



Lotus Symphony Developer's Guide



Lotus Symphony Developer's Guide

Note

Before using this information and the product it supports, read the information in "Part 9. Appendixes" on page 141.

This edition applies to version release 1.3 of IBM Lotus Symphony and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 2003, 2009.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Part 1. About this publication 1

Chapter 1. Intended audience.	1
Chapter 2. Requirements	1
Chapter 3. How to use this guide	1
Chapter 4. The Lotus Symphony toolkit	3

Part 2. Product overview 7

Chapter 1. Introduction to Lotus Symphony	7
Chapter 2. Introduction to Lotus Symphony in Lotus Notes.	7
Chapter 3. Lotus Symphony and Lotus Notes architecture.	7
3.1 Overview of the Lotus Symphony architecture	7
3.2 Overview of Lotus Notes architecture	8
3.3 Overview of Eclipse.	8
3.4 Overview of Lotus Expeditor.	9
3.5 OpenOffice.org	10
Chapter 4. Lotus Symphony programming model	11
Chapter 5 Understanding the development environment	12

Part 3. Designing Lotus Symphony applications 13

Chapter 1. Rich client applications.	13
Chapter 2. Lotus Symphony back-end service	13
Chapter 3. Business logic	14
Chapter 4. Lotus Expeditor toolkit for Lotus Symphony developers	14
4.1. Update from jclDesktop to J2SE	14
Chapter 5. Packaging and deployment	15
5.1 Design and develop components with Lotus Expeditor toolkit	15
5.2 Group components into features with the Lotus Expeditor toolkit	15
5.3 Package the features into the update site with the Lotus Expeditor toolkit	15
5.4 Distribute the update site	15
5.5 Deploy the update site into Lotus Symphony	15
Chapter 6. Globalization	16
Chapter 7. Cross platform considerations	16
Chapter 8. Developing Applications for Lotus Symphony and for Lotus Symphony in Lotus Notes.	16

Part 4. Extending Lotus Symphony . . . 17

Chapter 1. Setting up the integrated development environment	17
Chapter 2. Customizing the Lotus Symphony user interface	20
2.1 Adding a sample menu	20
2.2 Adding a control to the toolbar	22
2.3 Adding to the launcher button.	24
2.4 Adding a new view in the shelf view	25
2.5 Using the auto recognizer	27
2.6 Adding an item to the status bar	32
2.7 Adding a preferences page	34

Chapter 3. Lotus Symphony Java APIs and extension points	37
3.1 Selection service	37
3.2 RichDocumentViewFactory	40
3.3 RichDocumentView	42
Chapter 4. Using the UNO API to access a document model	44
Chapter 5. Packaging and deploying your plug-ins	47
5.1 Prepare custom plug-in for deployment.	47
5.2 Create a feature and an Eclipse location update site	48
5.3 Install a custom Lotus Symphony application	52
5.4 Disable or enable custom Lotus Symphony applications	54
5.5 Uninstall custom Lotus Symphony application	56

Part 5. The Lotus Symphony Object Model 57

Chapter 1. Lotus Symphony Document Object Model	57
1.1 Overview	57
1.2 Handling Lotus Symphony documents	57
1.3 Working with Lotus Symphony document	59
Chapter 2. Lotus Symphony Spreadsheet Object Model	66
2.1 Overview	66
2.2 Handling Lotus Symphony spreadsheets	67
2.3 Working with Lotus Symphony spreadsheets	69
Chapter 3. Lotus Symphony Presentation object model	74
3.1 Overview	74
3.2 Handling Lotus Symphony presentations	75

Part 6. Lotus Expeditor and UNO Programming 79

Chapter 1. Developing Lotus Expeditor applications	79
Chapter 2. UNO Programming	80
2.1 Getting the global service factory	80
2.2 Using the import and export functions	82
2.3 Text documents	86
2.4 Spreadsheets	91

Part 7. Sample plug-ins 95

Chapter 1. Hello World sample plug-in	96
1.1 Creating a new plug-in	96
1.2 Adding the plug-in dependency	96
1.3 Adding a side shelf element.	96
1.4 Running the application	99
Chapter 2. Editor View sample plug-in	101
2.1 Creating a plug-in	101
2.2 Creating a new button	102
2.3 Creating an editor view part	105
Chapter 3. Spreadsheet sample plug-in	106
3.1 Introduction to the scenario	107

3.2 Preview of the result	107
3.3 Prepare your development environment	108
3.4 Deploying the sample	108
3.5 Creating the sample	109
3.6 Core code demonstration	112
3.7 Extending the sample	113
Chapter 4. Writer Sample plug-in	113
4.1 Introduction to the scenario	114
4.2. Preview of the result	114
4.3 Deploying the sample	115
4.4 Using the sample	115
4.5 Building the sample	117
Chapter 5. Presentation sample plug-in	120
5.1 Introduction to the scenario	121
5.2 Preview of the result.	121
5.3 Prepare your development environment	121
5.4 Deploying the sample	121
5.5 Creating the sample	121
5.6 Core code demonstration	123
5.7 Extending the sample	126
Chapter 6. C2A sample plug-in	126
Chapter 7. Customizing a Sample plug-in	127
7.1 Introduction to the scenario	127
7.2 Preview of the result.	128
7.3 Prepare development environment	128
7.4 Deploying the sample	128

7.5 Creating the sample	128
7.6 Core code demonstration	132
7.7 Extending the sample	134
Chapter 8. Convertor Sample plug-in	134
8.1 Introduction to the scenario	135
8.2 Preview of the result.	135
8.3 Prepare development environment	135
8.4 Deploying the sample	135
8.5 Design overview	135
8.6 Creating the sample	136
8.7 Core code demonstration	137
8.8 Extending the sample	138

Part 8. Troubleshooting and support 139

Chapter 1. Troubleshooting the development environment.	139
Chapter 2. Troubleshooting during application development	139
Chapter 3. Troubleshooting during deployment	139
Chapter 4. Contacting support.	140

Part 9. Appendixes 141

Appendix . References	141
Appendix . Notices	141

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 Web site at: <http://symphony.lotus.com/software/lotus/symphony/developers.jspa>.

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

Chapter 2. Requirements

To use the Lotus Symphony toolkit for Lotus Symphony, the Eclipse 3.4.0 development environment on Microsoft® Windows® XP or SUSE Linux® Enterprise 10 sp 1 is required.

To build samples, you must install Lotus Symphony. Samples from this toolkit can be deployed into Lotus Symphony on all operating systems that are supported by Lotus Symphony.

For detailed information about software requirements for the Lotus Symphony toolkit, see the `readme.txt` file that is included with the toolkit. For the detailed information about this toolkit, refer to Part 1 Chapter 4.

Chapter 3. How to use this guide

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

Part 1: Chapter 4 : introduces the main component of the Lotus Symphony developer's toolkit and explains how to begin your Lotus Symphony development journey.

Part 2: Product overview

1. Chapter 1: introduces what Lotus Symphony is.
2. Chapter 2: introduces the embedded Lotus Symphony in the Lotus Notes.
3. Chapter 3: introduces the Lotus Symphony architecture and the components.
4. Chapter 4: introduces the programming model for custom Lotus Symphony development.
5. Chapter 5: introduces the Lotus Symphony development environment.

Part 3: Designing Lotus Symphony applications

1. Chapter 1: introduces the rich client application.
2. Chapter 2: introduces Lotus Symphony back-end service.
3. Chapter 3: introduces two ways to build office applications.
4. Chapter 4: describes the Lotus Expeditor toolkit for Lotus Symphony application developers.
5. Chapter 5: describes Lotus Symphony application's packaging and deployment.
6. Chapter 6: describes globalization support in Lotus Symphony.
7. Chapter 7: describes multi-platform of Lotus Symphony application.
8. Chapter 8: describes developing applications for Lotus Symphony and Lotus Symphony in Lotus Notes.

Part 4: Extending Lotus Symphony

1. Chapter 1: describes step-by-step how to set up a custom Lotus Symphony development environment.
2. Chapter 2: introduces how to customize the user Lotus Symphony interface, such as custom menus, toolbars, launcher items, side shelf, auto recognizer, status bar and preferences.
3. Chapter 3: introduces how to use the Lotus Symphony java APIs and extensions in the Lotus Symphony toolkit.
4. Chapter 4: introduces how to use UNO APIs to operate three kinds of document models after getting them from Lotus Symphony APIs.
5. Chapter 5: describes step-by-step how to deploy a custom Lotus Symphony application and manage it.

Part 5: Lotus Symphony Object Model

- Chapter 1: introduces how to use Lotus Symphony Document Object Model to access text documents.
- Chapter 2: introduces how to use Lotus Symphony Spreadsheet Object Model to access spreadsheet documents.
- Chapter 3: introduces how to use Lotus Symphony Presentation Object Model to access presentation documents.

Part 6: Lotus Expeditor and UNO programming

1. Chapter 1: introduces developing applications on the Lotus Expeditor platform.
2. Chapter 2: introduces how to use UNO's function in Lotus Symphony development, such as get a global service factory, use the import/export function, export documents to HTML file or JPEG image.

