IBM Lotus Symphony Developer’s Guide

Note: Before using this information and the product it supports, read the information in Notice page.

Symphony Beta 4 Edition (Jan 2008)

This edition applies to release Beta 4 of IBM Lotus Symphony toolkit (license number L-AENR-7AYBE2) and to all subsequent releases and modifications until otherwise indicated in new editions.

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 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

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

Chapter 3. Accessing the toolkit on the web

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.

Chapter 4. Symphony toolkit

4.1 Samples

The examples found in this guide can be run directly from Eclipse development environment. For instructions on accessing and running the samples, refer to the tutorials within the toolkit.

4.2 Convention

The following convention is used for sample code in this guide:

```
ProgramExample
```

4.3 Related documentation

- *Lotus Symphony Java Toolkit Javadoc Reference*
- Lotus Symphony Java Toolkit Tutorial

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 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.

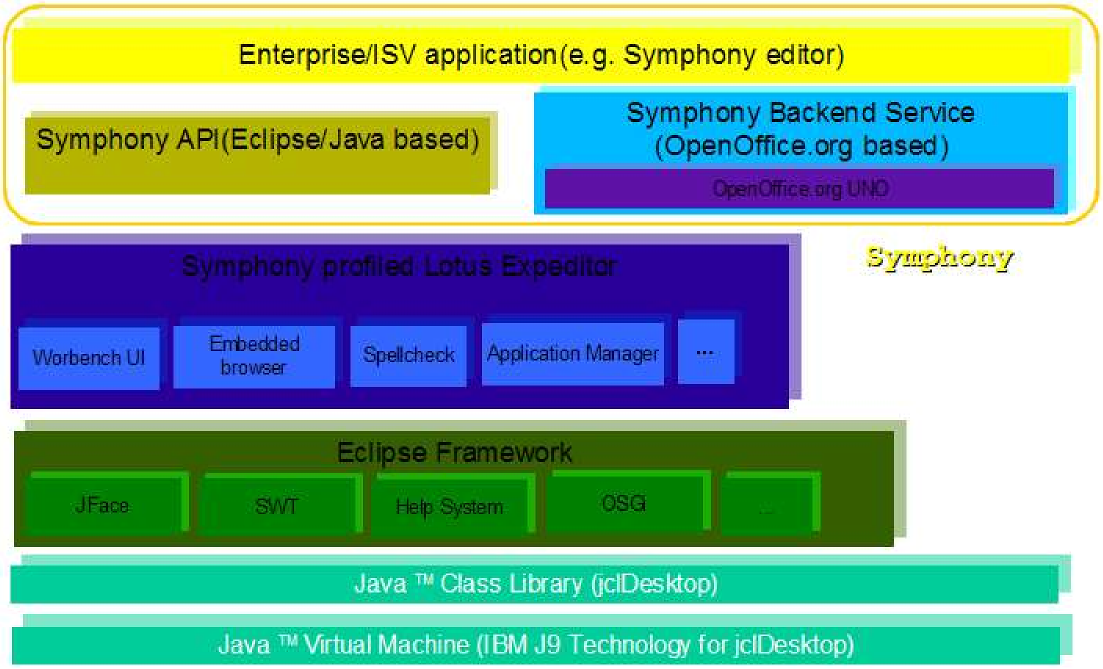
Chapter 2. Lotus Symphony architecture

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.

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


Enterprise/ISV application(e.g. Symphony editor)
Symphony API(Eclipse/Java based)
Symphony Backend Service (OpenOffice.org based)
OpenOffice.org UNO
Symphony
Symphony profiled Lotus Expeditor
Worbench UI
Embedded browser
Spellcheck
Application Manager
...
Eclipse Framework
JFace
SWT
Help System
OSGi
...
Java ™ Class Library (jclDesktop)
Java ™ Virtual Machine (IBM J9 Technology for jclDesktop)

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 which 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 contribute to or 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 Eclipse platform.

2.2 Overview of Eclipse

Eclipse was originally 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.

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 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 end-to-end 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 Client for Desktop 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 and 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	Spell check functionality is used to check misspelled words in document. It is based on Text Analyze framework.
Personalities	Personalities define 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	The 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 Eclipse platform.
Spell check	Spell check functionality is used to check misspelled words in document. It is based on Text Analyze framework.

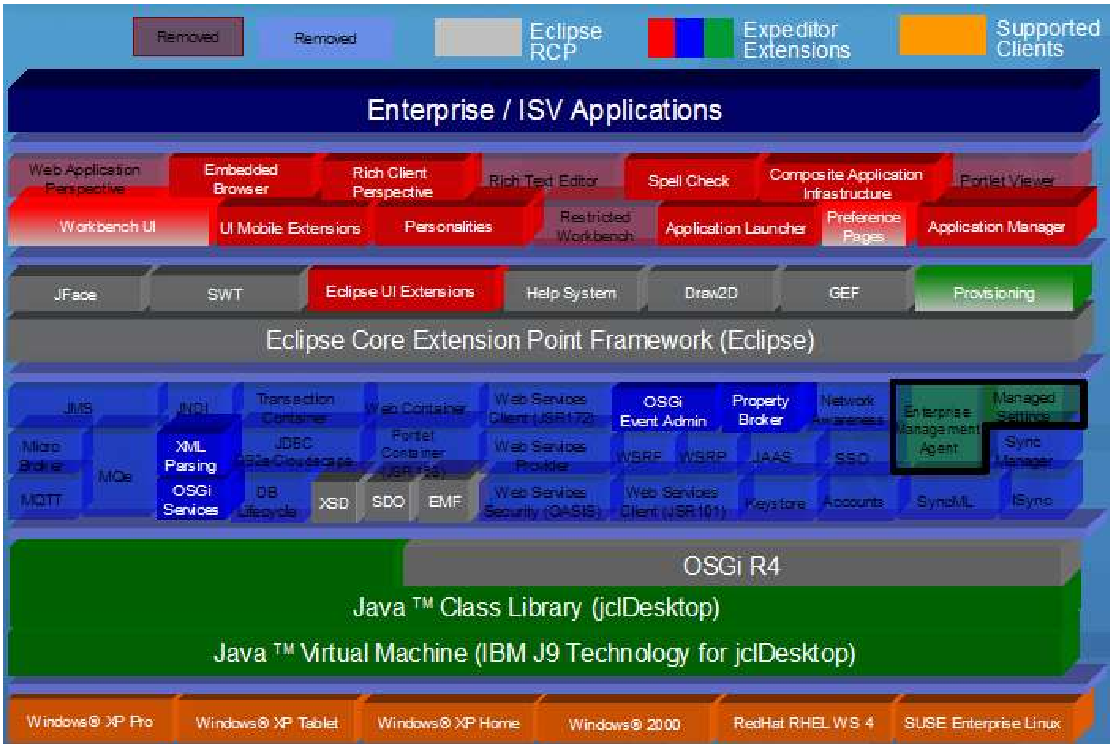
2.3.1 J9 JCL Desktop

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 references a class or package (such as AWT or Swing) that is not included in the J9 VM.

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 Expeditor platform, such as Web Application Perspective, Portlet Viewer, WSRP, and SSO. The Lotus Symphony profiled 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^{T} 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 utilize 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

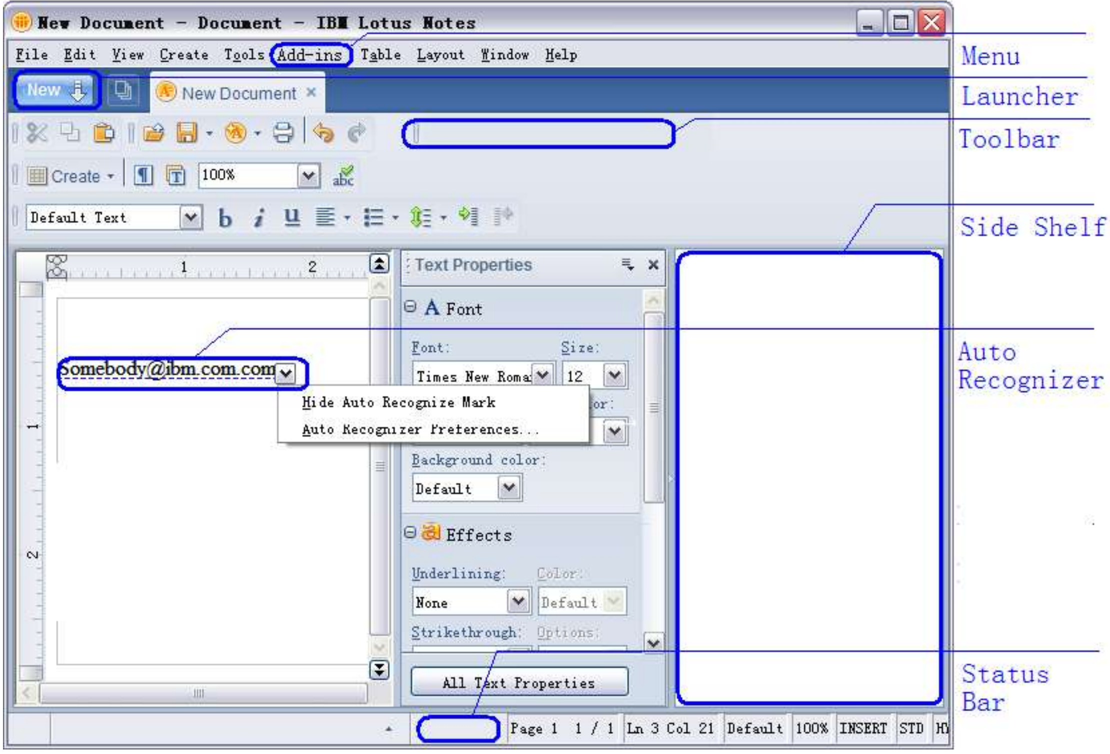
Lotus Symphony is the combination of Eclipse-based Lotus Expeditor and OpenOffice.org. Both of these products provide rich APIs for application integration. In Lotus Symphony, the OpenOffice.org window is re-parented to a SWT control in Eclipse. Most of the user interface contribution items are provided 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/Expeditor extension points and plug-in framework.
- **Document content level API** is based on OpenOffice.org UNO capability.
- **Lotus Symphony API** focuses on the integration between OpenOffice.org and Eclipse/Expeditor.
- **Add-in mechanism** is based on Expeditor Application Manager.

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

The following screen capture shows the user interface items.



Part 3. Extending Lotus Symphony

Chapter 1. Setting up the integrated development environment

The integrated development environment (IDE) is based on Eclipse 3.2 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.

1. Install Lotus Symphony and Eclipse 3.2.
   a) Download Eclipse 3.2 and Lotus Symphony.
   b) Unpack Eclipse 3.2 to a local disk, for example, D:\eclipse as <ECLIPSE_HOME>.
   c) Install Lotus Symphony to a local disk, for example, D:\Lotus\Lotus Symphony as <SYMPHONY_HOME>.
2. Install the J9 launching plug-in and the J9 SDK.
   - Download Lotus Expeditor Toolkit zip file from the site: http://www14.software.ibm.com/webapp/download/nochargesearch.jsp?q=Lotus+Expeditor+Toolkit.
   - Extract the downloaded zip file; copy the J9 JDT launching plug-in ZIP file from \Expeditor\C10FQML\Expeditor_Toolkit_install\plugins\, such as org.eclipse.jdt.launching.j9_6.1.1.jar, from directory plugins just unzipped to the Eclipse subdirectory in the directory where you installed Eclipse, for example, <ECLIPSE_HOME>\plugins\.
   - Extract and unzip the contents of the downloaded J9 SDK zip file from \Expeditor\C10FQML\Expeditor_Toolkit_install\plugins\, such as com.ibm.pvc.wct.runtimes.jcl.desktop.sdk.win32.x86_6.1.1.200707311521 on Windows platform, from directory plugins just unzipped to a directory, such as D:\J9_SDK\.

   Note 1: There is also another way to install J9 SDK for the Symphony based development, to see details, follow the site: http://www-128.ibm.com/developerworks/lotus/library/expeditor-toolkit/.
3. Start the Eclipse IDE.
   Make sure that it is starting up in a new or freshly cleaned workspace. Remnants from prior installations might cause problems, so create a new workspace for each installation.
4. Set the target platform.
   a) Select **Window > Preferences.**

   b) In the left panel, select **Plug-in Development > Target Platform.**
   c) In the location field, click **Browse** and select the Eclipse under the Lotus Symphony installation root directory, for example, <SYMPHONY_HOME>\framework\eclipse.
   d) Click **Reload**.
5. Set up the J9 JRE runtime.
   a) Select **Window > Preferences**.
   b) In the left panel, select **Java > Installed JREs**.
   c) Click Add and enter the following settings in the window:
      JRE type: J9 VM
      JRE name: JCL Desktop
      JRE home directory: <J9_SDK>\jre
      Default VM arguments: -jcl:max
   d) Click **OK** to return to the Installed JREs window, and then select the configuration you just created (JCL Desktop).
   e) Click **Edit** to access the Edit JRE window.
   f) Click **Add External JARs** and browse to `<JRE_HOME>\lib\jclmax\ext`, where <JRE_HOME> is the home directory of J9 SDK(<J9 SDK>\jre).
   g) Select all the JAR files listed and click **Open**. The JAR files display under the JRE system libraries in the Edit JRE window.
   h) Click **Add External JARs** again and browse to `<JRE_HOME>\lib\jclmax\opt-ext`. Select all the files, and click **Open** to return to the Edit JRE window.
   i) Click **OK** to return to the Installed JREs window.
   j) Select the JCL Desktop option to make it the default.

