Introduction to the scenario

You might want to load a Lotus Symphony document with its path, or you want to load documents implicitly and convert Lotus Symphony documents into a different type. You might also need to change some metadata of the document.

6.2 Preview of the result

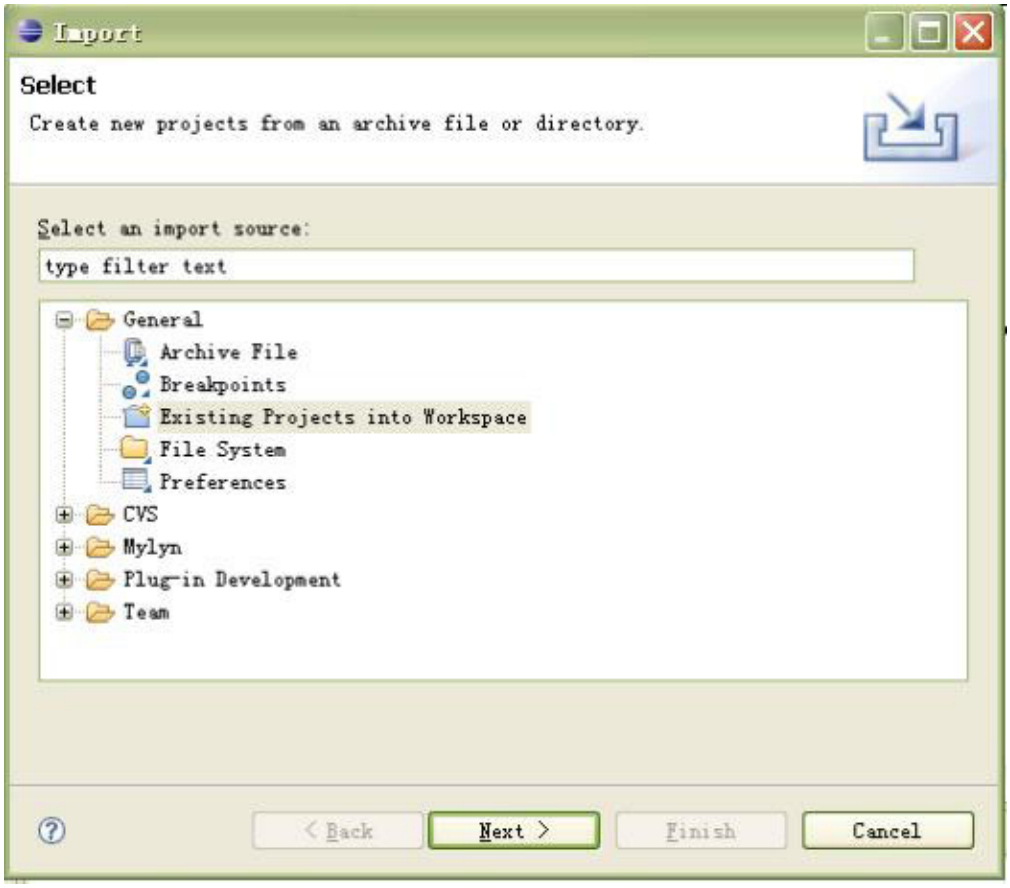
According to the scenario above, this plug-in first creates a side shelf, and then adds a button to load a document by its path implicitly, then it adds a button to export this loaded document into a HTML file or JPEG image, and adds two buttons to set and get this document's metadata of modified name.