Part 7: Example plug-ins

1. Chapter 1: describes step-by-step how to create a hello world plug-in on Lotus Symphony. This sample adds a side shelf to say hello.
2. Chapter 2: demonstrates how to create a simple editor in a view on Lotus Symphony. This sample creates a sample editor in a view.
3. Chapter 3: demonstrates how to operate a spreadsheet on a Lotus Symphony side shelf. This sample shows how to open a spreadsheet by opening two demo files, set and get a cell's value and its address dynamically, how to create a chart and how to create a DataPilot table.
4. Chapter 4: demonstrate how to manipulate documents programmatically on a Lotus Symphony side shelf. This sample creates a side shelf for operating a writer document, such as creating sections, creating tables, and creating user defined fields.

5. Chapter 5: demonstrates how to use presentations programmatically on a Lotus Symphony side shelf. This sample creates a side shelf for operations in a presentation, such as opening a presentation, inserting, removing, and copying pages.
6. Chapter 6: demonstrates how to develop C2A applications based on Lotus Symphony. Two keywords are predefined in the sample recognizer: PropertyBroker and AutoRecognizer. Auto Recognizer engine will call the detector to check if there are keywords found. When users click the context menu for recognized tags, a message window is opened.
7. Chapter 7: shows a typical sample application on the Lotus Expeditor platform on which Lotus Symphony development is based. This sample creates a custom perspective and adds three views, an early startup when Lotus Symphony was startup, a status bar, a custom help, a custom preference page, and a simple globalization customizing sample.
8. Chapter 8: shows how to load documents implicitly and export to HTML or JPEG by document type. This sample shows a button for loading documents implicitly; a button for exporting and converting the loaded document into an HTML file or JPEG image according its type, ODT and ODS into the HTML file, or ODP into JPEG image array. The sample also shows a setter and getter methods to show how to access metadata.

Part 8: Troubleshooting and support

1. Chapter 1: describes problems and solutions about the development environment setting up.
2. Chapter 2: describes problems and solutions about Lotus Symphony hang conditions when executing UNO call in Java code.
3. Chapter 3: describes problems and solutions about applications that do not work when plug-ins are deployed
4. Chapter 4: describes how to get support from Lotus Symphony forum.

Part 9: Appendixes

The Appendixes of this developers guide.

Chapter 4. The Lotus Symphony toolkit

To access the toolkit, see <http://symphony.lotus.com/software/lotus/symphony/developers.nsf/home>. The Lotus Symphony download page contains links to all the documentation and downloads. The Lotus Symphony download page contains links to Lotus Symphony toolkit. You can extract it on your local system.

The toolkit has developer's guides to help you learn more about Lotus Symphony. This developer's guide offer a detailed introduction of developing applications on the product. Plug-ins and features which are introduced in the developer's guides can be found with source code in samples directory. Also, if you want to do further development on the products, refer to the Java APIs which are supplied in javadoc directory. To develop applications on Lotus Symphony, files in the update_site directory help to configure the Eclipse environment.

4.1 Get started with the toolkit

To get a quick development experience using the Lotus Symphony toolkit, create a "Hello world" plug-in. To create this plug-in, do the following steps:

1. Set up the development environment. Refer to Part 4 Chapter 1: Setting up the integrated development environment.
2. Create a “Hello world” plug-in. Refer to Part 7 Chapter 1: Hello world sample plug-in.

To get more experience, the next best choice is the tutorial plug-in sample **DocumentWorkflow** and the tutorial document in the Lotus Symphony toolkit.

4.2 Document

In Lotus Symphony toolkit, the following document and tutorial are supplied.

- Lotus Symphony Developer’s Guide.pdf
This guide.
- tutorial-DocumentWorkflow
A plug-in sample and a tutorial for beginning Lotus Symphony development.

4.3 Samples

There are two ways to deploy the plug-ins and features. One way is deploying them to Lotus Symphony directly. Refer to Part 4 Chapter 5 for instructions on how to deploy the package into Lotus Symphony. Another way is to import them from the folder features and plugins into Eclipse, and then attaching them to Lotus Symphony.

These samples show how to develop custom plug-ins and applications, how to use the Lotus Symphony APIs and other support functions to add custom UI (user interface) elements and create Lotus Symphony documents. These plug-ins run on Lotus Symphony.

The list of sample plug-ins is as follows:

- `com.ibm.productivity.tools.samples.helloworld`
- `com.ibm.productivity.tools.samples.DocumentWorkflow`
- `com.ibm.productivity.tools.samples.views`
- `com.ibm.productivity.tools.samples.spreadsheet`
- `com.ibm.productivity.tools.samples.writer`
- `com.ibm.productivity.tools.samples.customizing`
- `com.ibm.productivity.tools.samples.convertor`
- `com.ibm.productivity.tools.samples.presentation`
- `com.ibm.productivity.tools.samples.C2A`

The features are used with an update site installation. When you are deploying feature on Lotus Symphony, select `com.ibm.productivity.tools.samples.symphony.feature`.

The plug-ins and features found in this guide can be run directly from the Lotus Symphony or Lotus Notes 8.5 development environment. For instructions on accessing and running the samples, refer to Part 7. Sample Plug-ins in this guide.

4.4 Java document

There are two parts of Javadoc:

- symphony consists of the API documentation for the Lotus Symphony APIs.

- common consists of the API documentation for the Lotus Symphony Object Model APIs.

4.5 Update site

The `update_site` folder contains an update site that can help you configure the Eclipse development environment for Lotus Symphony automatically. For instructions on how to install this update site, please refer to Part 4 Chapter 1.

4.6 Related documentation

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

Part 2. Product overview

Chapter 1. Introduction to Lotus Symphony

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

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

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

Chapter 2. Introduction to Lotus Symphony in Lotus Notes

Lotus Symphony wraps Lotus Symphony applications as Eclipse components to provide rich document editor service to Lotus Notes. It is a stand-alone office productivity suite composed of IBM Lotus Symphony documents, IBM Lotus Symphony spreadsheets, IBM Lotus Symphony presentations and many other document editors.

In Lotus Notes, it also supplies composites application which enable users to integrate Lotus Symphony with other services, like web services into one single screen on Lotus Notes. At the same time, developers can use LotusScript® programming to load Lotus Symphony documents in Lotus Notes. See introduction of the Lotus Symphony Developer's Guide for Lotus Notes for more information.

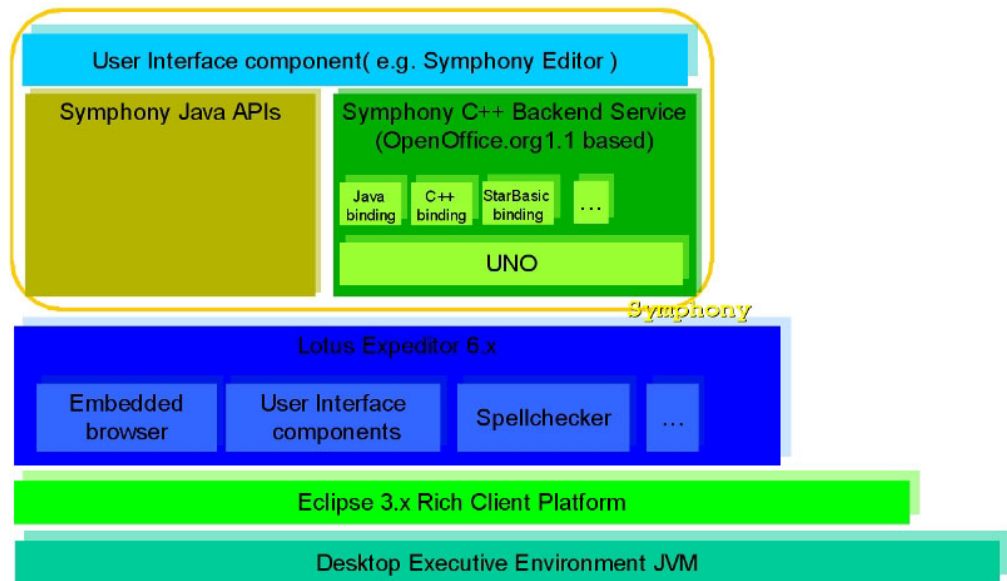
Chapter 3. Lotus Symphony and Lotus Notes architecture

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.

3.1 Overview of the Lotus Symphony architecture

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

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



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

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

3.2 Overview of Lotus Notes architecture

For details about Lotus Notes architecture, refer to the *Lotus Symphony Developer's Guide for Notes* in Lotus Symphony toolkit.

3.3 Overview of Eclipse

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

<http://www.eclipse.org>

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

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

Component	Description
Platform runtime	Provides the foundational support for plug-ins and for the plug-in registry, a mechanism for declaring extension points, and for extending objects dynamically. The Eclipse runtime uses the standard OSGi framework to define how plug-ins are packaged.

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

3.4 Overview of Lotus Expeditor

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

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

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

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

Service	Description
Application manager	Enables users to directly install applications and components from standard Eclipse update sites onto managed clients.
Embedded browser	Provides a configurable embedded Web browser.
Spell check	Is used to check misspelled words in document. It is based on the text analyze framework.
Personalities	Defines the framework that the platform uses to determine what perspectives or windows, menus, actions, action bar items, and status line controls are displayed when the application starts.

Service	Description
Application launcher	Is represented in the user interface as a button with a drop-down menu that contains the list of applications available to the user.
Eclipse UI extensions	Common UI extensions provided by the Eclipse platform.