   Note 2**:** When can't get through with d) or finished step 5, select the JRE installed in this step and click the **Edit** to make sure that the JAR files in the directory of `<JRE_HOME>\lib\jclmax\ext and <JRE_HOME>\lib\jclmax\opt-ext` are really added to the JRE system libraries' list in the **`Edit JRE`** dialogue window. If not, select the **JCL Desktop** and click **edit,** then repeat the step f) to i) in this step 5 at **`Edit JRE`** dialogue window.
6. Create your own project code in this Eclipse workspace.
7. Create a new runtime configuration in the Eclipse IDE:
   a) Select **Run** > **Run…**.
   b) In the left panel, select **Eclipse Application** from the configuration list.
   c) Right-click and select **New**.
   d) Enter a name, for example, Lotus Symphony.
   e) Select **Clear workspace data before launching**.
   f) Under **Program to Run**, select **Run a product**.
   g) For **Run a product**, select **com.ibm.productivity.tools.standalone.branding.productivitytools** from the product list.
   h) In the **Runtime JRE**, select **JCL Desktop** which was created in step 5.

i) Select the **Arguments** tab, and set the arguments as described in the following steps:

Set the **Program arguments**:

```
-personality com.ibm.productivity.tools.standalone.personality
-debug
-console
```

Set the **VM arguments**:

```
-Xbootclasspath/a:${rcp_home}/rcp/eclipse/plugins/com.ibm.rcp.base_${rcp_version}/
rcpbootcp.jar
```

Set the variables for arguments:

i) Click **Variables**.

ii) On the **Select Variable** window, click **Edit Variables**.

iii) On the **Preferences** window, click **New**.

iv) Enter `rcp_home` for the name.

Note 3**:** If the dir path contains space character, sometime it may make the Eclipse can’t parse the path and find the file rcpbootcp.jar correctly. If this happens when you finished all of steps and launch the Symphony , you can change the install directory of the Symphony or change its long name to a short name to avoid space character, such as change “Program Files” to “Progra~1”.

v) Enter the path where you installed Lotus Symphony for the value, for example, <SYMPHONY_HOME>`\framework`.

Note 4**:** All path separators slash (/) in directories listed in the following arguments can also be backslash (\) on Windows, but the slash must be used on Linux systems.

There are also several variables in the argument strings, which are resolved in next step.

vi) Click **OK** to finish adding this variable.

vii) Click **New** again.

viii) Enter `rcp_version` for another variable.

ix) Enter the current version number of the **com.ibm.rcp.base** plug-in. You can find its exact version number from your installed Lotus Symphony directory. For example, if the installed plug-in is located in directory <SYMPHONY_HOME>`\framework\rcp\eclipse\plugins\com.ibm.rcp.base_6.1.0.0-200701121347`, then its version number is 6.1.0.0-200701121347.

x) Click **OK** to finish adding this variable.

xi) On **Preferences** window, click **OK** to finish adding variables.

xii) Click **Cancel** on the **Select Variable** window.

Note 5: If you select **OK** on **Select Variable** window, the first variable "${build_project}" in the list is added automatically to **Program arguments field**, and it causes a launching error. In this case, delete ${build_project} from this field, and continue again.

j) Select **Close** to finish creating the new configuration.

8. To run or debug, select **Run > Run** or **Debug > Debug**.

Note6: Please use Java compiler 1.4 as plugins' Java compiler. Java compiler 5.0 may not work correctly.

Note 7: On Redhat system, sometimes a java.lang.UnsatisfiedLinkError exception is thrown when launching the Lotus Symphony. Try to fix it with the command like the following:

```
lddconfig /opt/ibm/lotus/Symphony/framework/shared/eclipse/plugins/
            com.ibm.productivity.tools.base.system.linux_3.0.1.20080123-2030
```

Chapter 2. Customizing the Lotus Symphony user interface

The followed examples are all need you build a plug-in project firstly, and then edit the plugin.xml file directly by the code provided below. If you don't familiar with how to build a plug-in project, please go to the Part 5 Example plug-in to see the details.

2.1 Contributing to menu

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

For convenient, we recommend menus of third parties contributed to somewhere under the menu "Add-ins". If other third party has defined the menu "Add-in", you can use it; otherwise, you should define such a menu and use it.

Adding to the menu

To add to the menu and toolbar, do 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="com.ibm.rcp.ui.actionsmenu">
                     <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

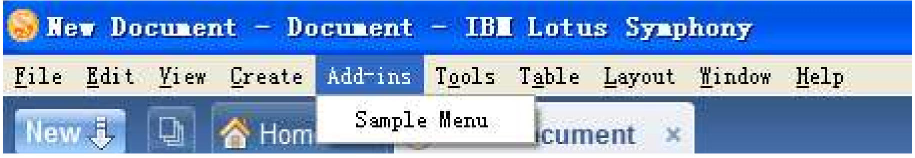
The extension point is provided by Eclipse Rich Client Platform.

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 Contributing to toolbar

Lotus Symphony allows the contribution to main toolbar. It is suggested to contribute you own toolbar group. The contribution is achieved through Expeditor extension point: com.ibm.rcp.ui.controlSets.

To contribute items to Symphony main toolbar, you can perform the following steps:

1) Make sure your plugin have the following dependencies:
   - com.ibm.productivity.tools.core
   - com.ibm.productivity.tools.ui.toolbar
   - com.ibm.rcp.jfaceex
2) Extend 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>
```

2) Provide 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.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;
   }
}
```

3) (Optional) Define association in plugin.xml if you want to associate your toolbar with Symphony views.

com.ibm.productivity.tools.ui.toolbar.controlSetSODCAssociations is an extension point defined to associate control sets with Symphony views so that those associated control sets only show up when a 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 control set has to be "false".

To associate this control set with symphony view, define below 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 controlset is visible by default.

Package

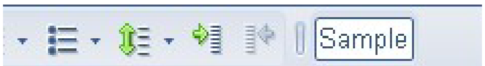
com.ibm.rcp.platform.controlSets are defined in Lotus Expeditor platform. com.ibm.productivity.tools.ui.toolbar.controlSetSymphonyAssociations are defined in com.ibm.productivity.tools.ui.toolbar plugin.

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

Example

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



2.3 Contributing to launcher button

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

Launching items

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="sym.guide.test.LauncherSet">

         <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 - Google"
          url="http://www.google.com/"/>

     </LauncherSet>
  </extension>
```

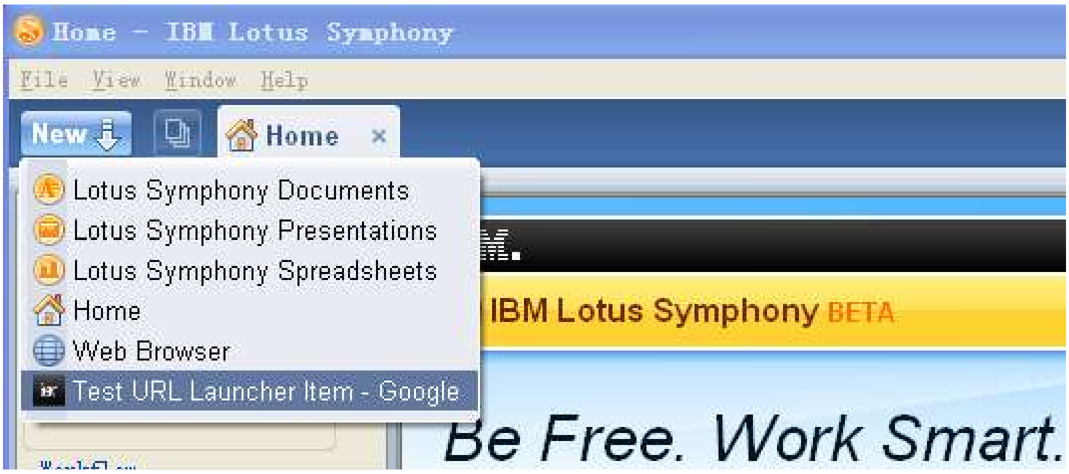
Package

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

Example


Home - IBM Lotus Symphony
File View Window Help
New
Home
Lotus Symphony Documents
Lotus Symphony Presentations
Lotus Symphony Spreadsheets
Home
Web Browser
Test URL Launcher Item - Google
IBM Lotus Symphony BETA
Be Free. Work Smart.

2.4 Contributing to side shelf

A side bar 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 side bar in the user interface which is based on the Lotus Expeditor extension point: `com.ibm.rcp.ui.shelfViews`.

Adding a new view in the shelf view

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 contribute items to Symphony shelf view, you can perform the following steps:

1) Make sure your plugin 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 the view a contribution 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

   }
}
```

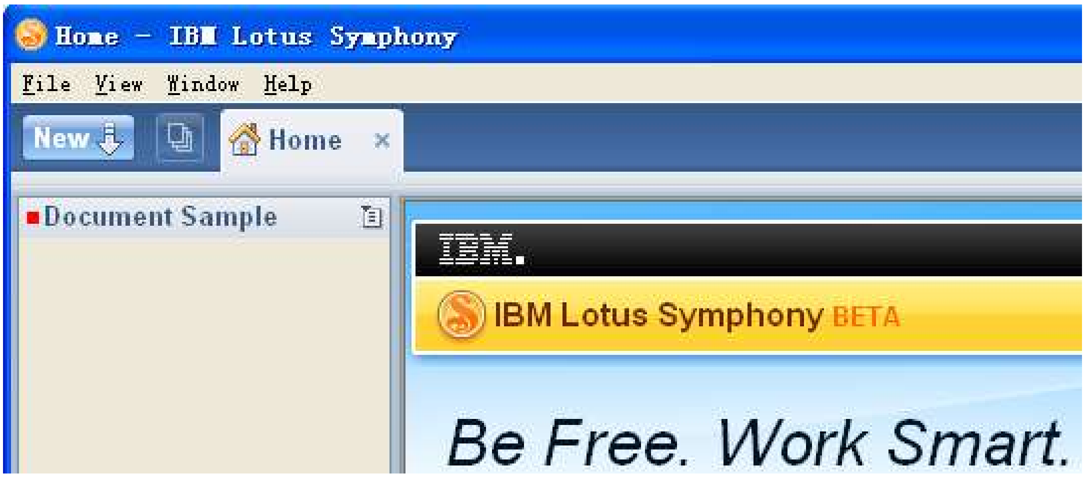
Package

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

Example



2.5 Auto recognizer

Auto recognizer is a technique to allow users to take actions based on the text 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.

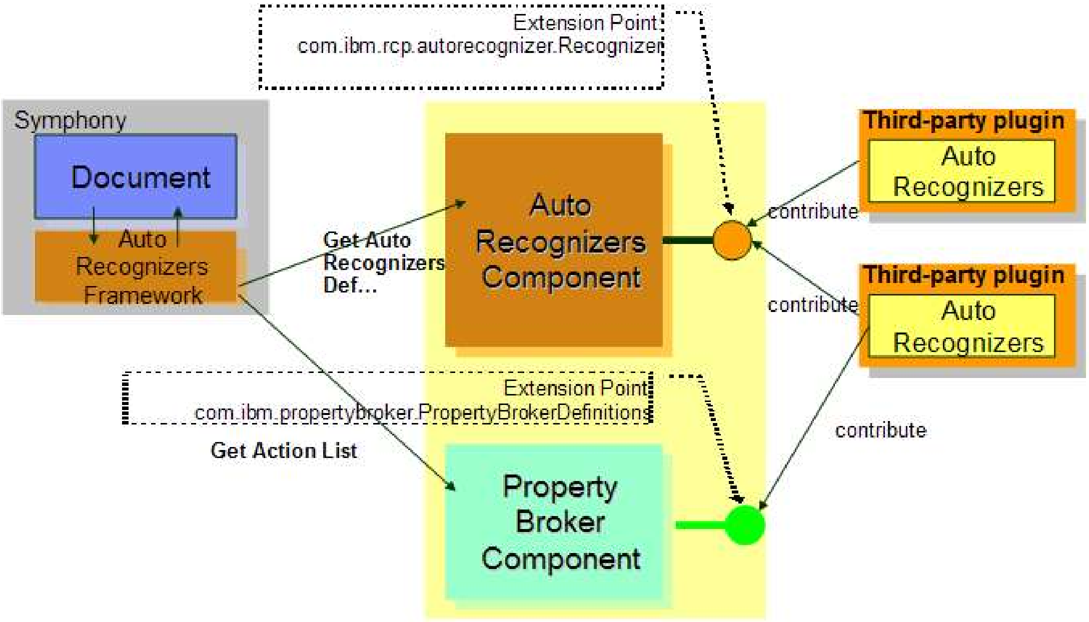
Note: Auto recognizer is only available in the Writer application and only single word patterns are supported.

Using the auto recognizer

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 do the following things:

1. Adding dependencies the com.ibm.rcp.autorecognizer and the 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 picture is the overall architect of auto recognizer.


Extension Point
com.ibm.rcp.autorecognizer.Recognizer
Symphony
Document
Auto Recognizers Framework
Get Auto Recognizers Def...
Auto Recognizers Component
Third-party plugin
Auto Recognizers
contribute
contribute
Third-party plugin
Auto Recognizers
Extension Point
com.ibm.propertybroker.PropertyBrokerDefinitions
Get Action List
contribute
Property Broker Component

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 will be detected by the underlying auto recognizer framework in the document:

```
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 `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:

```
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

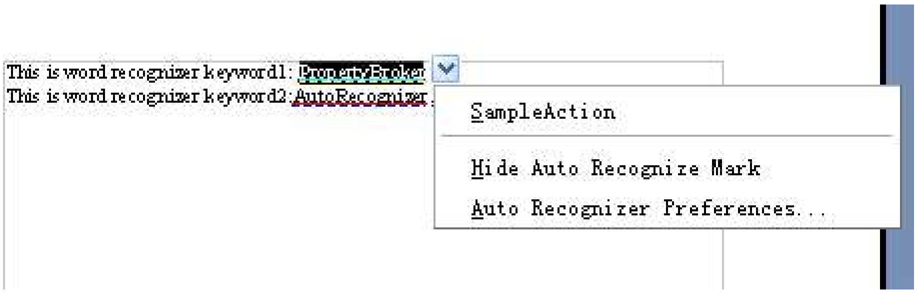
com.ibm.rcp.autorecognizer.