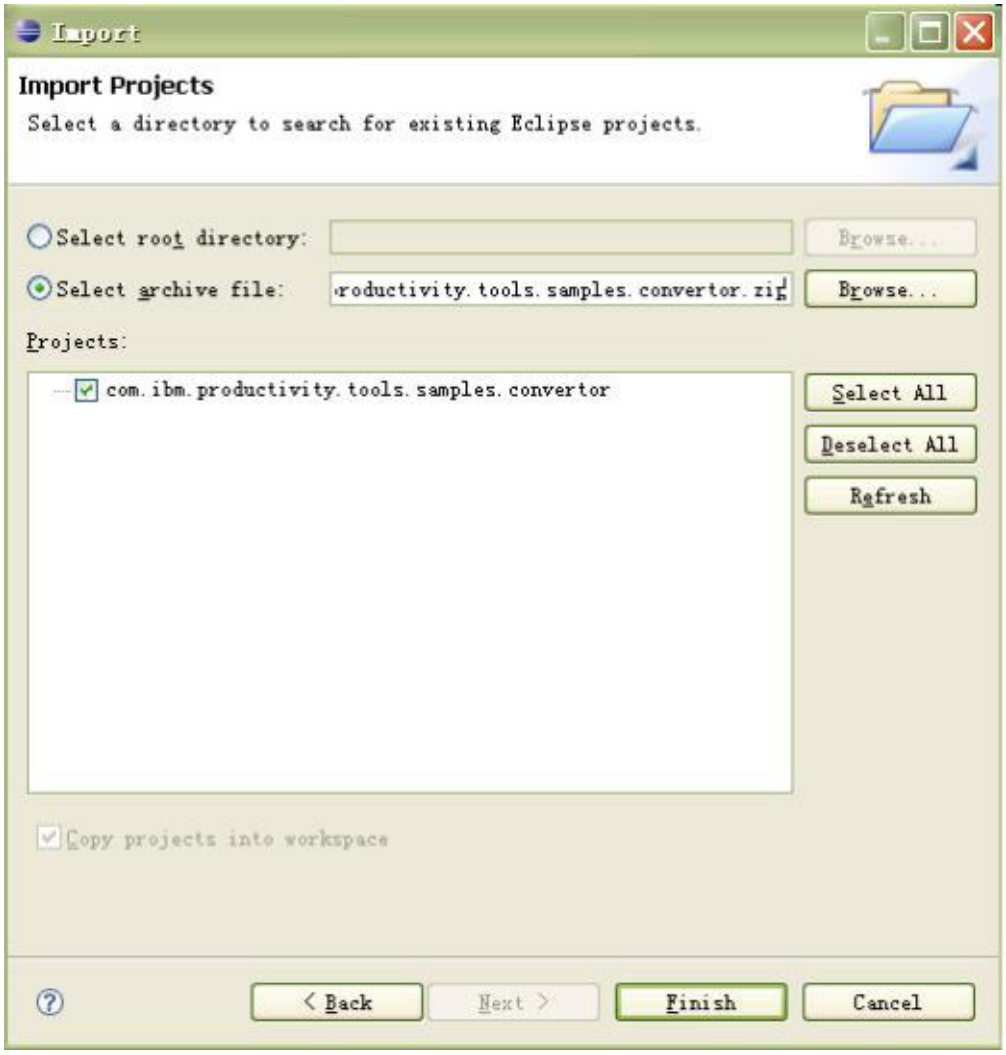
6.3 Prepare development environment

Refer to Part 4 chapter 1: Setting up the integrated development environment, which shows how to prepare your Lotus Symphony development environment step by step.

6.4 Deploying the sample

If you already have this plug-in, you can import it into Eclipse from an existing project using the Eclipse import function. Otherwise, the following sections show you how to build this plug-in.





6.5 Design overview

This sample has these goals:

1. Add a side shelf.
2. Add two groups to load implicitly and export.
3. Add a group to change the document's metadata.

6.6 Creating the sample

Creating a new plug-in

1. Launch the Eclipse development environment.
2. Click **File > New > Project**.
3. Select **Plug-in Project**, and click **Next**.
4. Type `com.ibm.productivity.tools.samples.convertor` in the **Project name** field. Click **Next**.
5. Type a descriptive name in the **Plug-in Name** field, for example, `Convertor sample`.
6. Click **Finish**.