3.4.1 DEE (Desktop Execute Environment) SDK VM

The default Java™ Runtime Environment (JRE) of Lotus® Expeditor is the IBM J9 VM with the desktop Java EE class libraries, an IBM-optimized subset of Java 5 that offers a smaller footprint and faster class loading than standard Java Runtime Environments. The desktop Java EE runtime environment also leverages a technology known as shared classes to improve runtime performance and reduce memory footprint. Shared classes are cached dynamically during platform operation. However, the default configuration caches the shared classes without debug information. This can inhibit the ability to set breakpoints. More information on shared classes is available below. The reduction in disk footprint that the desktop Java EE class libraries and the J9 VM provide is quite substantial. A standard Java 5 JRE takes approximately 65 MB of space on disk, while the desktop Java EE runtime environment requires only approximately 17 MB of space for installation. When creating lightweight Lotus Expeditor-based client applications, a difference of over 45 MB can make a difference in client download and deployment time. To achieve this reduction in footprint, several components of the Java 5 class libraries are not included in desktop Java EE. Most specifically, AWT and SWING. The preferred windowing API for Lotus Expeditor is SWT. SWT is provided as part of the core Lotus Expeditor platform. The list of classes that have been removed from desktop Java EE is not limited to AWT and SWING. The J9 VM used in desktop Java EE is the same code base as the IBM® Java SE 6 VM. It includes the same JIT (Testarossa), and the same garbage collector with a customizable policy and the same shared classes support.

The desktop Java EE VM that is included as part of most client runtime environments has been modified to improve the performance and footprint. As a side effect, much of the information that is needed by the Java compiler and the Eclipse IDE is removed from the VM. Therefore, it is necessary to include the SDK version of the DEE VM in the client environment. The Lotus Expeditor toolkit is able to load and use the SDK VM from the client platform. The standard and the SDK version of the VM is available as part of the Expeditor product build. The SDK version of the DEE VM is packaged in the `com.ibm.rcp.dee.sdk.[platform].x86` plug-in.

3.4.2 The profile of Lotus Expeditor used by Lotus Symphony

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

3.5 OpenOffice.org

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

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

Chapter 4. Lotus Symphony programming model

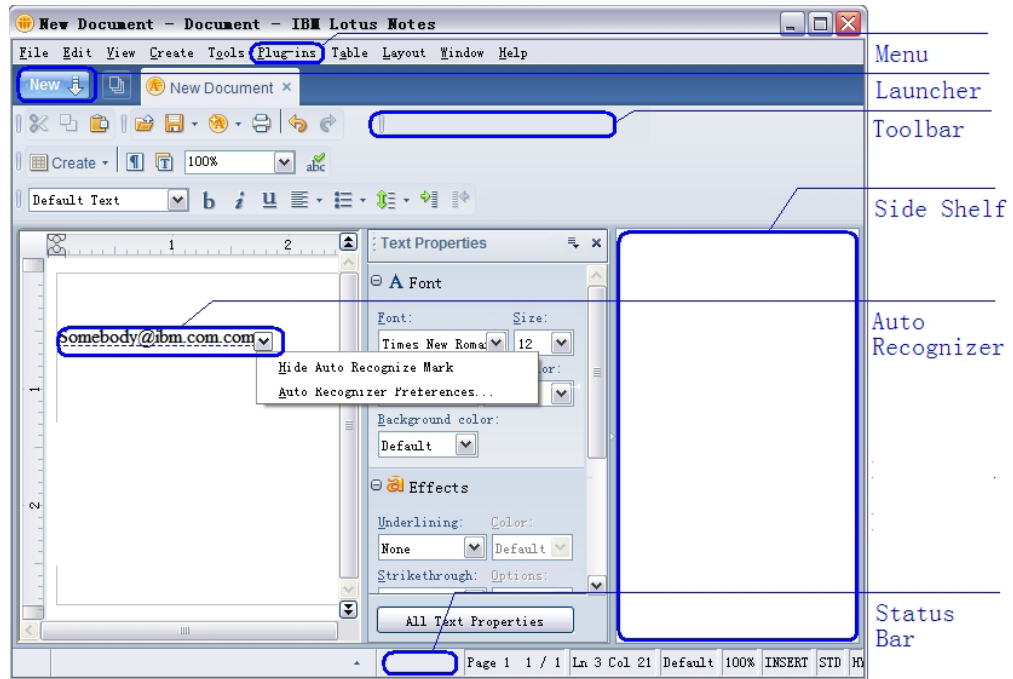
Lotus Symphony is the combination of Eclipse-based Lotus Expeditor and OpenOffice.org. Both of these products provide rich APIs for application integration. In Lotus Symphony, the OpenOffice.org window is re-parented to a SWT control in Eclipse. Most of the user interface items that you can add are provided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

The programming model of Lotus Symphony can be described as follows:

- User interface integration is based on Eclipse and Lotus Expeditor extension points and a plug-in framework.
- The document content level API is based on the Lotus Symphony Object Model APIs and OpenOffice.org UNO capability.
- The Lotus Symphony API focuses on the integration between OpenOffice.org and Eclipse and Lotus Expeditor.
- The add-in mechanism is based on Lotus Expeditor application manager.

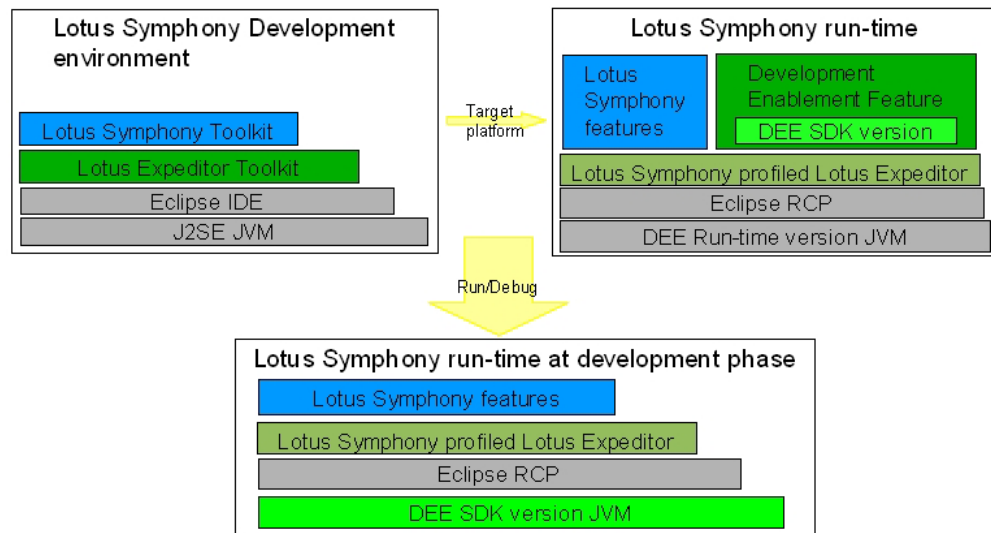
In this way, Lotus Symphony inherits the user interface flexibility of Eclipse and Lotus Expeditor and the rich functionality of UNO APIs.

The following screen capture shows the user interface items.



Chapter 5 Understanding the development environment

The development environment consists of the target platform and the integrated development environment. The following figure describes the overall structure of development environment.



The target platform is the Lotus Symphony runtime, designed for users, with a development enablement feature into the runtime. The development enablement feature contains a desktop execute environment SDK version JVM and Javadoc plug-ins.

The integrated development environment (IDE) is based on the Eclipse IDE. The Lotus Expeditor toolkit provides the documentation and examples for developers. The Lotus Symphony toolkit provides a toolkit configuration feature to configure the development environment.

Part 3. Designing Lotus Symphony applications

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

Chapter 1. Rich client applications

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

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

You can open, save, close documents and access the document content using the Lotus Symphony Object Model APIs. Lotus Symphony Object Model provides entire APIs to control documents. For more information, refer to Part 5.

If you want to access the document model of a loaded document, use the UNO API. There is typical usage provided in Part 6 Chapter 2. You can also find samples in the Lotus Symphony toolkit.

Chapter 2. Lotus Symphony back-end service

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

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

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

Chapter 3. Business logic

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

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

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

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

Chapter 4. Lotus Expeditor toolkit for Lotus Symphony developers

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

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

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

4.1. Update from jclDesktop to J2SE

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

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

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

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

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

Chapter 5. Packaging and deployment

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

5.1 Design and develop components with Lotus Expeditor toolkit

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

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

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

5.2 Group components into features with the Lotus Expeditor toolkit

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

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

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

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

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

5.4 Distribute the update site

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

5.5 Deploy the update site into Lotus Symphony

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

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

Chapter 6. Globalization

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

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

Chapter 7. Cross platform considerations

In the development phase, use Windows XP or SUSE Linux Enterprise 10 as the primary development environment. The component developed can be deployed into all platforms supported by Lotus Symphony. The Java APIs provided by Lotus Symphony or Lotus Expeditor are platform independent. UNO APIs are also designed for cross platform applications. Some functions can be platform dependent, for example, OLE Automation bridge of UNO is only available on Windows operating system.

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

Lotus Symphony is available as a standalone editor product, it is also provided in Lotus Notes 8 client version. Either it is or it is not. The same code base is used in the two products. You can design applications that work for both products. There are still some issues that you should be aware of in the design phase:

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

Part 4. Extending Lotus Symphony

Chapter 1. Setting up the integrated development environment

The integrated development environment (IDE) is based on Eclipse 3.4 and Lotus Symphony. All the steps in this procedure are for a Windows operating system, but the process on the Linux operating system is similar. If you have any questions during the set up process, refer to Part 8 Troubleshooting and support or get help from the Lotus Symphony forum: Lotus Symphony forum.