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

Example

In following example, a plug-in defines that “PropertyBroker” and “AutoRecognizer” are two keywords, and add a special action (in this example, the SampleAction) 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.


This is word recognizer keyword1: PropertyBroker

This is word recognizer keyword2: AutoRecognizer

2.6 Contributing to 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`.

Adding an item to the status bar

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)`.

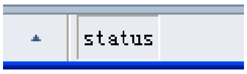
Package

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

Example



2.7 Preferences page

After a plug-in has added extensions to the Lotus Symphony user interface, let users control some of the behavior 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.

Adding a preferences page

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.Doc
umentEditors"/>
 </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");
            }
}
```

Package

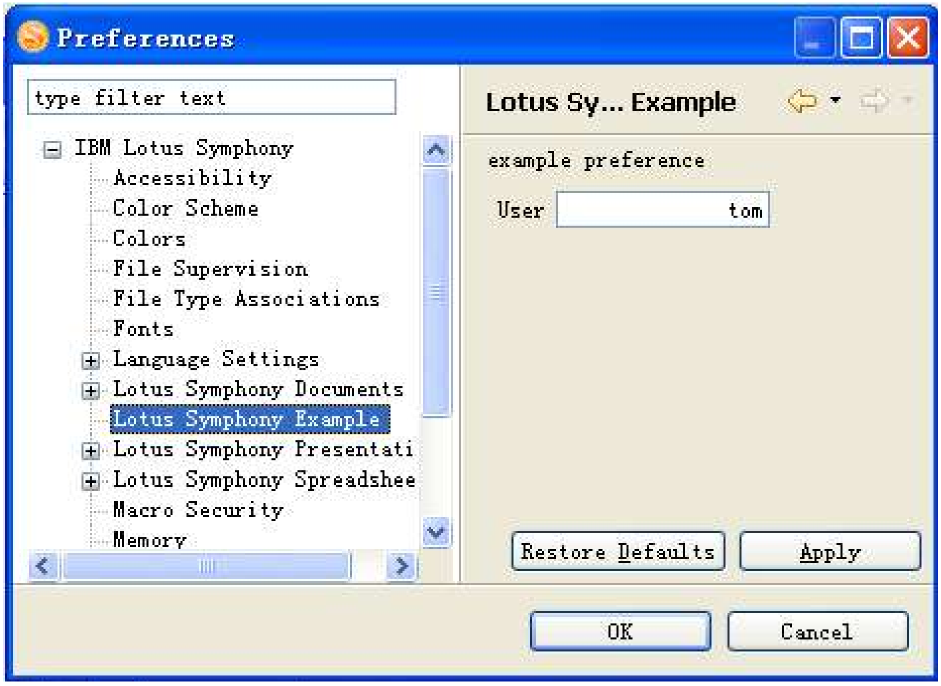
The extension point is provided by Eclipse Rich Client Platform.

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

Example



example preference

User tom

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 changed in the view.

Selection 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 or ODF 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.getClass(RichDocumentContentSelection.class );
String text = textSel.getText();
```

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

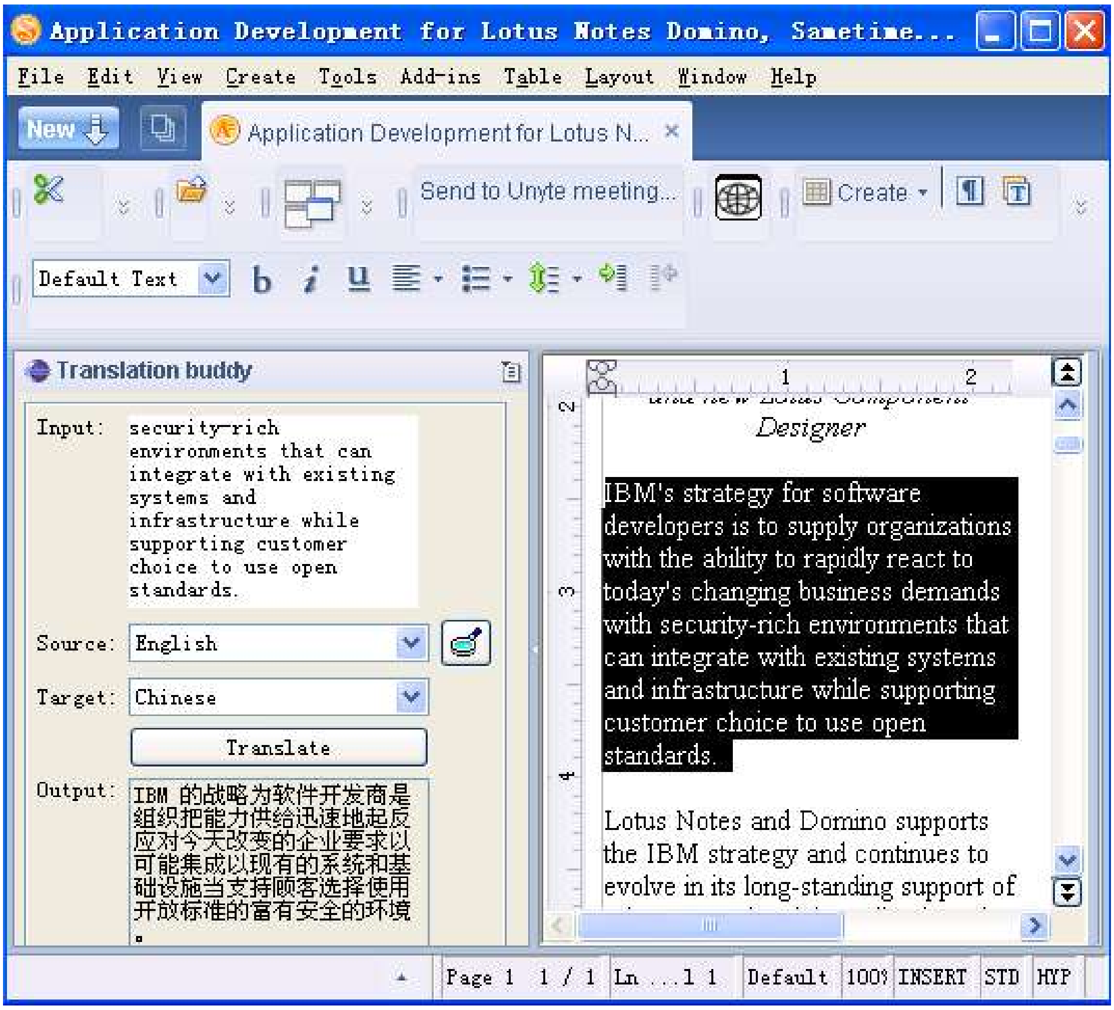
com.ibm.productivity.tools.ui.views

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.


Application Development for Lotus Notes Domino, Sametime...
File Edit View Create Tools Add-ins Table Layout Window Help
New
Application Development for Lotus N...
Send to Unyte meeting...
Create
Default Text
Translation buddy
Translate
Page 1 1 / 1 Ln ...1 1 Default 100% INSERT STD HYP

Input: security-rich environments that can integrate with existing systems and infrastructure while supporting customer choice to use open standards.

Source: English

Target: Chinese

Output: IBM 的战略为软件开发商是組织把能力供给迅速地起反应对今天改变的企业要求以可能集成以现有的系统和基础设施当支持顾客选择使用开放标准的富有安全的环境。

Designer

IBM's strategy for software developers is to supply organizations with the ability to rapidly react to today's changing business demands with security-rich environments that can integrate with existing systems and infrastructure while supporting customer choice to use open standards.

Lotus Notes and Domino supports the IBM strategy and continues to evolve in its long-standing support of

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 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 side bar 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

Perform 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 RichDocumentView instance. If a listener is registered when `RichDocumentView` is created, closed or a document is loaded, the listener is notified. Currently the following events 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.getType() );
            }


}
```

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

Package

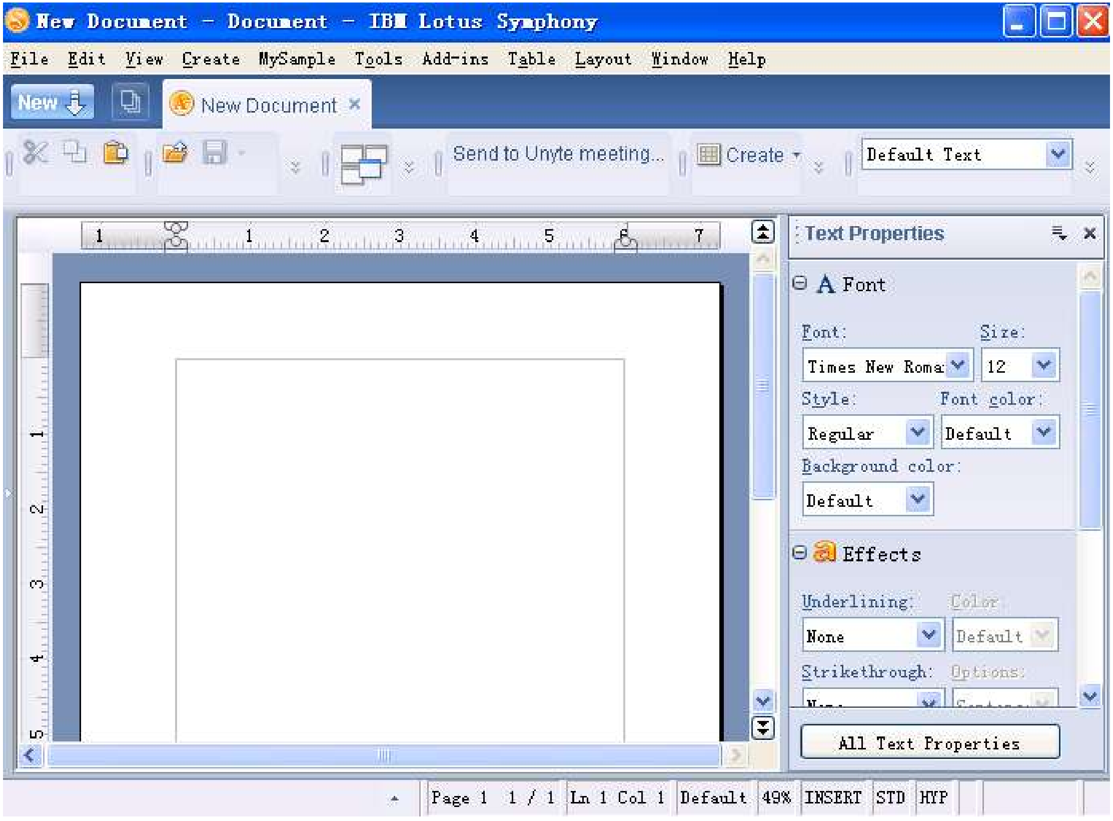
com.ibm.productivity.tools.ui.views.

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.



New Document - Document - IBM Lotus Symphony
File Edit View Create MySample Tools Add-ins Table Layout Window Help
New
New Document
Send to Unyte meeting...
Create
Default Text
Text Properties
Font
Effects
All Text Properties
Page 1 1 / 1 Ln 1 Col 1 Default 49% INSERT STD HYP

Font: Times New Roma
Size: 12
Style: Regular
Font color: Default
Background color: Default
Underlining: None
Color: Default
Strikethrough:
Options:

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

Beside 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 which 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 Productivity Tools toolkit:

```
  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, CloseOperation provided in the Javadoc API. The usage is similar to LoadOperation 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 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 example 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 example code to insert a table to 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 example code to set content in 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 document 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 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, you ought to run your code in installed Lotus Symphony product environment, or distribute your plug-ins to customer 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 from IBM Lotus Symphony toolkit.