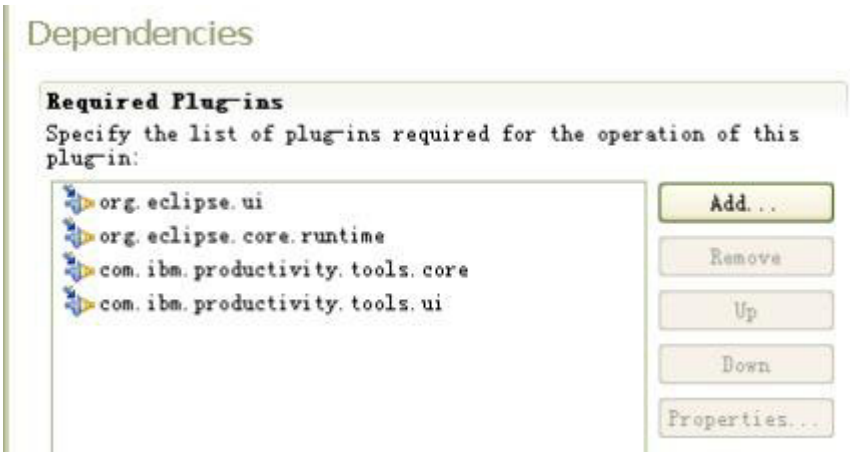
Adding the plug-in dependency

The following table lists some of the plug-in dependencies used by the document library. The plug-in names are abbreviated:

Plug-in	Description
org.eclipse.core.runtime org.eclipse.ui	Eclipse core plug-ins
com.ibm.productivity.tools.ui.views com.ibm.productivity.tools.core	Lotus Symphony API plug-ins

Perform the following steps to add the plug-in dependency.

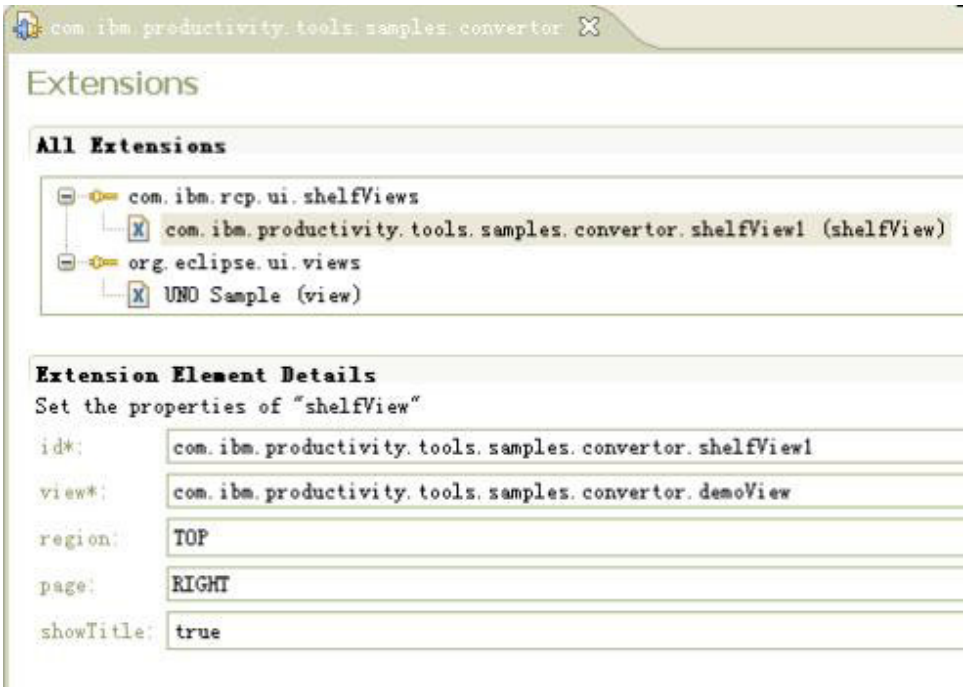
1. Click the **Dependencies** tab of the Convertor sample plug-in manifest.
2. Click **Add**.
3. Add the following plug-ins:
   - com.ibm.productivity.tools.ui.views
   - com.ibm.productivity.tools.core