1. Install Lotus Symphony and the Lotus Symphony toolkit.
 - a. Download Lotus Symphony from the Lotus Symphony Web site and Lotus Symphony toolkit from the download page.
 - b. Install Lotus Symphony to a local disk, for example, D:\IBM\Lotus\Symphony as <Symphony installation home>.
 - c. Unzip Lotus Symphony toolkit to a local disk. For details about the toolkit, refer to Part 1 Chapter 4.
2. Enhance Lotus Symphony with the Lotus Symphony toolkit for development.
 - a. Start Lotus Symphony. From the Lotus Symphony main menu, click **File > Application > Install**. On the **Install/Update** window, select **Search for new features to install**, and then click **Next**.
 - b. Select **Add Zip/Jar Location**, and select the development_enablement_updatesite.zip update site from the Lotus Symphony Toolkit's update_sites folder, and then click **Open > OK > Finish**.
 - c. Select **Symphony Development Feature**, and click **Next**. If you agree with the license, select to accept the license and click **Next > Finish**.
 - d. Wait for the installation to finish and select **Restart Now**. After it restart, close Lotus Symphony.
3. Install Eclipse SDK 3.4 and the Lotus Symphony toolkit configuration with the Lotus Expeditor toolkit (update package).
 - a. Download Eclipse 3.4 SDK from <http://www.eclipse.org/downloads/> and extract it to a local disk.

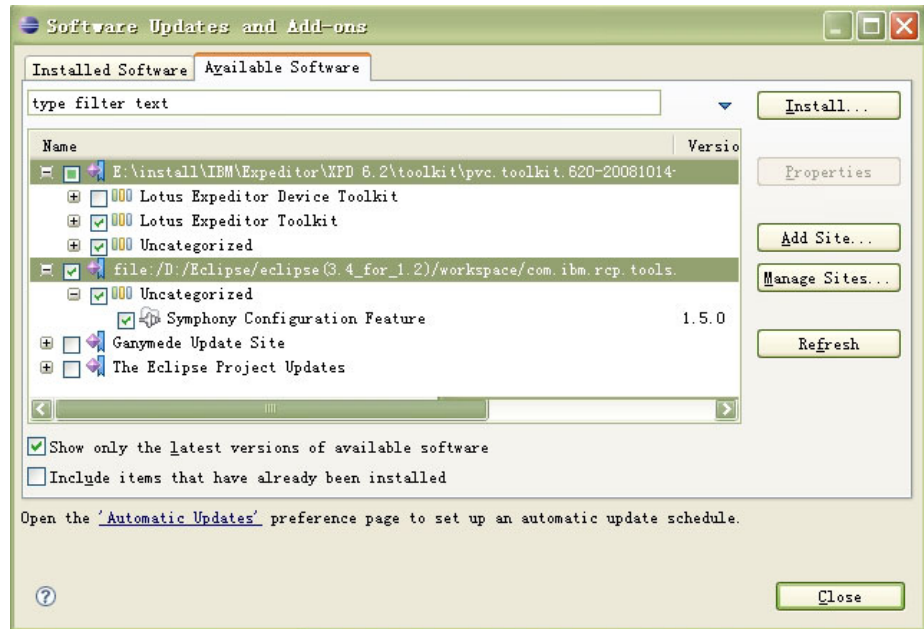
Note: For Eclipse 3.4 SDK reference platforms, refer to Eclipse embedded readme or readme_eclipse_3.4

- b. Download the Lotus Expeditor toolkit (use the update package of 6.2.0 edition).

Note: For Lotus Expeditor Toolkit update site installation, JDK version 5.0(1.5) is required.

- c. Start the Eclipse IDE. From the main menu, click **Help > Software Updates**. On the **Software Update and Add-ons** window, click the **Available Software** tab. Click **Add Site > Archive**, select the Lotus Expeditor Toolkit update site from Lotus Expeditor Toolkit (for example, pvc.toolkit.620-20081017-1037-site.zip), and then click **OK**.
- d. Click **Add Site > Archive** again, and select the toolkit_configuration_updatesite.zip update site from the Lotus Symphony toolkit's update_sites folder, and then click **OK**.

- e. Select all the features available except Lotus Expeditoer Device Toolkit feature as showing in following figure, and then click **Install**. After the installation is finished, click **Next**.



Note: When selecting Lotus Expeditoer Toolkit feature on Red Hat or SuSE operating systems, it might throw a `NullPointerException`. This exception does not cause problems during installation.

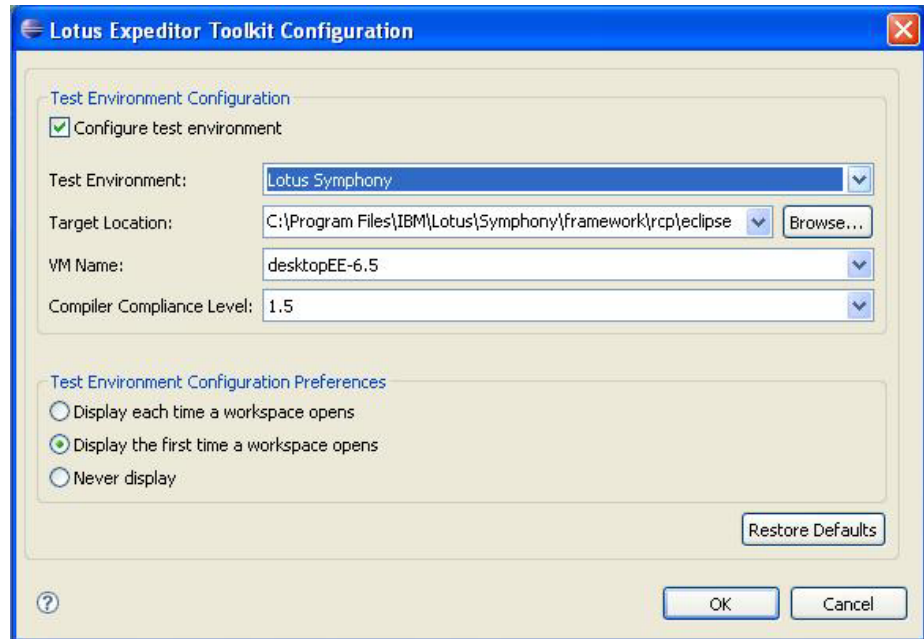
- f. In the **Feature License** window, read the licensing information for each feature that you are installing, and if you agree with the license, select to accept the license and click **Next**.
- g. Click **Finish** to begin the installation. When installation completes, you are prompted to restart your IDE for changes to take effect. Click **Yes** to continue.

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

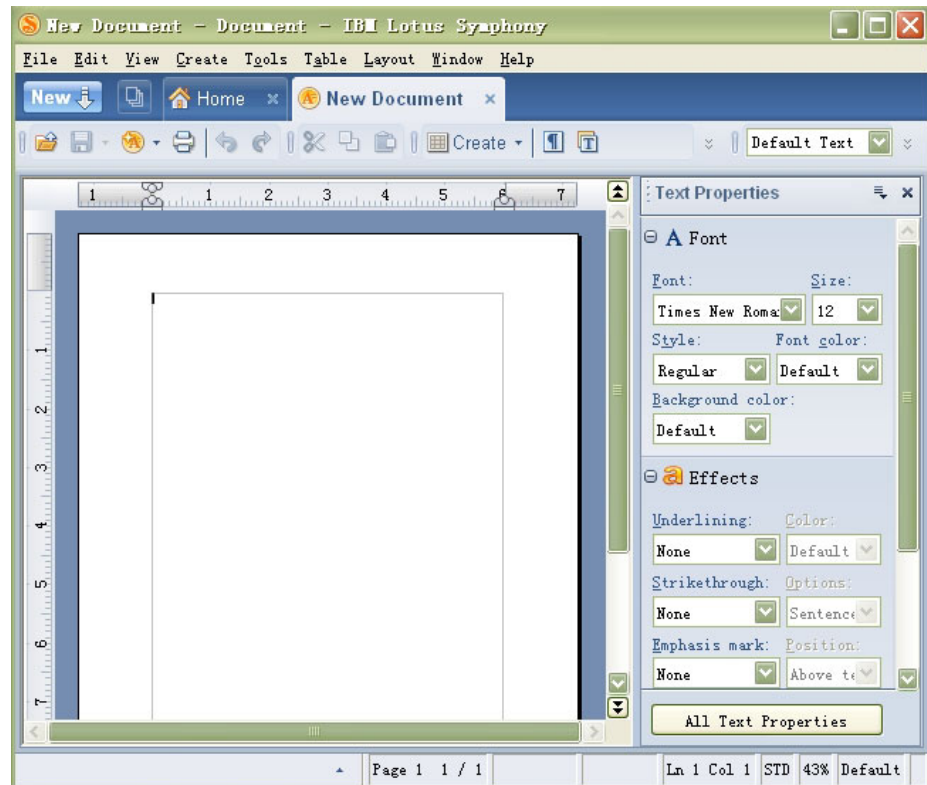
4. Configure Lotus Symphony development support.