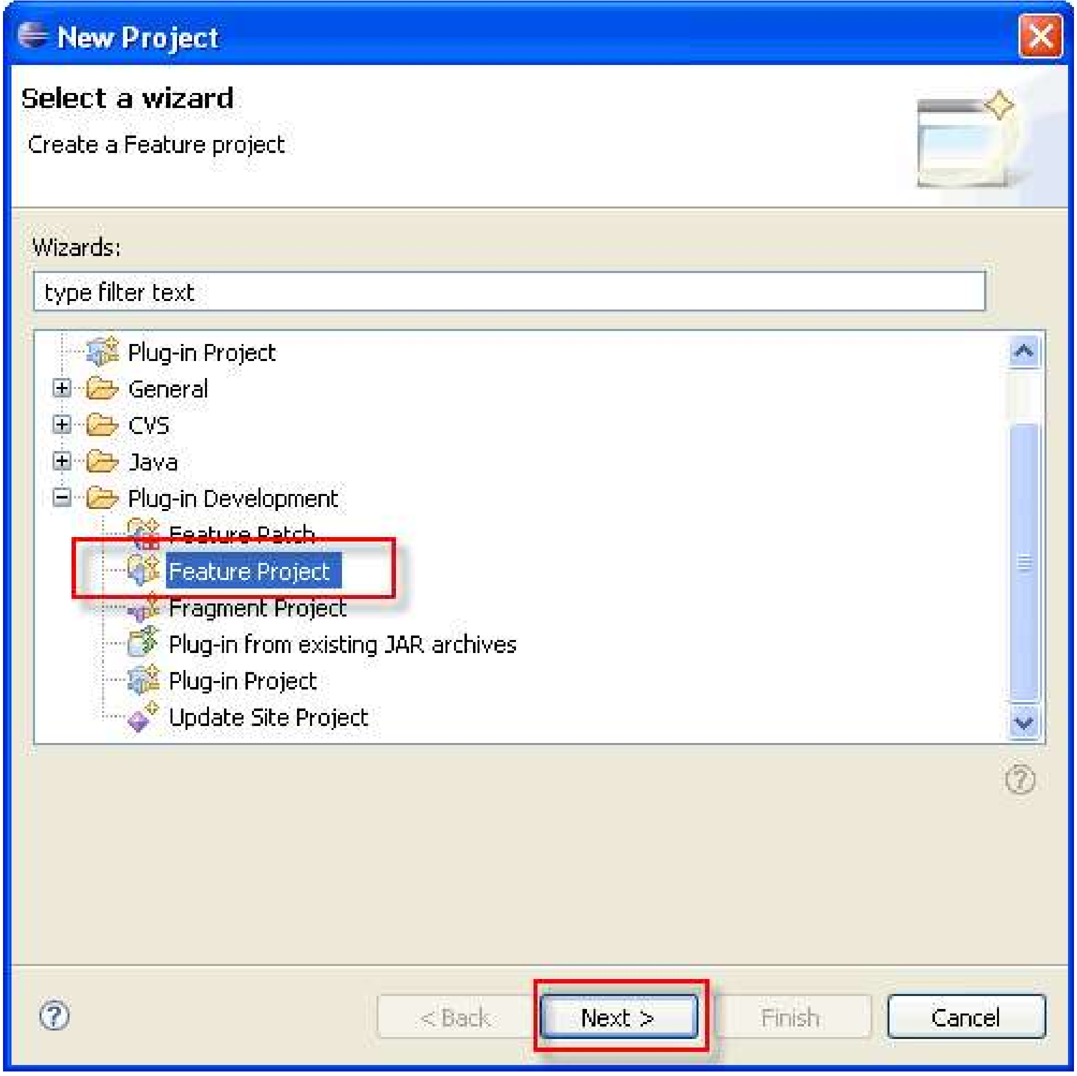
5.1 Creating a feature

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.

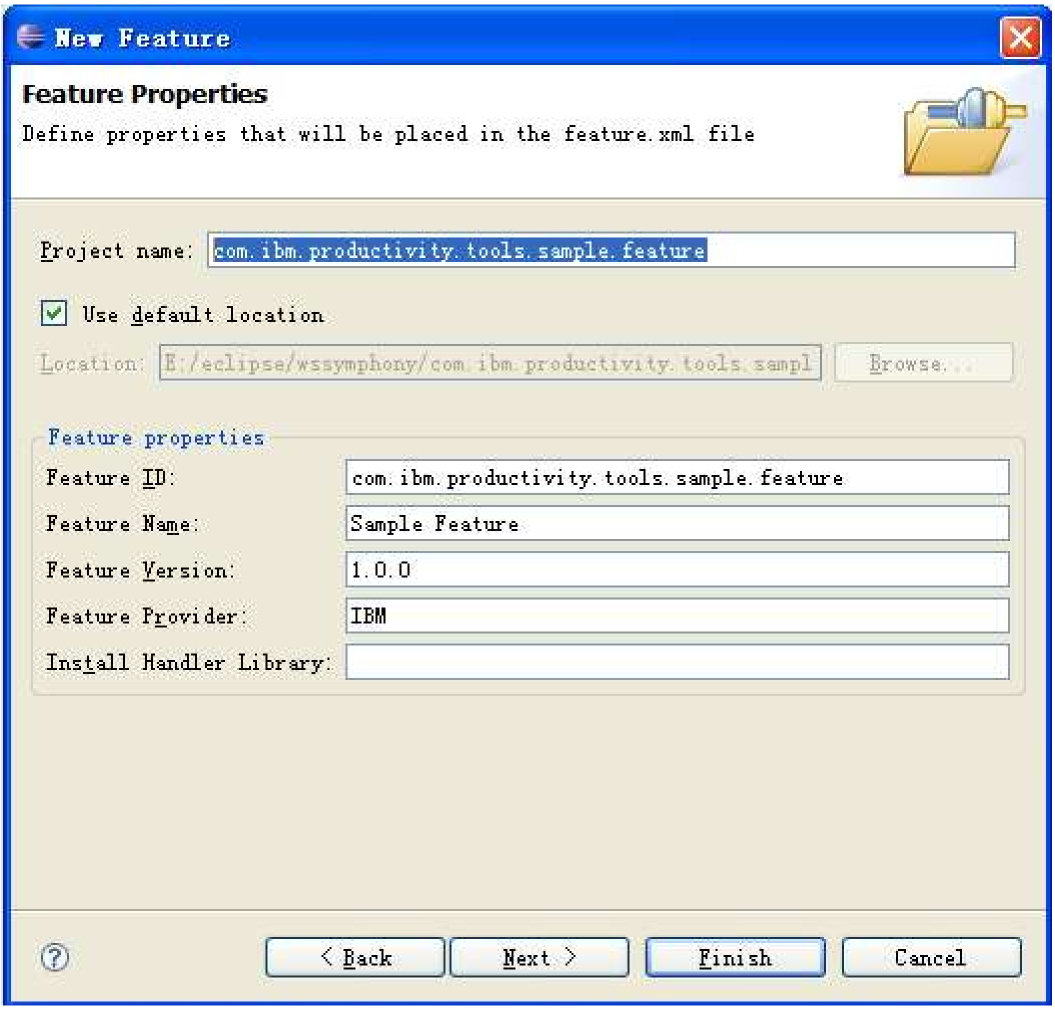
Now that your plug-in is ready to be deployed, it will need to be packaged in a manner that will be recognized by Eclipse Update Manager. Eclipse Update Manager is an Eclipse tool that manages versions and deployment of plug-ins and fragments.

Next, you need to create a Feature for your plug-in(s):

1. Make sure your plug-in is open in the Workspace you created.
2. Select **File** → **New** → **Project**. This will launch the New Project Wizard.
3. Select **Feature Project** wizard, and then click **Next**, as shown in the following figure:



4. On the New Properties page for the New Feature wizard, complete the Properties as follows:



Feature Properties

Define properties that will be placed in the feature.xml file

Project name: com.ibm.productivity.tools.sample.feature

☑ Use default location

Location: E:/eclipse/wssymphony/com.ibm.productivity.tools.sampl Browse...

Feature properties

Feature ID: com.ibm.productivity.tools.sample.feature

Feature Name: Sample Feature

Feature Version: 1.0.0

Feature Provider: IBM

Install Handler Library:

5. Click **Next**.

6. On the Referenced Pug-ins and Fragments page, select the plug-in 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.

7. There are many options in this view. Change the following fields if necessary:
  a. In the Branding Plug-in field, click the **Browse...** field.
  b. Select the plug-in you wish to deploy and click **OK**, as shown in.
  c. In the Update Site URL field, enter the Eclipse Update Site URL.
  d. In the Update Site Name field, enter the site name.

This information is used to specify the site that will be 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.

e. In the Supported Environments section, enter OS, platform, and language specifications, if these are required by your plug-in. For our example, this is not necessary.

8. Click the **Information** tab.

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.

9. 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.

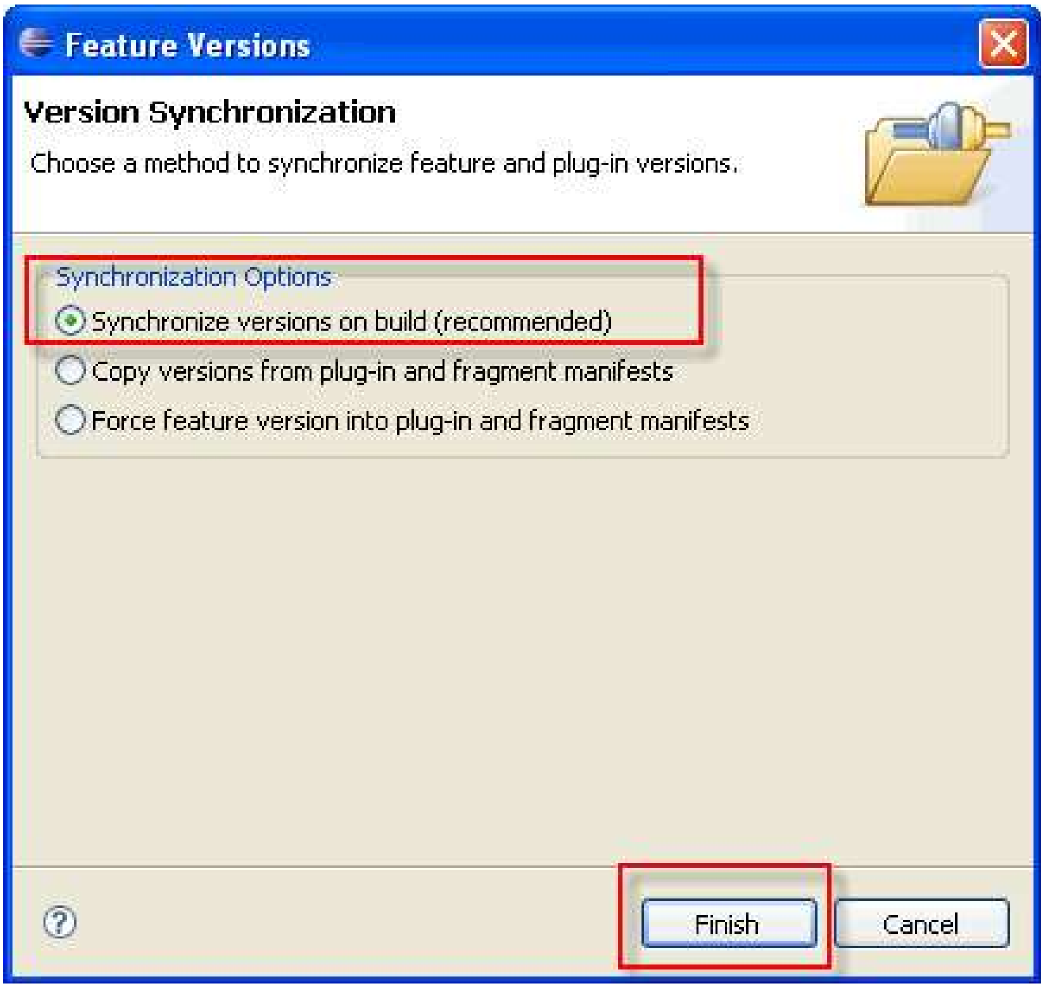
10. In the Optional URL field, enter a URL and name for any other relevant Update sites you have.

11. Click the **Plug-in** tab.

12. 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 you wish to include, and then click **OK**.

13. Click the **Version** button.

14. Select **Synchronize Versions on Build (recommended)**, as shown in the following figure, and then click **Finish**. This will synchronize your feature version and plug-in version:



Feature Versions

Version Synchronization

Choose a method to synchronize feature and plug-in versions.

Synchronization Options

● Synchronize versions on build (recommended)

○ Copy versions from plug-in and fragment manifests

○ Force feature version into plug-in and fragment manifests

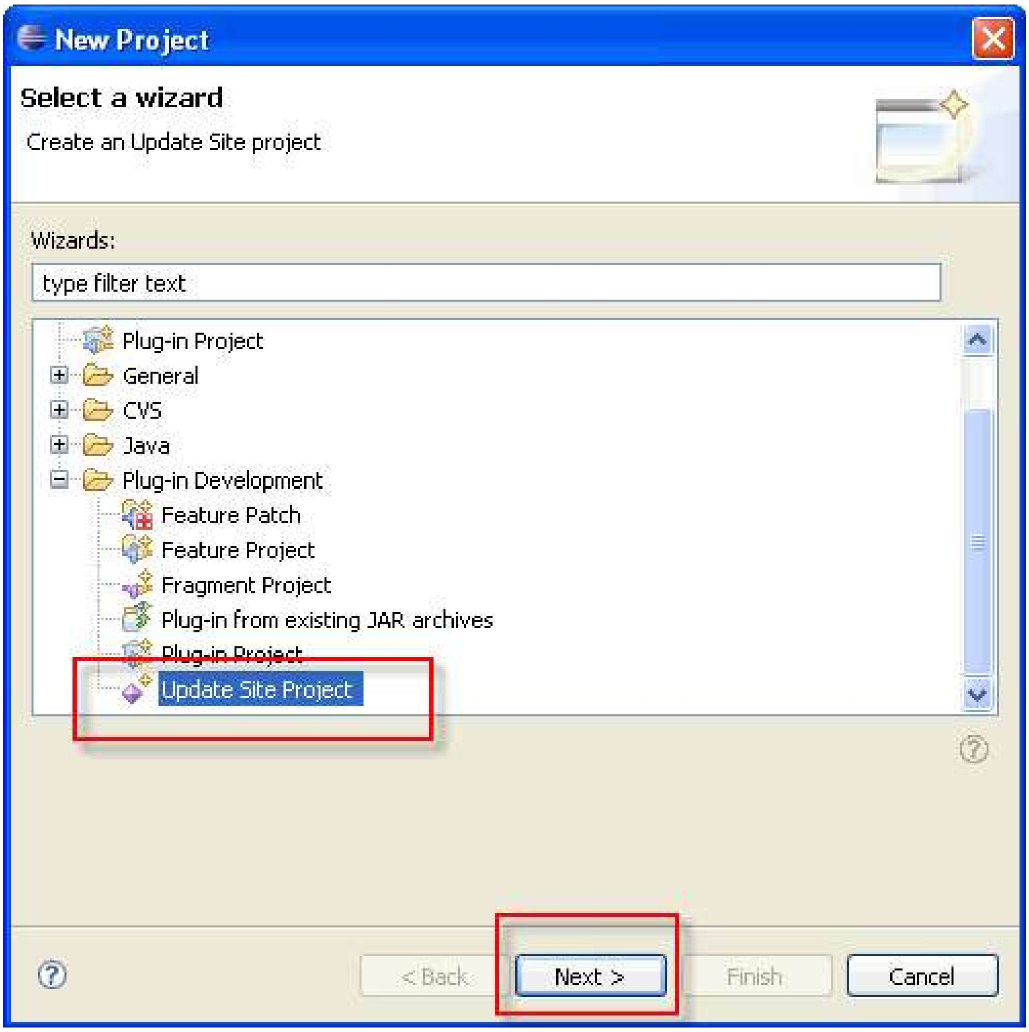
15. Your feature and plug-in are now ready to deploy.