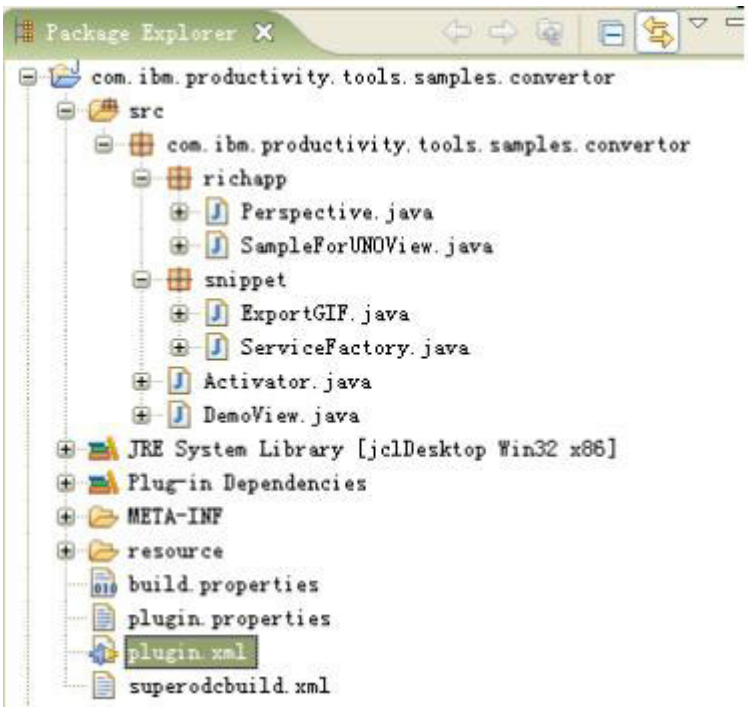


Adding shelf views

1. Click the **Extensions** tab.
2. Click **Add**.
3. Add the following extension: org.eclipse.ui.views.
4. Click **Finish**.
5. Right-click the added extension and select **New > view**.
6. Click the **plugin.xml** tab.
7. Copy and paste the following sample code into the `plugin.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
<extension
      point="com.ibm.rcp.ui.shelfViews">
   <shelfView
         id="com.ibm.productivity.tools.samples.convertor.shelfView1"
         page="RIGHT"
         region="TOP"
         showTitle="true"
         view="com.ibm.productivity.tools.samples.convertor.demoView"/>
</extension>
<extension
      point="org.eclipse.ui.views">
   <view
        id="com.ibm.productivity.tools.samples.convertor.demoView"
        name="UNO Sample"
        category="com.ibm.productivity.tools.samples"
        class="com.ibm.productivity.tools.samples.convertor.DemoView">
   </view>
</extension>
</plugin>
```





6.7 Core code demonstration

The following section shows core code snippets for the function. For details, refer to this sample code.

1. Get the com.sun.star.lang.XMultiServiceFactory object reference. Refer to Getting the global service factory.
2. Load the Lotus Symphony document by file path implicitly.

   The following sample code shows how to load the Lotus Symphony document implicitly.

```
protected void loadDocumentImplictly(String filePath) {
                XMultiServiceFactory xServiceFactory = getServiceFactory();
                try {
                    Object object = xServiceFactory
                            .createInstance("com.sun.star.frame.Desktop");
                    XComponentLoader loader = (XComponentLoader) UnoRuntime
                            .queryInterface(XComponentLoader.class, object);
                    PropertyValue[] aArgs = new PropertyValue[1];
                    aArgs[0] = new PropertyValue();
                    aArgs[0].Name = "Hidden";
                    aArgs[0].Value = new Boolean(true);
                    sourceURL = "file:///" + Path.fromOSString
            (filePath).toPortableString();
                    object = loader.loadComponentFromURL(sourceURL, "_blank",
                            FrameSearchFlag.CREATE, aArgs);
                    xDocument = (XComponent) UnoRuntime.queryInterface(
                            XComponent.class, object);
                } catch (com.sun.star.io.IOException e) {
                    e.printStackTrace();
                } catch (IllegalArgumentException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
```

Create the implicit loading control by using the property named hidden and set it to true.

3. Resolve the document type.

   The following sample code shows how to resolve document type.