After restarting the IDE, the **Lotus Expeditoer Toolkit Configuration** window opens. To configure the toolkit, follow these steps:

- a. Select **Lotus Symphony** in the **Test Environment** list.
- b. Use the **Browse** button to select the **Eclipse** directory of the Lotus Symphony installation location, for example, <Symphony installation home>\framework\rcp\eclipse.



- c. Click **OK**.
5. Create your own project code in this Eclipse environment.
6. Launch Lotus Symphony.
 - a. Select **Run > Run Configuration** or **Run > Debug Configuration** from Eclipse main menu.
 - b. Select the **Client Services** launch type and click the new icon or double click Client Services, named the new configuration as Symphony.
 - c. Click **Run** or **Debug** to start Lotus Symphony.



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

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

Note: On Red Hat systems, sometimes a `java.lang.UnsatisfiedLinkError` exception is thrown when launching the Lotus Symphony. Try to fix it with the command similar to the following (*versionnumber* means the version number, you need change it according the version you installed) :

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

Chapter 2. Customizing the Lotus Symphony user interface

The following examples are all you need to build a plug-in project, and then edit the `plugin.xml` file directly by the code provided below. If you are not familiar with how to build a plug-in project, go to Part 7 Sample plug-ins to see the details.

2.1 Adding a sample menu

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

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

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

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

```
<extension point="org.eclipse.ui.actionSets">  
  <actionSet id="com.ibm.lotus.symphony.example.ui.actionSet"  
    label="example action set"  
    visible="true">  
    <menu  
      id="com.ibm.rcp.ui.pluginsmenu"  
      label="&Plug-ins"  
      path="additions">  
        <separator name="additions"/>  
      </menu>  
    <action id="com.ibm.lotus.symphony.example.ui.exampleAction"  
      menubarPath="com.ibm.rcp.ui.pluginsmenu/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 the Eclipse Rich Client Platform.

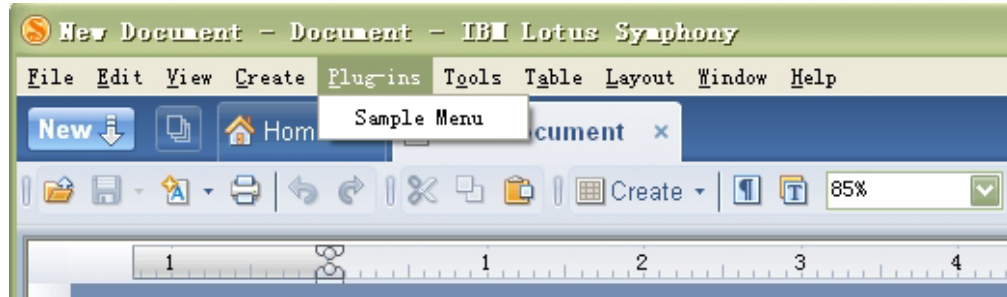
See Also

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

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

Example

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



2.2 Adding a control to the toolbar

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

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

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

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

3. Provide a class to define your control:

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

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

public class SampleControl extends SODCAActionContributionItem {

    public IAction createAction() {
        Action action = new Action() {
            public void run() {
                MessageDialog.openInformation(PlatformUI.getWorkbench()
                    .getActiveWorkbenchWindow().getShell(), "Information",
                    "Control pressed");
            }
        };
        action.setText("Sample");
        action.setToolTipText("Sample");
        // action.setImageDescriptor(Activator.imageDescriptorFromPlugin(
        //     Activator.PLUGIN_ID, "docs/itemCampo.png"));
        return action;
    }
}
```

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

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

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

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

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

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

Package

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

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

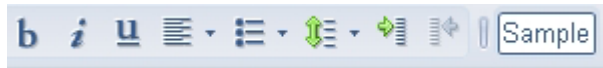
See Also

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

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

Example

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



2.3 Adding to the launcher button

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

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

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

The following markup adds a new perspective launch item:

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

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

    </LauncherSet>
  </extension>
```

Package

The extension point is provided by Lotus Expeditor.

See Also

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

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

Example



2.4 Adding a new view in the shelf view

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

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

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

1. Make sure that your plug-in have the following dependencies:
 - `com.ibm.productivity.tools.ui.views`
 - `com.ibm.productivity.tools.core`
 - `com.ibm.rcp.jfaceex`
 - `com.ibm.rcp.ui`
 - `com.ibm.rcp.swtex`

2. Extend the `com.ibm.rcp.ui.shelfViews` extension point in `plugin.xml` file:

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

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

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

Make sure that the following attributes are specified:

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

4. Implement the view class:

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

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

public class ShelfView extends ViewPart {

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

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

Package

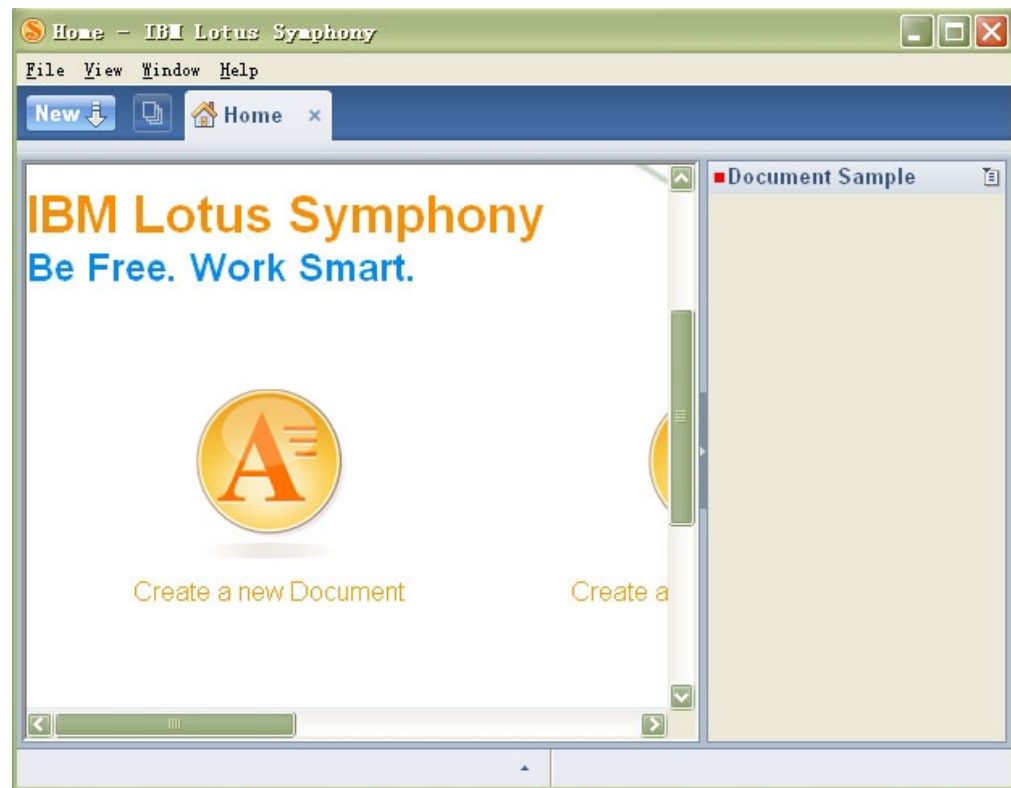
The extension point is provided by Lotus Expeditor.

See Also

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

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

Example



2.5 Using the auto recognizer

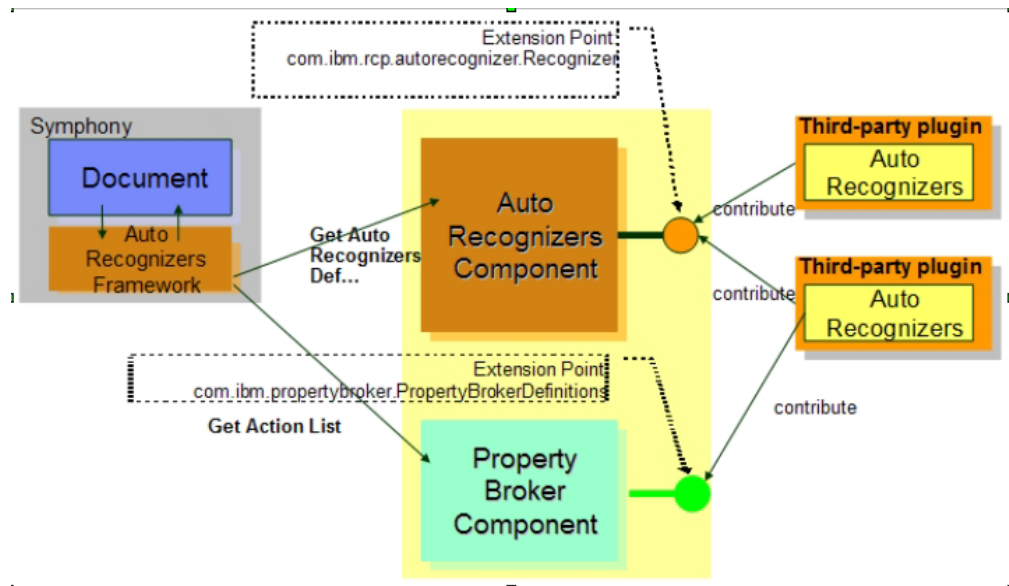
The auto recognizer is a framework to allow users to take actions based on the text that they input in the Lotus Symphony editor. It assists users to do extra operations on the content of a document by underlining items in a special pattern. It provides a gateway to provide further information and activities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

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

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

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

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



Adding the auto recognizer to the extension point

To add the auto recognizer, perform the following steps:

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