5.2 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, please see the Getting Started > Update Sites section of the PDE Guide.

The steps of creating an update site are:

1. Open Eclipse. Be sure to open the workspace where you created your plug-in and feature.
2. Select **File** → **New** → **Project**.
3. Select **Update Site Project**, as shown in the following figure, and then click Next to launch the wizard.



4. 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 this is an update site project.
   b. Select **Use the default location**.
   c. Check the **Generate a Web page listing all available resources** within the site if you

want it deployed through web site.

5. Click **Finish**. The wizard creates your update site within your Eclipse workspace.

6. To add your feature(s), double-click the **site.xml** file located in the Package Explorer frame.

7. This will open your site manifest editor in the editor frame (center frame).To add your new feature, click **Add Feature**. If you are adding more than one feature/plug-in or plan to in the future, you may choose to organize them by category.

8. Select the feature you are including in this update site. You can select more than one by holding down the Ctrl key. When finished selecting, click **OK**.

9. Click the **Build All** button. This 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 will now build your update site locally.

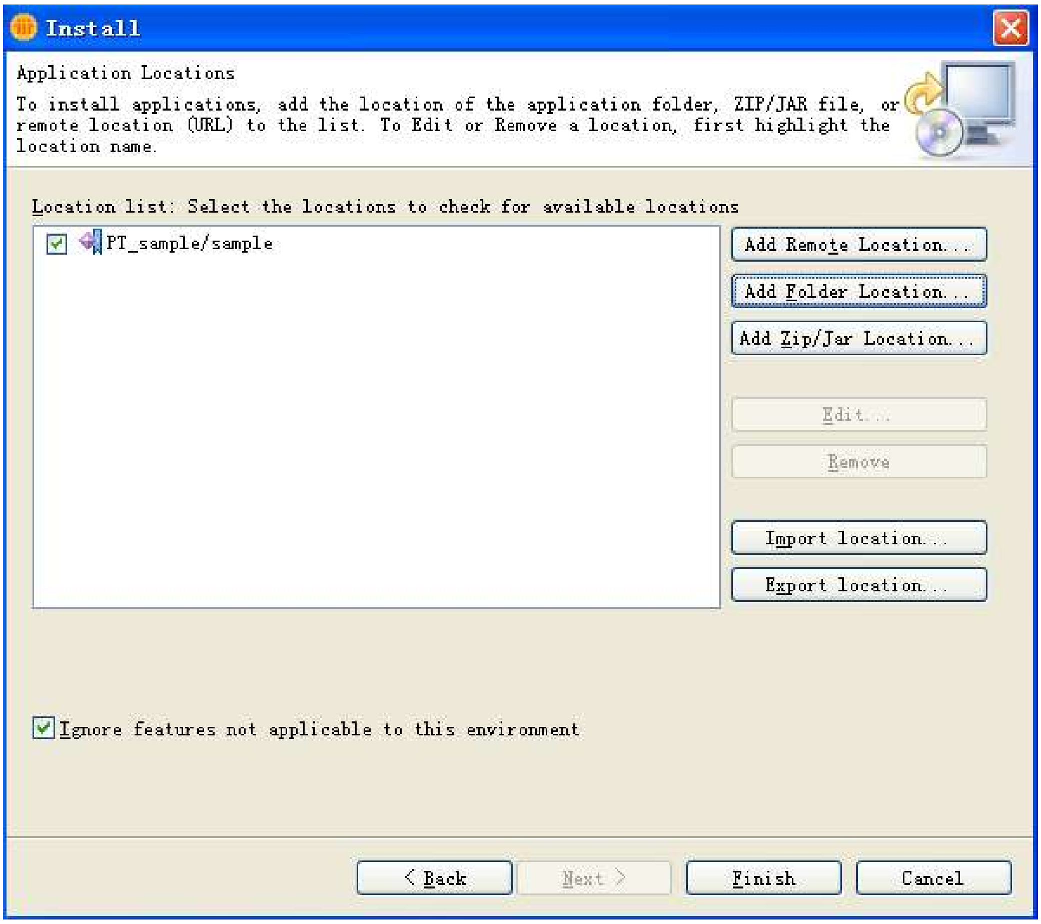
10. Export this update site project to file system, e.g. F:\deploy\PT_sample.

5.3 Deploying with an update site

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

Launch Lotus Symphony and choose the **File > Applications > Install** menu item.

- Select "**Search for new features to install**" and next.
- Click "Add Folder Location" and select the update site project which is just created in local file system, then click Finish button.



Application Locations

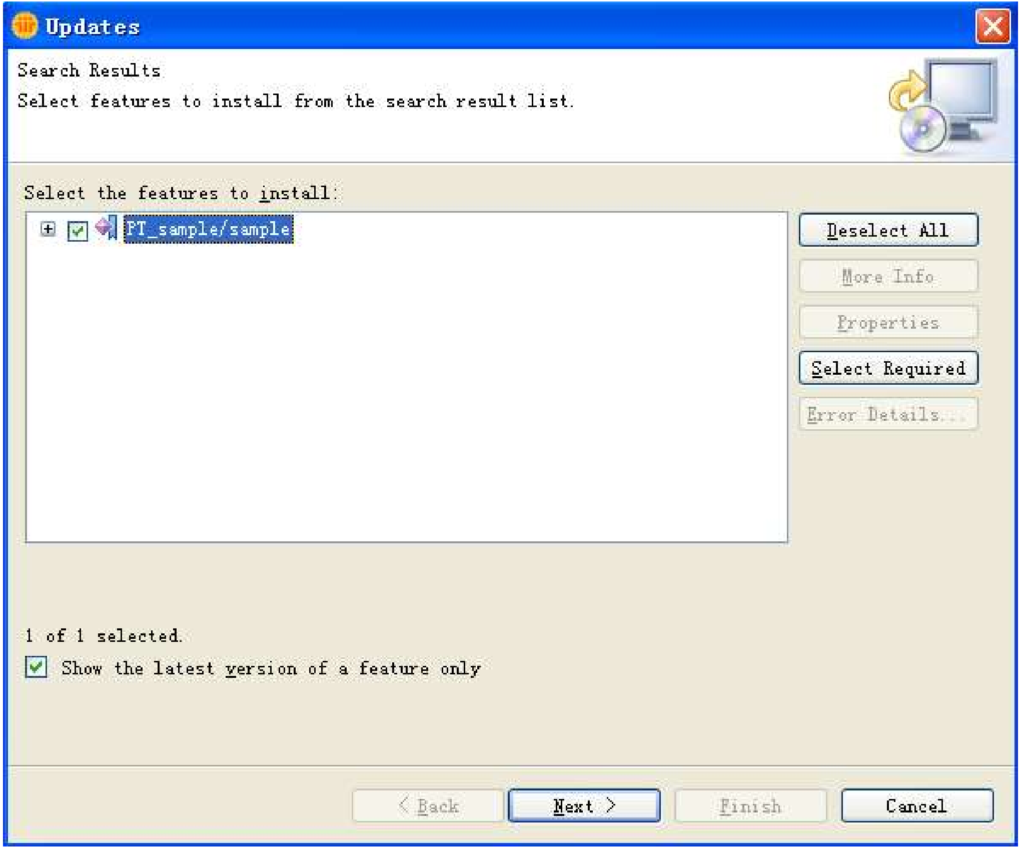
To install applications, add the location of the application folder, ZIP/JAR file, or remote location (URL) to the list. To Edit or Remove a location, first highlight the location name.

Location list: Select the locations to check for available locations

☑ PT_sample/sample

☑ Ignore features not applicable to this environment

- In "**Updates**" dialog, select the update site, and then click **Next**.



Search Results

Select features to install from the search result list.

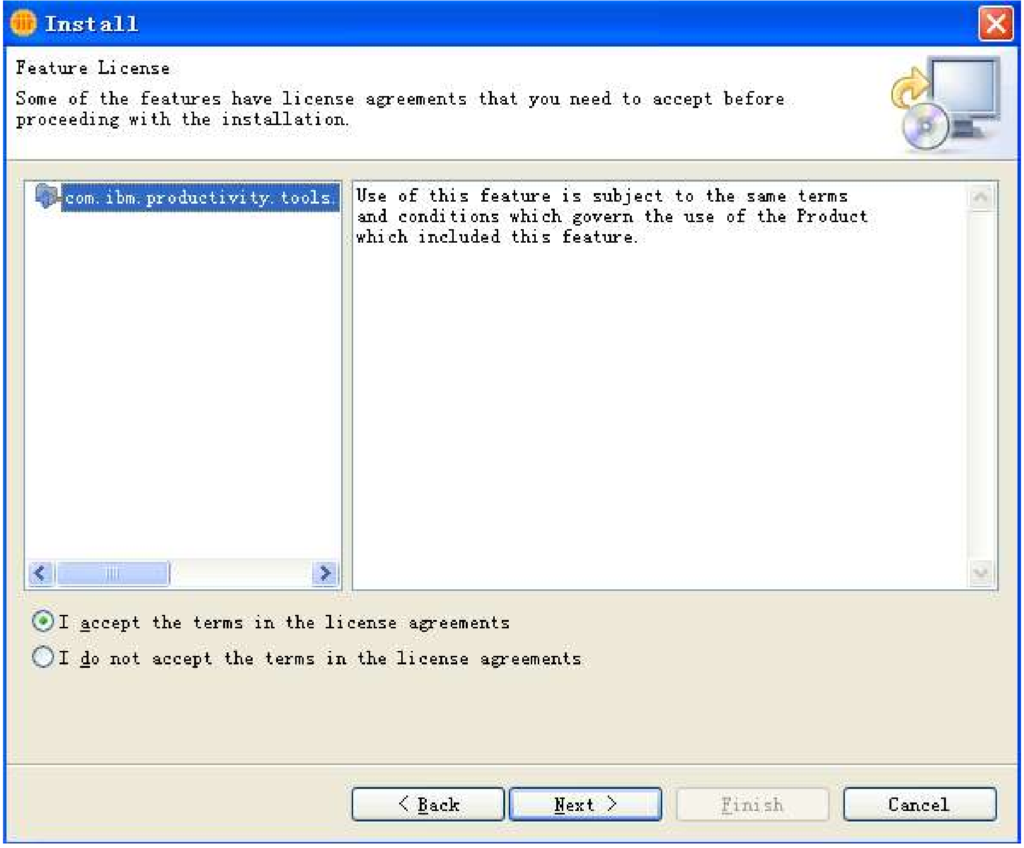
Select the features to install:

⊞ ☑ PT_sample/sample

1 of 1 selected.

☑ Show the latest version of a feature only

- Accept the terms in the license agreements in **Install** dialog, click **Next**, and **Finish** in next dialog.



Feature License

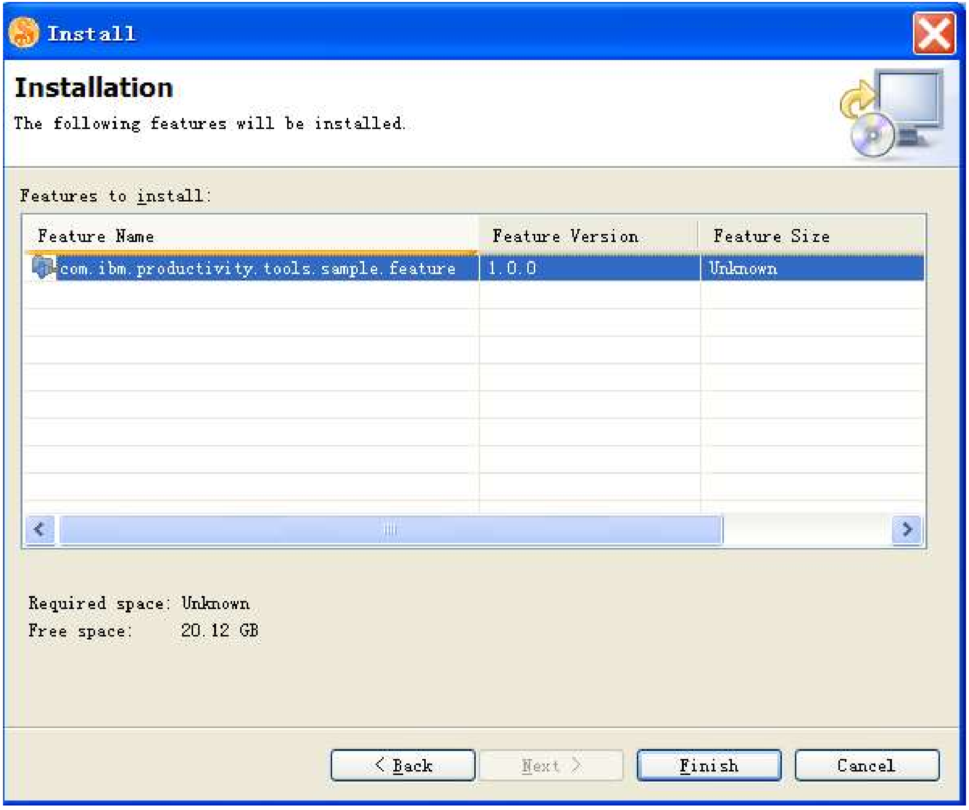
Some of the features have license agreements that you need to accept before proceeding with the installation.