```
protected void resolveDocument() {
                XServiceInfo xInfo = (XServiceInfo) UnoRuntime.queryInterface(
                        XServiceInfo.class, xDocument);
                if (xInfo != null) {
                    if (xInfo.supportsService("com.sun.star.text.TextDocument")) {
                        filter = new String("HTML (StarWriter)");
                    } else if (xInfo.supportsService("com.sun.star.text.WebDocument"))
            {
                        filter = new String("HTML");
                    } else if (xInfo
                            .supportsService
            ("com.sun.star.sheet.SpreadsheetDocument")){
                        filter = new String("HTML (StarCalc)");
                    } else if (xInfo                    .supportsService
            ("com.sun.star.presentation.PresentationDocument")) {
                    // do something
                    }
            }
```

4. Export the documents into a HTML file.

Refer to Exporting documents and drawing objects.

5. Export the document into a JPEG image

   Refer to Exporting documents and drawing objects .

6.8 Extending the sample

Next, you can add an auto-recognizer, and use this function to convert ODP file to SWF file.

Part 7. Troubleshooting and Support

Most of the troubleshooting information for the Lotus Expeditor toolkit is also useful for Lotus Symphony developers. It involves a lots of known issues and solutions for Lotus Expeditor developers. You can find the information from Eclipse, **Help > Help content > Lotus Expeditor Troubleshooting and support**.

In following chapters, you will find some typical issues and solutions. If you have more questions, contact support at Lotus Symphony Web site http://symphony.lotus.com.

Chapter 1. Troubleshooting the Development Environment

Problem: When you setup your development environment, Lotus Symphony does not run.

Solution: Check the development tools that you are using, and following the process in Part 4 Chapter 1. If you are using another tool or version, you can have unexpected errors. Make sure that you have correctly installed:

1. Eclipse 3.2.2
2. Lotus Expeditor toolkit 6.1.2
3. Lotus Symphony profile tool from the Lotus Symphony toolkit

Chapter 2. Troubleshooting During Application Development

Problem: As you develop Lotus Symphony applications, if there are UNO calls within your code, sometimes Lotus Symphony hangs when the code is being executed.

Solution: Create a new job for UNO calls, especially for the functions which are invoked by Lotus Symphony backend. For example, the code within a listener which is added to Lotus Symphony backend. The sample code would look like the following:

```
Job job = new Job("Your job") {
                public IStatus run(IProgressMonitor progress) {
                    //your code comes there
                    return Status.OK_STATUS;
                }
            };
job.schedule();
```

Chapter 3. Troubleshooting During Deployment

Problem: Your application works fine in the development environment, but after you deploy it into Lotus Symphony, when Lotus Symphony is launched, your application does not work correctly.

Solution: Perform the following steps to resolve the problem:

1. Ensure that you are using the Lotus Symphony profile in the development phase. For example, the default VM used by Lotus Symphony is jclDesktop. If you have not upgraded the VM to J2SE, you should use the VM in development phase. The target platform should be the Lotus Symphony installation directory.
2. Check the `$SymphonyDir\data\applications` directory to ensure that your plug-ins are installed successfully. Go through the feature and plug-in directory one by one, to check if there are missing files.
3. Check the platform details when Lotus Symphony runs. Click **Help > About IBM Lotus Symphony**, check the Feature Details, Plug-in Details and Configuration Details. You should be able to find your applications in the list. Configuration Details marks the status for each plug-in. If the status is unexpected for your plug-ins, perhaps you will find out the root cause.
4. Check the log file for unexpected exceptions.The log files are located in `$SymphonyDir\data\logs.Check` to see if there are exceptions.
5. Contact support if the problem remains.

Chapter 4. Contacting Support

To contact support, you can post problems in the Lotus Symphony forum. Include the screen captures of error, all the log files, or platform configuration information which will be helpful to identify issues.

The log files are available in the `$SymphonyDir\data\logs` directory.

The platform configuration information is available from **Help > About IBM Lotus Symphony > Configuration Details**.

Part 8. Appendixes

Appendix . References

For Lotus Expeditor and Lotus Expeditor toolkit, refer to the following sites:

http://www.ibm.com/software/lotus/products/expeditor/

http://www-128.ibm.com/developerworks/lotus/products/expeditor/

For Lotus Notes 8, refer to the following site:

http://www-306.ibm.com/software/lotus/products/notes/

For composite applications, refer to the following sites:

http://www.ibm.com/developerworks/lotus/composite-apps/

http://www-306.ibm.com/software/lotus/products/notes/compositeapplications.html

Appendix . Notices

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of

performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, include a copyright notice as follows:

(C) (your company name) (year). Portions of this code are derived from IBM Crop. Sample Programs. (C) Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information soft copy, the photographs and color illustrations may not appear.

List of Trademarks

These terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
AIX
DB2
DB2 Universal Database Domino
Domino
Domino Designer
Domino Directory
i5/OS
Lotus
Lotus Notes
Notes
OS/400

Sametime
WebSphere

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Printed in USA