```

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

```

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

```

import java.util.ArrayList;

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

public class SampleDetector implements IDetect {

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

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

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

```

Add an action

To add an action, perform the following steps:

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

```

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

```

2. Define the `SampleAction.wsdl` file:

```

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

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

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

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

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

  <binding name="SampleBinding" type="tns:Sample_Service">
    <portlet:binding/>
    <operation name="sample_event">

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

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

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

```

3. Implement a SampleAction class:

```

import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.commands.IHandler;
import org.eclipse.core.commands.IHandlerListener;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;
import com.ibm.rcp.propertybroker.event.PropertyChangeEvent

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() {
            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

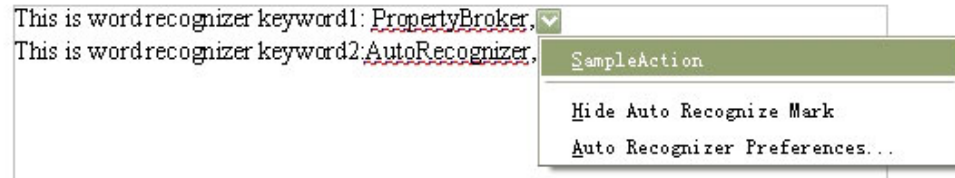
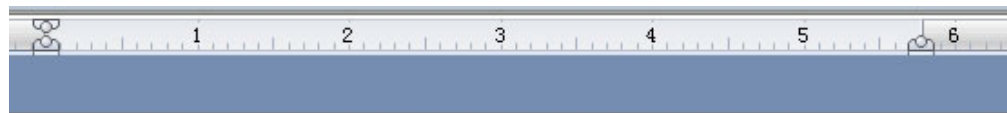
Propertybroker extension point in Lotus Expeditor:

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

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

Example

In following example, a plug-in defines that **PropertyBroker** and **AutoRecognizer** are two keywords, and a special action (in this example, **SampleAction**) is added to this pattern. When the keywords are found in the document, the words are underlined which indicates that this is a special pattern. If users move the cursor to the pattern, pull-down button displays and they can click the button to invoke pattern-related actions. The source code for this example is provided above.



2.6 Adding an item to the status bar

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

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

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

```

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

```

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

2. Implement the control class:

```

package com.ibm.Lotus.Symphony.example;

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

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

```

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

Package

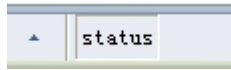
This extension point is provided by Lotus Expeditor.

See also

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

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

Example



2.7 Adding a preferences page

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

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

The `org.eclipse.ui.preferencePages` extension point lets you add pages to the Lotus Symphony preferences (**File > Preferences**). The preferences window presents a hierarchical list of user preference entries. Each entry displays a corresponding preference page when selected.

To add a preference page, complete the following steps:

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

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

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

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

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

Package

The extension point is provided by the Eclipse Rich Client Platform.

See also

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

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

Example



Chapter 3. Lotus Symphony Java APIs and extension points

3.1 Selection service

In Eclipse, the selection service provided by the Eclipse workbench allows efficient linking of different parts within the workbench window. Each workbench window has its own selection service instance. The service keeps track of the selection in the currently active part and propagates selection changes to all registered listeners. Such selection events occur when the selection in the current part is changed or when a different part is activated. Both can be triggered by user interaction or programmatically.

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

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

From the user's point of view, a selection is a set of highlighted text or objects in a view. Internally, a selection is a data structure holding the model objects which correspond to the graphical elements selected in the view. Almost all text or objects can be selected in the view for these kinds of applications: writer, spreadsheet, and

presentation. The selection can be presented in several ways and you can only get the text content from the selection. It might be possible to present the selection using HTML, ODF, or XML format.

Accessing the current selection

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

To access the current selection of current Lotus Symphony view:

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

Retrieving text content from the selection

To get the text content from the selection:

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

Tracking selection change

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

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

Note: If you want to get notified when the selection of document content changes, you must call the following code fragment to enable selection change service of document:

```
IPreferenceStore store = SuperODCPlugin.getInstance().getPreferenceStore();
//enable document selection service to listen the document content selection
//changes.
store.setValue("Enable_Selection", true);
```

After that, all selection changed within document, spreadsheet and presentation will be notified. It only affects documents that are opened after the enabling operation. You can disable document selection change service using the following

code fragment and it only affects documents that are opened after the disabling operation.

```
IPreferenceStore store = SuperODCPlugin.getInstance().getPreferenceStore();
store.setValue("Enable_Selection", false);
```

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

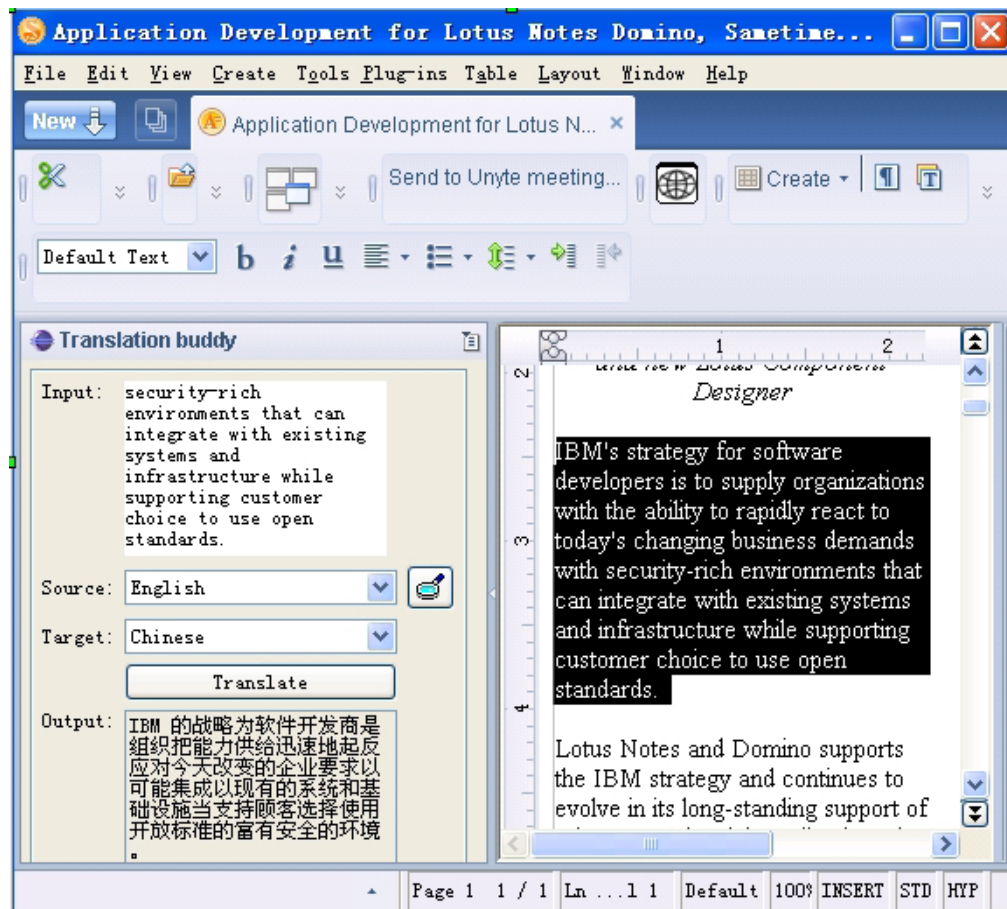
`com.ibm.productivity.tools.ui.views`

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 in the Input area of the view automatically.



3.2 RichDocumentViewFactory

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

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

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

Creating a new rich document view

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

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

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

Opening a local file in a new rich document view

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

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

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

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

Getting the list of opened rich document views

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

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

All rich document views opened in Lotus Symphony are returned.

Closing a rich document view

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

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

Registering the listener using the extension point

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

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

To use the listener, perform the following steps:

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

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

2. Implement a `RichDocumentViewListener` class:

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

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

Package

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

See also

javadoc in Lotus Symphony toolkit.