Use of this feature is subject to the same terms and conditions which govern the use of the Product which included this feature.

◉ I accept the terms in the license agreements

○ I do not accept the terms in the license agreements

- Click "Finish" to install the imported feature.



Installation

The following features will be installed.

Features to install:

Feature Name	Feature Version	Feature Size
com.ibm.productivity.tools.sample.feature	1.0.0	Unknown

Required space: Unknown
Free space: 20.12 GB

After the mentioned steps, there is a new menu appearing in menu bar which shows you have successful installed your applications.

Part 4. Expeditor and UNO programming

Chapter 1. Developing Lotus Expeditor applications

There are several application models that are defined in Lotus Expeditor, although some of them are not available in Lotus Symphony, for example, the Web application model and portlet application model.

When opening a document from the user interface or API, the document is opened using the following steps:

1) A new perspective is created as a new tab window.
2) A Lotus Symphony view is created in the perspective as an Eclipse `ViewPart.`
3) A new SWT control is created in the view.
4) The SWT control loads the document.

This guide focuses on how to extend Lotus Symphony with Expeditor and Eclipse extension points. After you understand the rich client application model in Lotus Expeditor, you can build your own 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.

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 example 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/export function

The import/export function is 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 then specific document types which 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 single 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 will 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, 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);
}
```

Interface for exporting presentation documents and drawing objects

Presentation documents and drawing objects 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.

Following is example code is from the HTML filter inside Lotus Symphony:

```
{
   try
   {
       Reference < XMultiServiceFactory > xMSF(comphelper.getProcessServiceFactory
           () );
       if( !xMSF.is() )
           return false;

       Reference< XExporter > xGraphicExporter( xMSF->createInstance( OUString
           ( RTL_CONSTASCII_USTRINGPARAM
           ("com.sun.star.drawing.GraphicExportFilter") ) ), UNO_QUERY );
       Reference< XFilter > xFilter( xGraphicExporter, UNO_QUERY );

       DBG_ASSERT( xFilter.is(), "no com.sun.star.drawing.GraphicExportFilter?" );
       if( !xFilter.is() )
           return false;

       Sequence< PropertyValue > aFilterData(((m_eFormat==FORMAT_JPG)&&(m_nCompression
           != -1))? 3 : 2);
       aFilterData[0].Name = OUString( RTL_CONSTASCII_USTRINGPARAM("PixelWidth") );
       aFilterData[0].Value <<= (sal_Int32)m_nWidthPixel;
       aFilterData[1].Name = OUString( RTL_CONSTASCII_USTRINGPARAM("PixelHeight") );
       aFilterData[1].Value <<= (sal_Int32)m_nHeightPixel;
```

```
        if((m_eFormat==FORMAT_JPG)&&(m_nCompression != -1))
        {
            aFilterData[2].Name = OUString( RTL_CONSTASCII_USTRINGPARAM("Quality") );
            aFilterData[2].Value <<= (sal_Int32)m_nCompression;
        }

        Sequence< PropertyValue > aDescriptor( 3 );
        aDescriptor[0].Name = OUString( RTL_CONSTASCII_USTRINGPARAM("URL") );
        aDescriptor[1].Name = OUString( RTL_CONSTASCII_USTRINGPARAM("FilterName") );
        aDescriptor[1].Value <<= OUString( RTL_CONSTASCII_USTRINGPARAM
            (m_eFormat==FORMAT_GIF ? "GIF" : "JPG") );
        aDescriptor[2].Name = OUString( RTL_CONSTASCII_USTRINGPARAM("FilterData") );
        aDescriptor[2].Value <<= aFilterData;

        for (USHORT nSdPage = 0; nSdPage < m_nSdPageCount; nSdPage++)
        {
            SdPage* pPage = pDoc->GetSdPage( nSdPage, PK_STANDARD );

            OUString aFull(m_aExportPath);
            aFull += *m_pImageFiles[nSdPage];

            aDescriptor[0].Value <<= aFull;

            Reference< XComponent > xPage( pPage->getUnoPage(), UNO_QUERY );
            xGraphicExporter->setSourceDocument( xPage );
            xFilter->filter( aDescriptor );

            if (mpProgress)
                mpProgress->SetState(++m_nPagesWritten);
        }
    }
    catch( Exception& )
    {
        return false;
    }

    return true;
```

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 the 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 same parts, we provide 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. So, all providers are surrounded with the character strings.

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 DrawPage.

The DrawPage is floating over the text and it is responsible for drawing contents. Drawing contents can affect the layout of the text around, 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 intuitionist ideas, refer to the *Illustration 7.1 Text Document Model* of OpenOffice 1.1 Development Guide.

Working with text documents

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 such function. Developers will meet 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 character, 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 content 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 real 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 will be 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 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 are 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 inserting 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, returns a com.sun.star.text.TextFieldMasters container holding the text field masters of the document. This container provides a com.sun.star.container.XNameAccess interface. All field masters are named by the service name followed by the name of the field master

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 with 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. TShape objects can float in front or behind text, and be anchored to paragraphs or characters in the text or page and so on. 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 brushfire formatting. TIt is a good way to unify the appearance of a document, and customize the formatting of a document by altering a style, instead of 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 documenting 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 numbering and outline numbering

It 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.

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 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

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

TextViewCursor

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

2.4 Spreadsheets

Spreadsheet documents derive all UNO API from OOo1.1.0(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. In development, Several APIs have been added or changed.

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

Spreadsheet document: Operations of spreadsheet documents are mainly in *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, and *com.sun.star.util.XProtectable* contains methods to protect and unprotect spreadsheet with a password, and text in of a cell. In cells and , as well as cell ranges, table rows, columns, and Single cell:

com.sun.star.sheet.SheetCell is used to present cell object, and *com.sun.star.table.CellProperteis is used to format cell(s).*

Cell Range: The service com.sun.star.sheet.SheetCellranges contains most of interface of 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.

Enhancement

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 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 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 is updated in user interface. All of the formulas or charts depending on this cell do not get refresh until SyncDocument is enabled.

/** 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.
*/

Example*:*

```
   SyncDocument(FALSE); //disable to update some of UI and document data
    for(i=0;i<100;1++)
        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 will be inserted to aDestPos with linkage kept.
When bLink == False, only the value will be inserted.

Do not use the following UNO APIs:

Interface and methods in service com.sun.star.sheet.DDELinks(Not fully tested).
Interface and methods in service com.sun.star.sheet.DatabaseImportDescriptor (Not fully tested).
Interface and methods in service com.sun.star.sheet.Scenarios(Not fully tested).
Methods in interface com.sun.star.sheet.XSheetAuditing(Not fully tested).
Methods to import data from a Web server(Not fully tested).

2.5 Charts

In IBM Lotus Symphony, charts are always embedded objects inside other Lotus Symphony documents. The chart document UNO API is almost the same as OOo1.1.0. Like the spreadsheet document, enhancements have been added in the Lotus Symphony core function, which will improve the API quality.
Chart can be added into spreadsheet document with source in a cell range. In presentation document or a writer document, a chart can be added as an OLE shape. Symphony Chart API provides capability of creating chart, accessing existing chart, and modifying chart property and elements. Ideally all the operations in chart which can be accomplished with UI can be done by API (refer to OO.1.1.0 Dev guide). For the reason of limitation of chart core function, the following operations are not supported by API:

- Chart with discrete data source in spreadsheet
- Trend Line

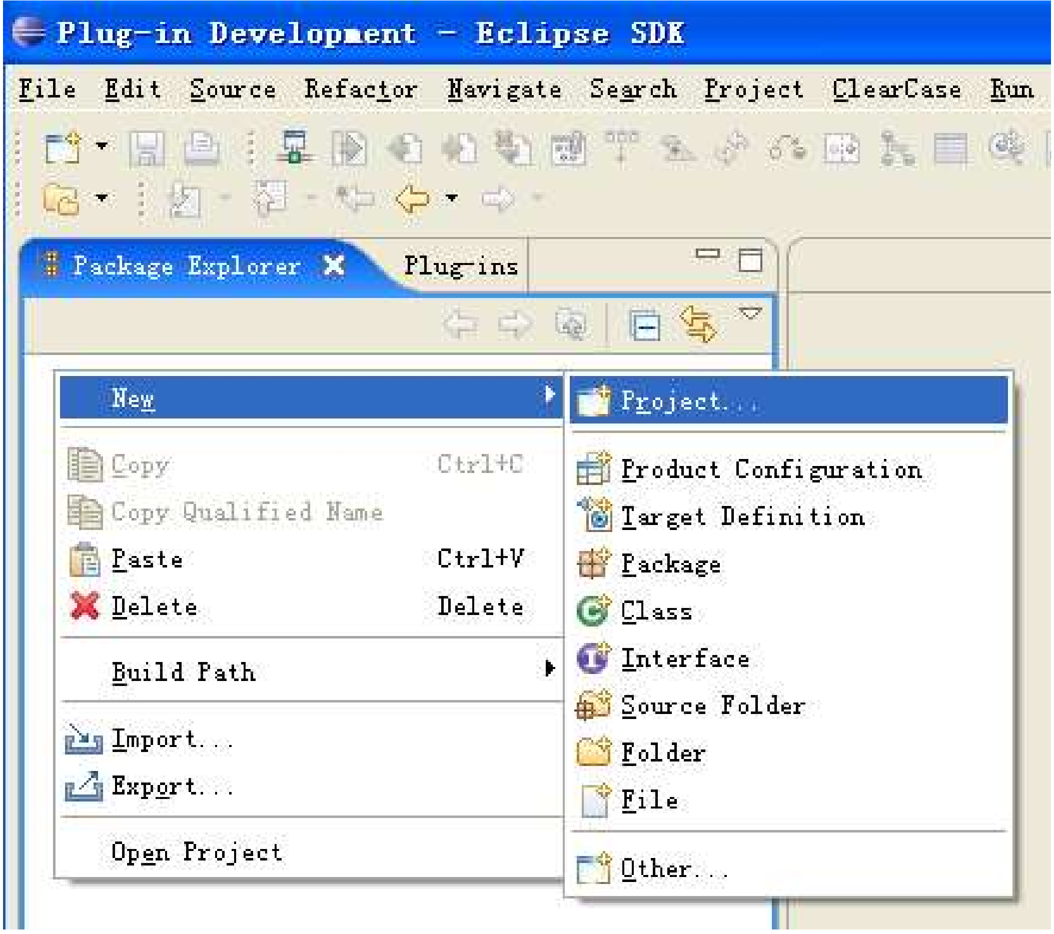
Part 5. Example plug-in

The example 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 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.views.

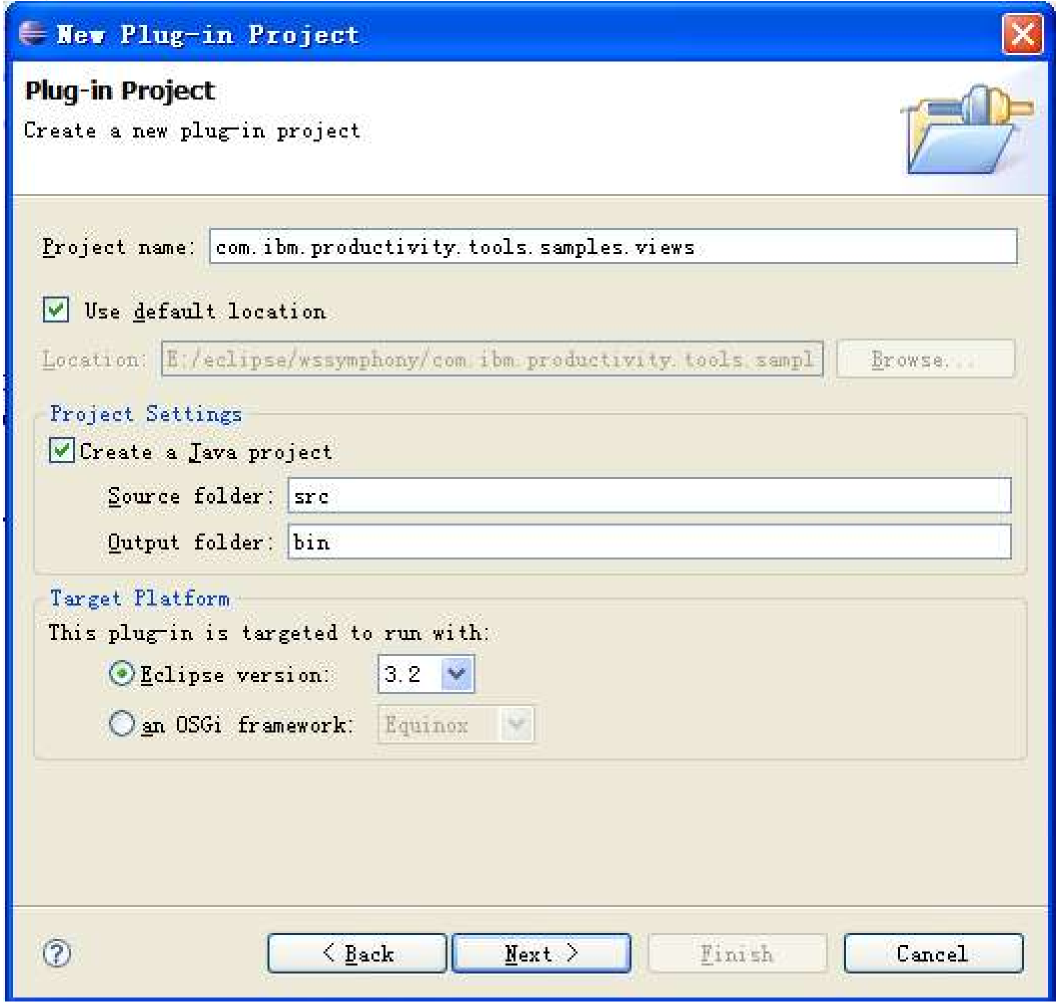
Creating a plugin

1. Set the development environment as in Chapter 1 of Part 3.
2. Select **File** > **New** > **Project…** menu item or click right mouse button and select **New** > **Project…** Context menu.



3. Select Plug-in Project in the Project Category.
4. Click **Next** button, type "com.ibm.productivity.tools.samples.views" as Project

name. Select **Next** button, and then click **Finish** button to finish creating the new project.



Plug-in Project

Create a new plug-in project

Project name: com.ibm.productivity.tools.samples.views

☑ Use default location

Location: E:/eclipse/wssymphony/com.ibm.productivity.tools.sampl

Project Settings

☑ Create a Java project

Source folder: src

Output folder: bin

Target Platform

This plug-in is targeted to run with:

● Eclipse version: 3.2

○ an OSGi framework: Equinox

5. Add dependencies: Select **Dependencies** tab, and click the **Add…**button to add required plug-ins. There are 2 plug-ins that be needed to add:

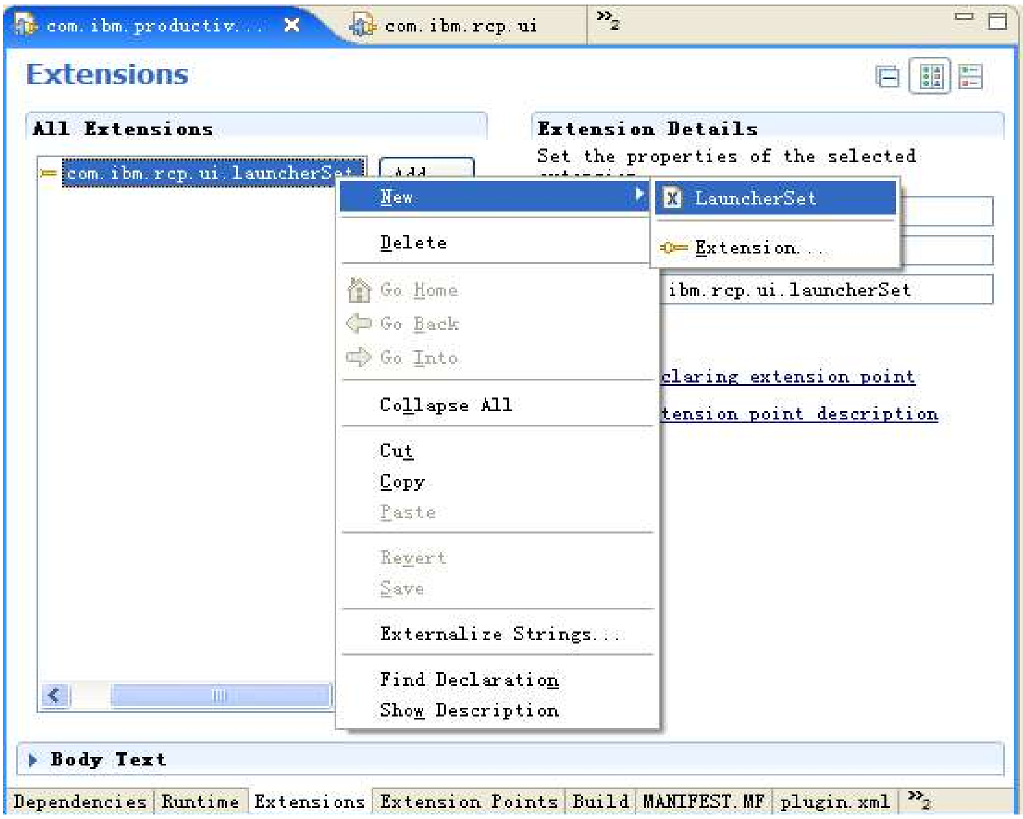
- com.ibm.rcp.ui
- com.ibm.productivity.tools.ui.views

Creating a new button

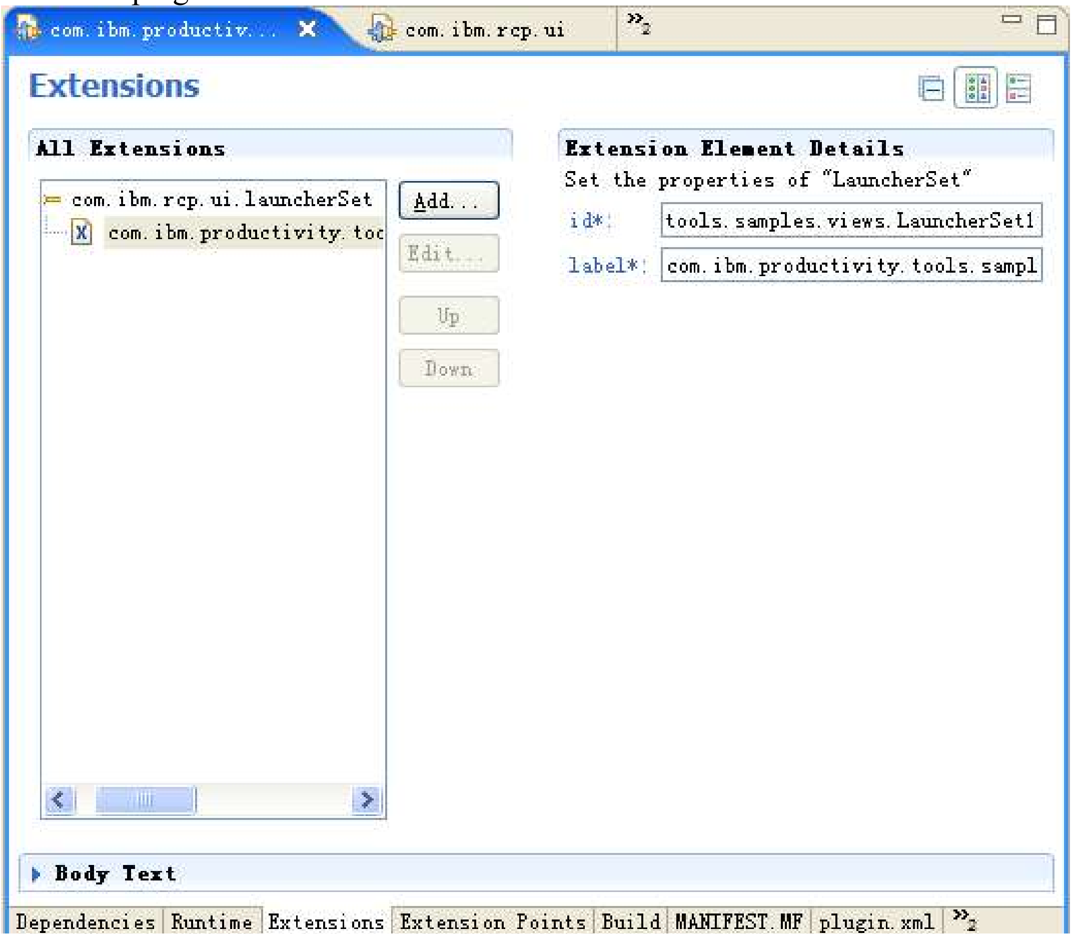
1. Select Extensions tab, click **Add…** button

- In the new extensions dialog, select "com.ibm.rcp.ui.launcherSet", then click **Finish** button.

2. In the Extensions page, select the added newly extension, click right mouse button on it and select **New > LauncherSet** context menu item.



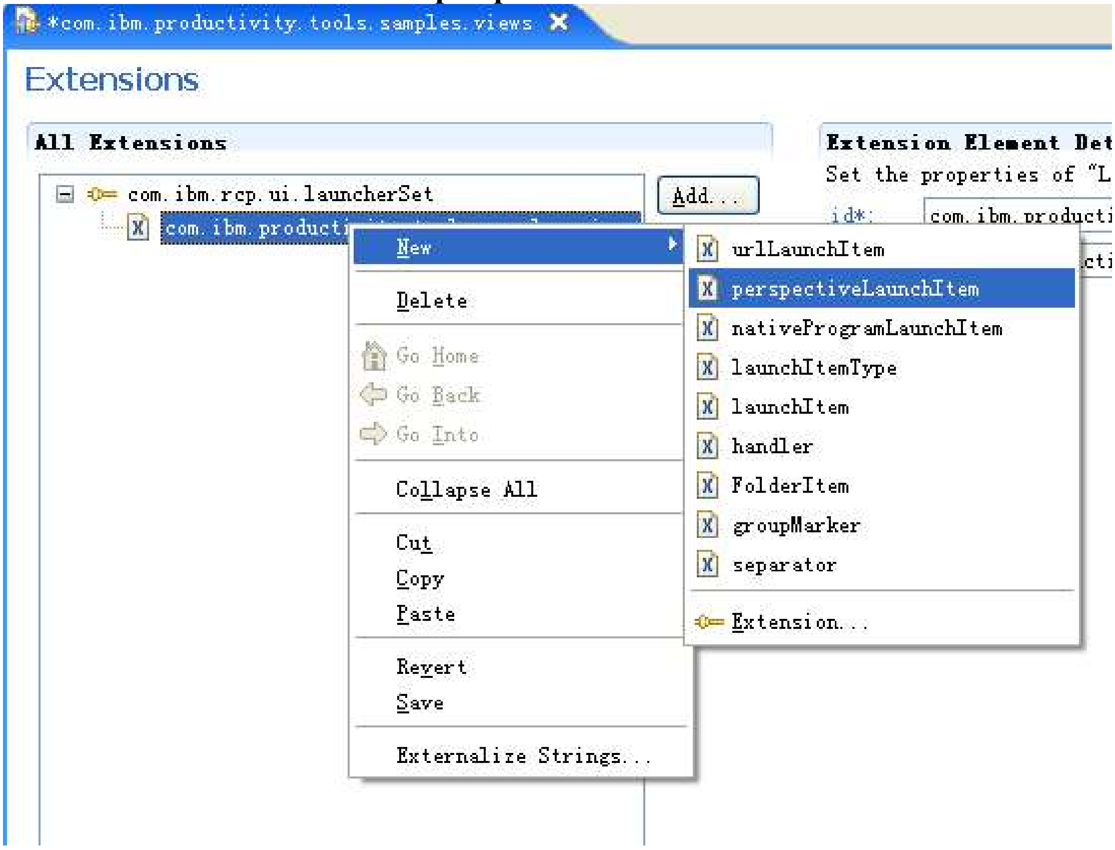
- Leave the id and label properties of the newly added LauncherSet unchanged, save the plugin.xml.



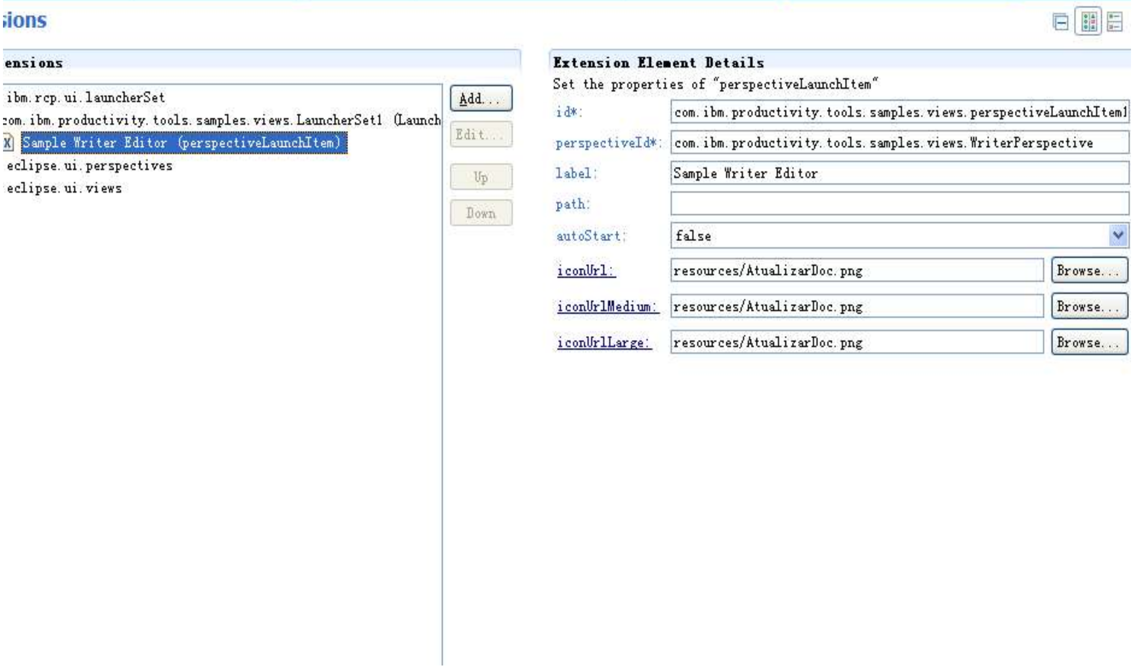
id*: tools.samples.views.LauncherSet1

label*: com.ibm.productivity.tools.sampl

- Right click the item “com.ibm.productivity.tools.samples.views.LauncherSet1”, select the menu item “**New->perspectiveLaunchItem**”.