Example

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

3.3 RichDocumentView

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

Accessing existing RichDocumentView instances

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

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

You also can get a RichDocumentView instance from an active document:

```
IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
IWorkbenchPage page = window.getActivePage();

IAdaptable adaptable = ( IAdaptable )page.getActivePart();
RichDocumentView view = (RichDocumentView)adaptable
    .getAdapter(RichDocumentView.class);
System.out.println( view.getFilePath() );
```

Using DefaultRichDocumentView directly

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

Extending a new view

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

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

```
public class WriterView extends DefaultRichDocumentView {

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

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

    private void createWriter() {
        NewOperation operation = OperationFactory
            .createNewOperation(RichDocumentType.DOCUMENT_TYPE );
        operation.execute(this);
    }
}
```

Operations on rich documents

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

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

Monitoring operations

The following code example demonstrates how to detect that a document is loaded into the rich document view. The `WriterView` is created as above. The example code demonstrates how to add an operation listener into the `ViewPart` when the `ViewPart` is created. When a load operation is issued, the monitor is called. The `OperationListener` is applicable to all default operations and is documented in the Lotus Symphony Javadoc API:

```
public class WriterView extends DefaultRichDocumentView {

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

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

    private void monitorLoading() {
        OperationListener listener = new OperationListener(){
            public void afterExecute(Operation operation, RichDocumentView view) {
                if( operation instanceof LoadOperation ){
                    System.out.println( "document is loaded:"
                        + ( (LoadOperation)operation ).getFileName());
                    Object document = (( LoadOperation )operation).getUNOModel();
                    afterLoading( document);
                }
            }
            public void beforeExecute(Operation operation, RichDocumentView view){
                if( operation instanceof LoadOperation )
                    System.out.println( "document is about to be loaded:"
                        + ( (LoadOperation)operation ).getFileName());
            }
        };
        this.addOperationListener( listener );
    }
}
```