- Set the properties of newly added **perspectiveLaunchItem** as the following; make sure the perspectiveID is “com.ibm.productivity.tools.samples.views.WriterPerspective”, then save the plugin.xml.

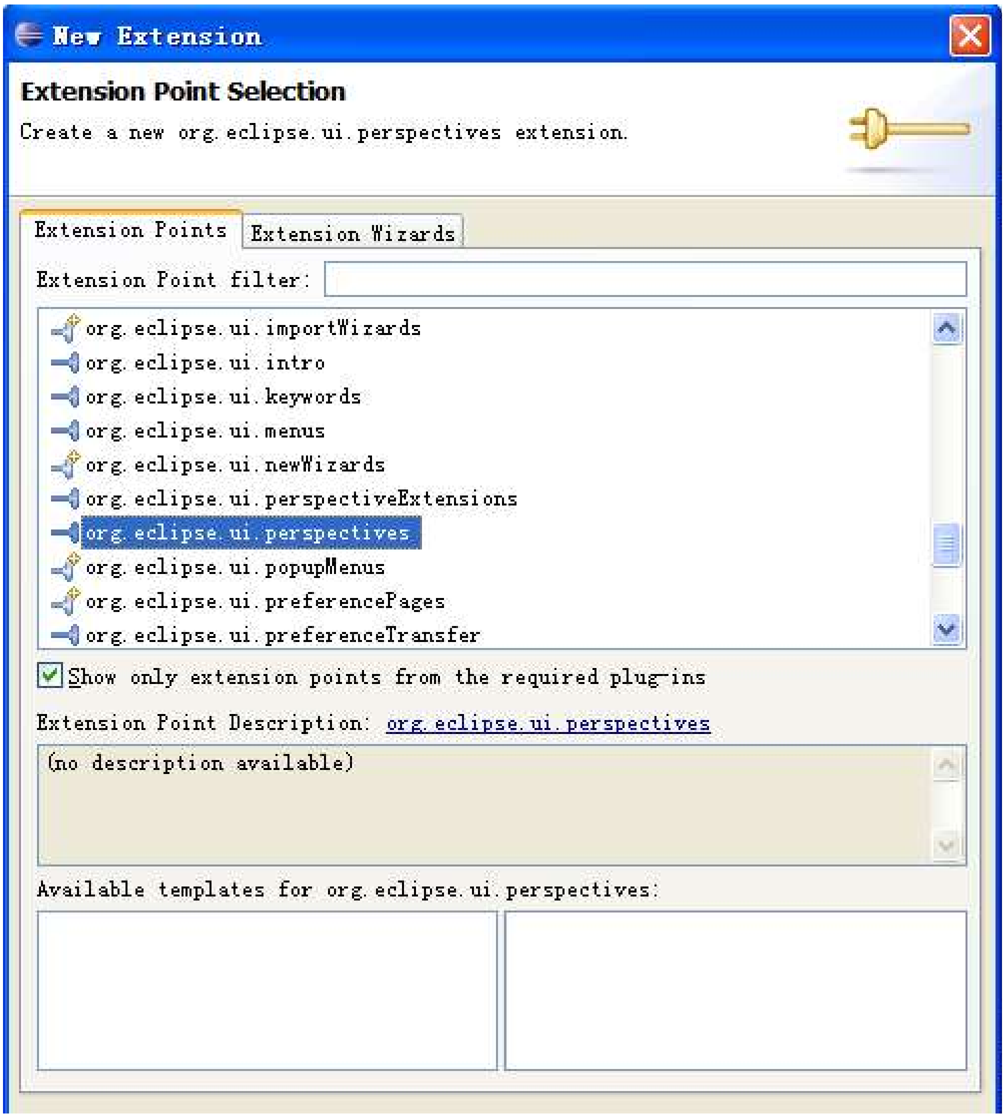


Extension Element Details

Set the properties of “perspectiveLaunchItem”

id*: com.ibm.productivity.tools.samples.views.perspectiveLaunchItem1
perspectiveId*: com.ibm.productivity.tools.samples.views.WriterPerspective
label: Sample Writer Editor
path:
autoStart: false
iconUrl: resources/AtualizarDoc.png
iconUrlMedium: resources/AtualizarDoc.png
iconUrlLarge: resources/AtualizarDoc.png

- Add an extension at extension point “org.eclipse.ui.perspective”.



Extension Point Selection

Create a new org.eclipse.ui.perspectives extension.

Extension Point filter:

- org.eclipse.ui.importWizards
- org.eclipse.ui.intro
- org.eclipse.ui.keywords
- org.eclipse.ui.menus
- org.eclipse.ui.newWizards
- org.eclipse.ui.perspectiveExtensions
- org.eclipse.ui.perspectives
- org.eclipse.ui.popupMenus
- org.eclipse.ui.preferencePages
- org.eclipse.ui.preferenceTransfer

☑ Show only extension points from the required plug-ins

Extension Point Description: org.eclipse.ui.perspectives

(no description available)

Available templates for org.eclipse.ui.perspectives:

Switch to the plugin.xml tab; change the extension declaration of the added perspectives extension point as the following:

```
      <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>
```

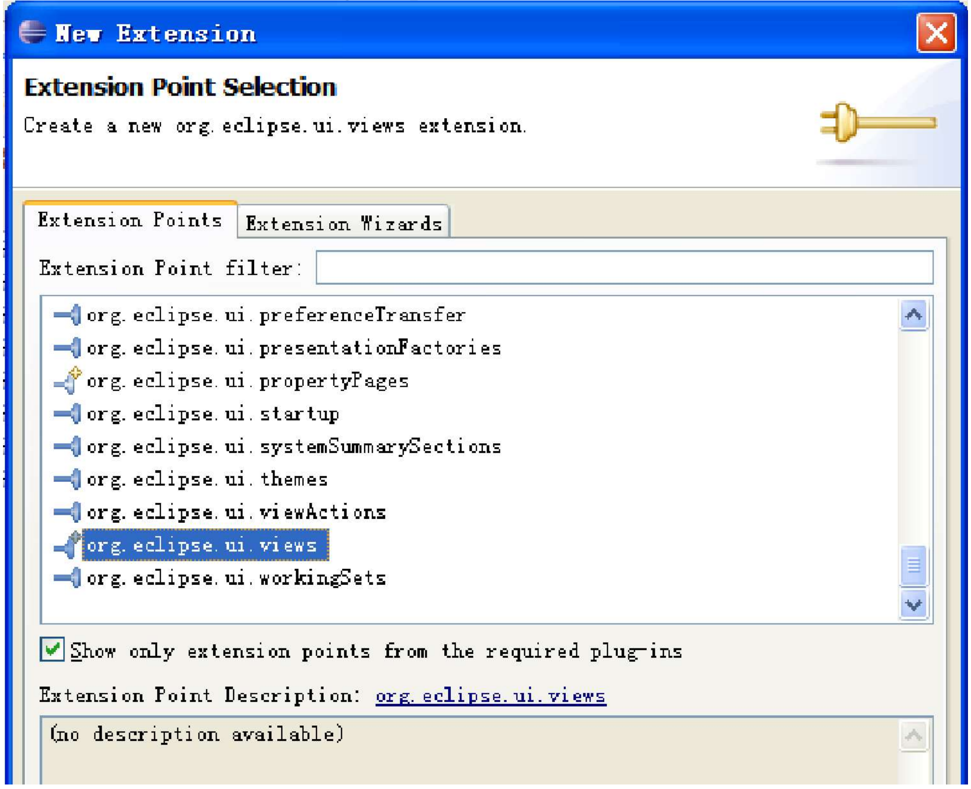
Create a java class named "com.ibm.productivity.tools.samples.views.WriterPerspective":

```
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
            ());
    }
}
```

Creating an editor view part

1. Select Extensions tab, click **Add…** button.
2. Add new extensions "org.eclipse.ui.views", then click Finish button.



Extension Point Selection

Create a new org.eclipse.ui.views extension.

Extension Point filter:

- org.eclipse.ui.preferenceTransfer
- org.eclipse.ui.presentationFactories
- org.eclipse.ui.propertyPages
- org.eclipse.ui.startup
- org.eclipse.ui.systemSummarySections
- org.eclipse.ui.themes
- org.eclipse.ui.viewActions
- org.eclipse.ui.views
- org.eclipse.ui.workingSets

☑ Show only extension points from the required plug-ins

Extension Point Description: org.eclipse.ui.views

(no description available)

Switch to "plugin.xml" page, add the markup as the following:

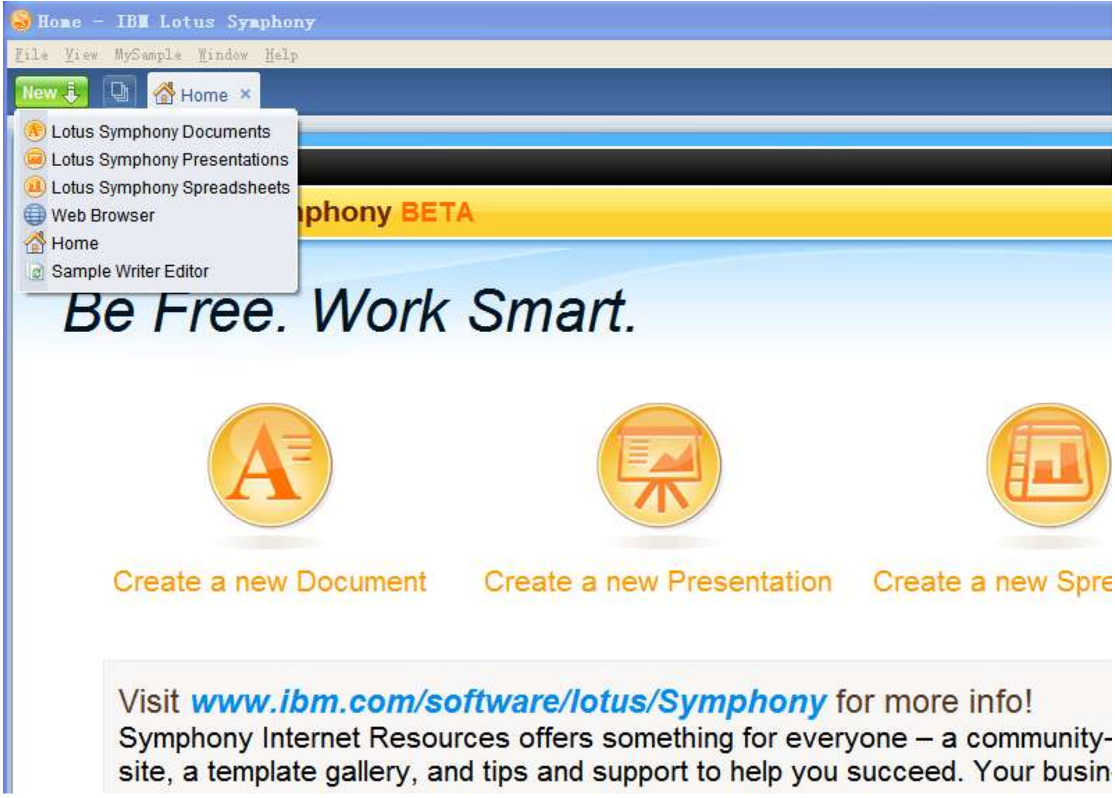
```
<extension
      point="org.eclipse.ui.views">
     <view
        allowMultiple="true"
        class="com.ibm.productivity.tools.samples.views.WriterView"
        id="com.ibm.productivity.tools.samples.views.WriterView"
        name="Writer View" />
</extension>
```

4. Create a view class: Select menu New>Class to create a new Java class for the view, set the Class arguments as below, and then click Finish button:
Package: com.ibm.productivity.tools.samples.views
Name: WriterView
Superclass: com.ibm.productivity.tools.ui.views.DefaultRichDocumentView

5. Implement the WriterView class as the following:

```
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 );
   }
}
```

Below is the result on display:



Home - IBM Lotus Symphony
File View MySample Window Help
New
Home
Lotus Symphony Documents
Lotus Symphony Presentations
Lotus Symphony Spreadsheets
Web Browser
Home
Sample Writer Editor
phony BETA
Be Free. Work Smart.
Create a new Document
Create a new Presentation
Create a new Spre

Visit www.ibm.com/software/lotus/Symphony for more info!
Symphony Internet Resources offers something for everyone – a community-
site, a template gallery, and tips and support to help you succeed. Your busin

Part 6. Appendixes

Appendix . References

For Lotus Expeditor and Lotus Expeditor toolkit, please reference to the following site:
http://www.ibm.com/software/lotus/products/expeditor/
http://www-128.ibm.com/developerworks/lotus/products/expeditor/

For Lotus Notes 8, please reference to the following site:
http://www-306.ibm.com/software/lotus/products/notes/

For composite application, please reference to the following site:
http://www.ibm.com/developerworks/lotus/composite-apps/
http://www-306.ibm.com/software/lotus/products/notes/compositeapplications.html

Appendix . Notices

Notices

Copyright

Licensed Materials - Property of IBM

Lotus Software
IBM Software Group
One Rogers Street
Cambridge, MA 02142

Revision History:
Original material produced for IBM Lotus Symphony Release Beta 4.

List of Trademarks

IBM, the IBM logo, AIX, DB2, Domino, iSeries, i5/OS, Lotus, Lotus Notes,