Chapter 4. Using the UNO API to access a document model

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

Accessing the document model

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

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

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

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

Setting the whole text of a document

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

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

Inserting a table in a document

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

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

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

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

Setting text in the current cursor

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

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

Using the spreadsheet document model

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

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

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

Setting the content of a cell

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

```
public void setCellText( XModel model, String content ){
    //query the sheet document
    XSpreadsheetDocument sheetDocument = ( XSpreadsheetDocument )
        UnoRuntime.queryInterface( XSpreadsheetDocument.class, model );
    XSpreadsheets xSheets = sheetDocument.getSheets();
    XSpreadsheet xSheet = null;
    try {
        XIndexAccess xSheetsIA = (XIndexAccess)UnoRuntime
            .queryInterface(XIndexAccess.class, xSheets );
        //get the first sheet in the document
        xSheet = (XSpreadsheet) UnoRuntime.queryInterface(XSpreadsheet.class,
            xSheetsIA.getByIndex( 0 ));
        XCell xCell = null;
        //get cell of column 2 row 3- (column, row)
        xCell = xSheet.getCellByPosition( 1, 2 );
        XText xText = (XText)UnoRuntime.queryInterface( 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 presentation sample plug-in and OpenOffice.org SDK Developer's Guide.

Chapter 5. Packaging and deploying your plug-ins

After you have completed plug-in development, run your code in an installed Lotus Symphony product environment, or distribute your plug-ins to customers in a Lotus Symphony environment.

This chapter illustrates how to package and deploy an application to Lotus Symphony using the sample plug-in customizing as a example, which can be found in the Lotus Symphony toolkit.

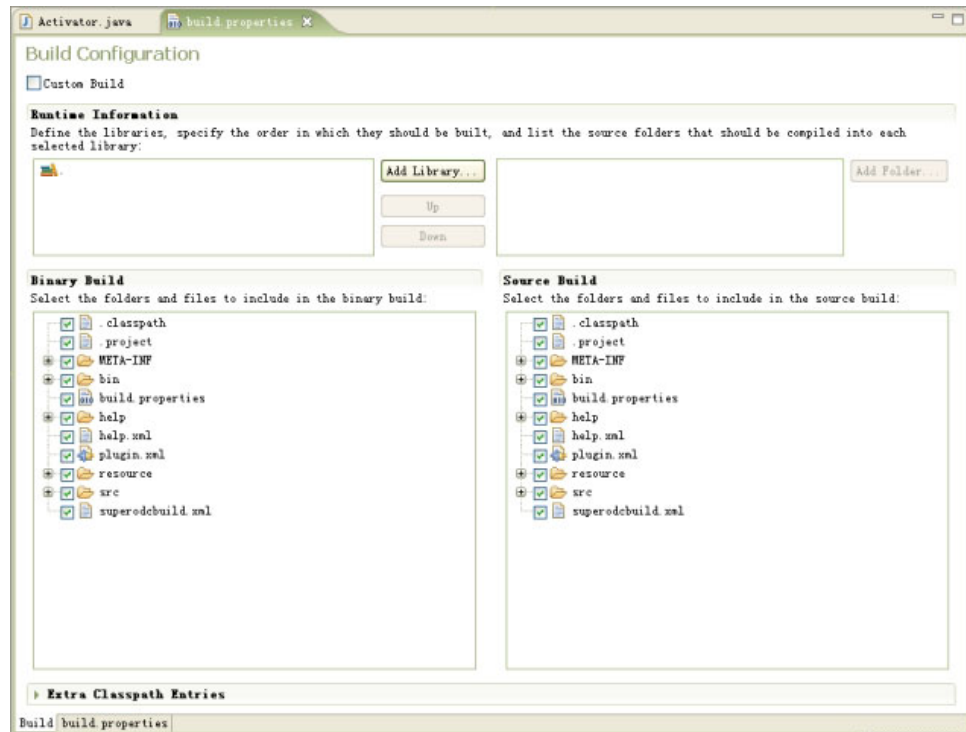
Use following steps to install your custom application into Lotus Symphony:

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

5.1 Prepare custom plug-in for deployment

The following steps prepare the plug-in for deployment:

1. Open Eclipse. Be sure to use the same workspace where you created your plug-ins.
2. Expand your plug-in in the **Package Explorer** perspective.
3. Double-click the build.properties file.
4. Select the portions of the plug-in that you want to include in the build. For the purposes of this example, all are chosen; however, this might not be necessary in your scenario.



5. Click **File > Save**.

5.2 Create a feature and an Eclipse location update site

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

5.2.1 Creating a feature

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

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

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

1. Make sure that your plug-in is opened in the workspace you created.
2. From your workspace, select **File > New > Project > Plug-in Development > Feature Project**.
3. On the **Feature Properties** page, enter the **Feature ID**, **Feature Name**, and **Feature Version**. The **Feature Provider** and **Install Handler Library** are optional.

New Feature

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

Project name:

☒ Use default location

Location:

Feature properties

Feature ID:

Feature Name:

Feature Version:

Feature Provider:

Install Handler Library:

? < Back Next > Finish Cancel

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

customizing Feature

General Information

This section describes general information about this feature.

ID:	com.ibm.productivity.tools.samples.customizing.featrue	
Version:	1.0.1	
Name:	customizing Feature	
Provider:	IBM	
Branding Plug-in:	com.ibm.productivity.tools.samples.customizing	Browse...
Update Site URL:		
Update Site Name:		

Supported Environments

Specify environment combinations in which this feature can be installed. Leave blank if the feature does not contain platform-specific code.

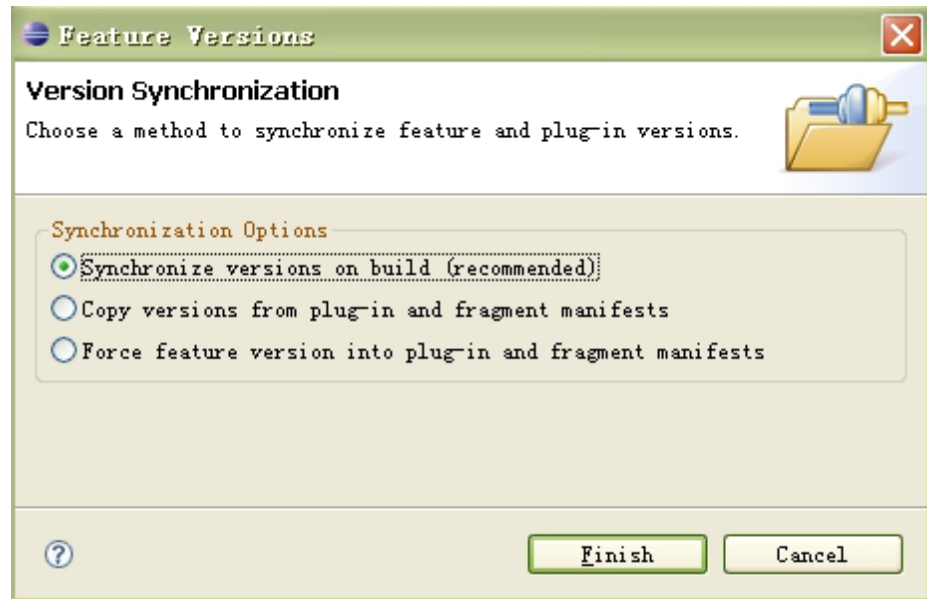
Operating Systems:		Browse...
Window Systems:		Browse...
Languages:		Browse...
Architecture:		Browse...

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

Note: This information is used to specify the site that is used to load your feature using Eclipse Update Manager. When Update Manager looks for updates, it will look for sites defined in your update site URL. If you have not created an Eclipse update site yet, you can change this setting later.

7. Click the **Information** tab.
 - a. The **Feature Information**, **Copyright**, **License** and **Sites to Visit** tabs are displayed. Feature information is displayed to the user by the Update Manager when the feature is selected.
 - b. For each of these tabs, you can either enter a URL, if sites already exist, or you can enter the information in the text area for each.
 - c. In the **Optional URL** field, enter a URL and name for any other relevant update sites that you have.
8. Click the **Plug-in** tab.
 - a. Confirm that your plug-in is listed in the Plug-ins and Fragments window. If it is not, click **Add** and select the plug-in that you want to include, and then click **OK**.
 - b. Click **Version**.

- c. Select **Synchronize Versions on Build (recommended)**, as shown in the following figure, and then click **Finish**. This step synchronizes your feature version and plug-in version.



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

5.2.2 Creating an update site

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

To create an update site, complete the following steps:

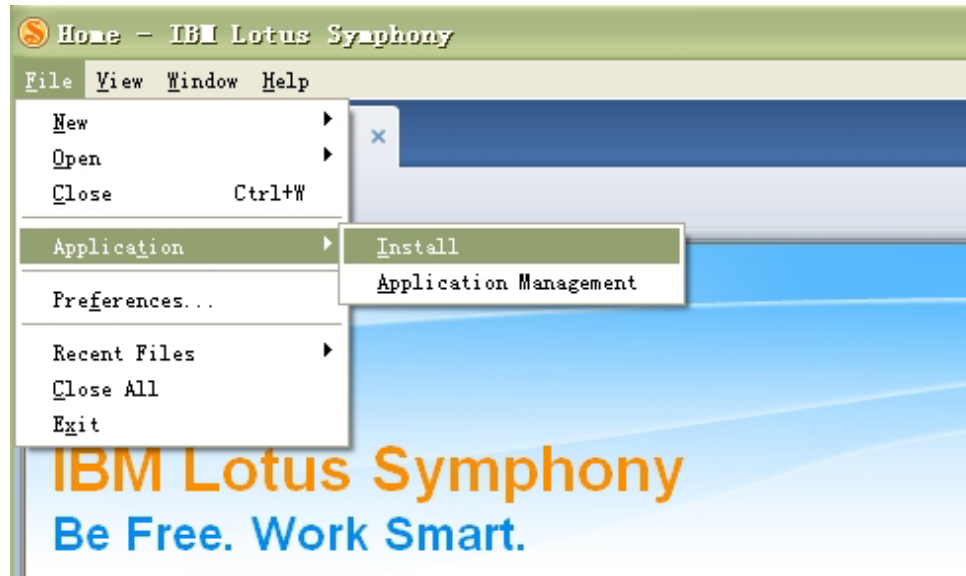
1. Open Eclipse. Be sure to open the workspace where you created your plug-in and feature.
2. Select **File > New > Project > Plug-in Development > Update Site Project**.
3. The **New Update Site** wizard has only one page:
 - a. Enter a Project name. You should enter the plug-in name and append another word to denote that it is an update site project.
 - b. Select **Use the default location**.
 - c. Click **Finish**. The wizard creates your update site within your Eclipse workspace.
4. To add your feature(s):
 - a. Double-click the `site.xml` file located in the Package Explorer frame. This step opens your site manifest editor in the editor frame (center frame).
 - b. To add your new feature, click **Add Feature**. If you are adding more than one feature or plug-in or plan to in the future, you can choose to organize them by category.
 - c. Select the feature that you are including in this update site. You can select more than one by holding down the Ctrl key. When you are finished selecting, click **OK**.

- Click the **Build All** button. This step adds the /Features and /Plug-ins directories to the Site project and populates them with JAR files containing your feature and plug-in files. This step builds your update site locally.
- Export this update site project to the file system, for example, D:\customizing.

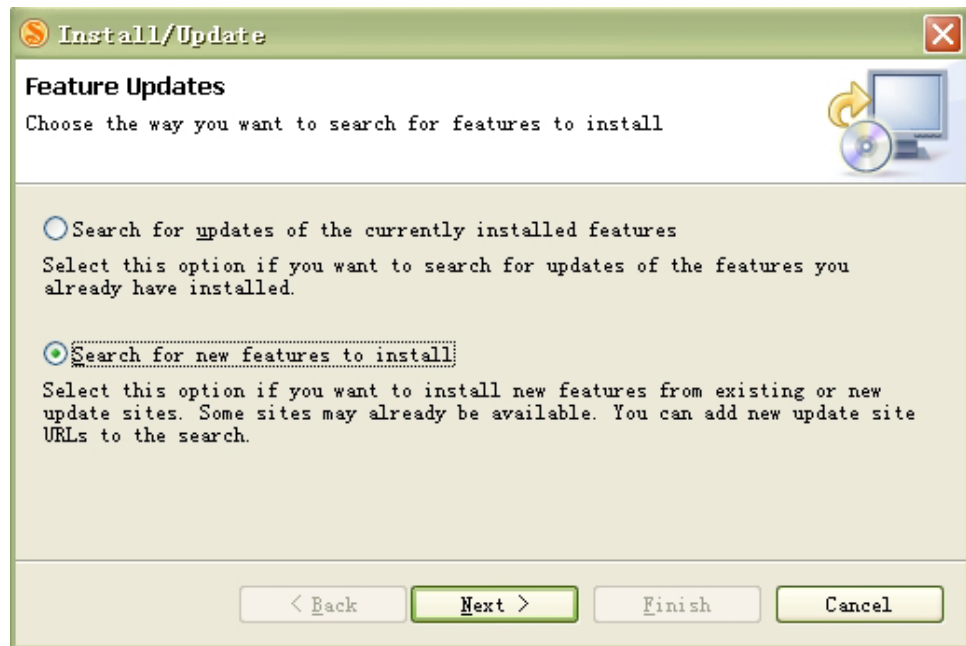
5.3 Install a custom Lotus Symphony application

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

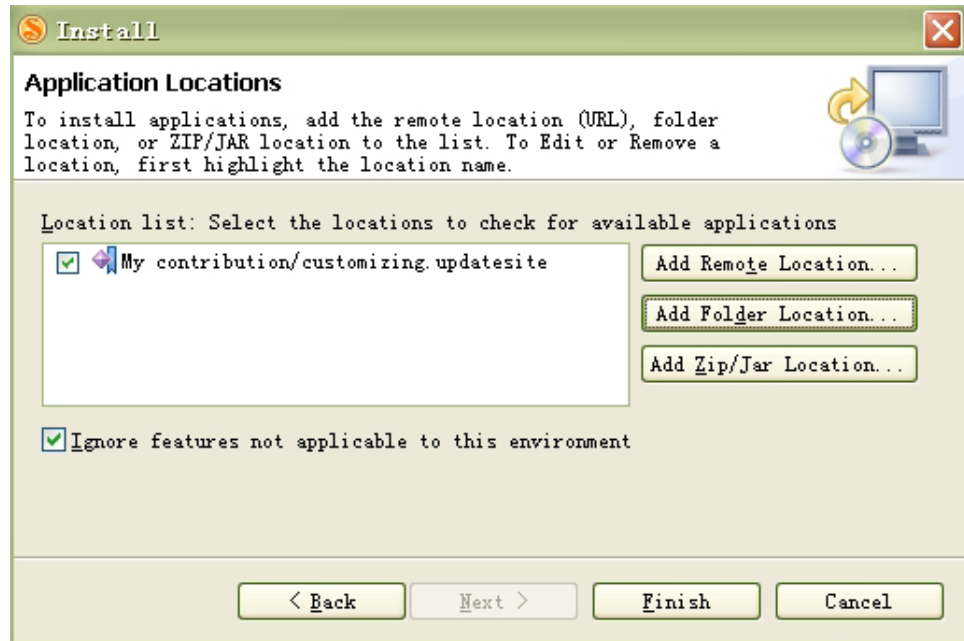
- Launch Lotus Symphony and select **File > Applications > Install**.



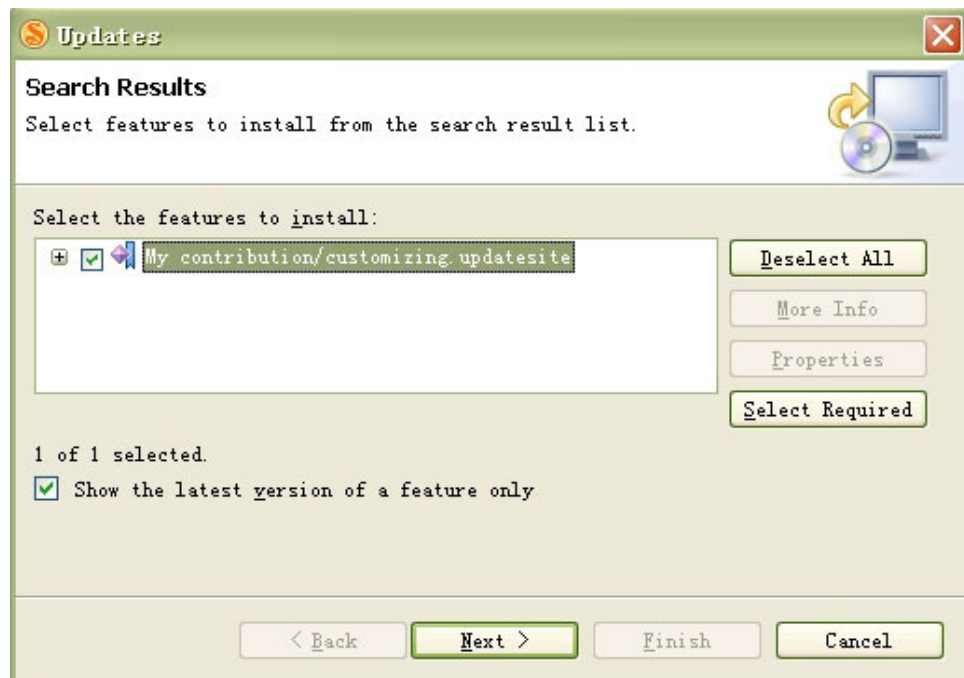
- Select **Search for new features to install** and click **Next**.



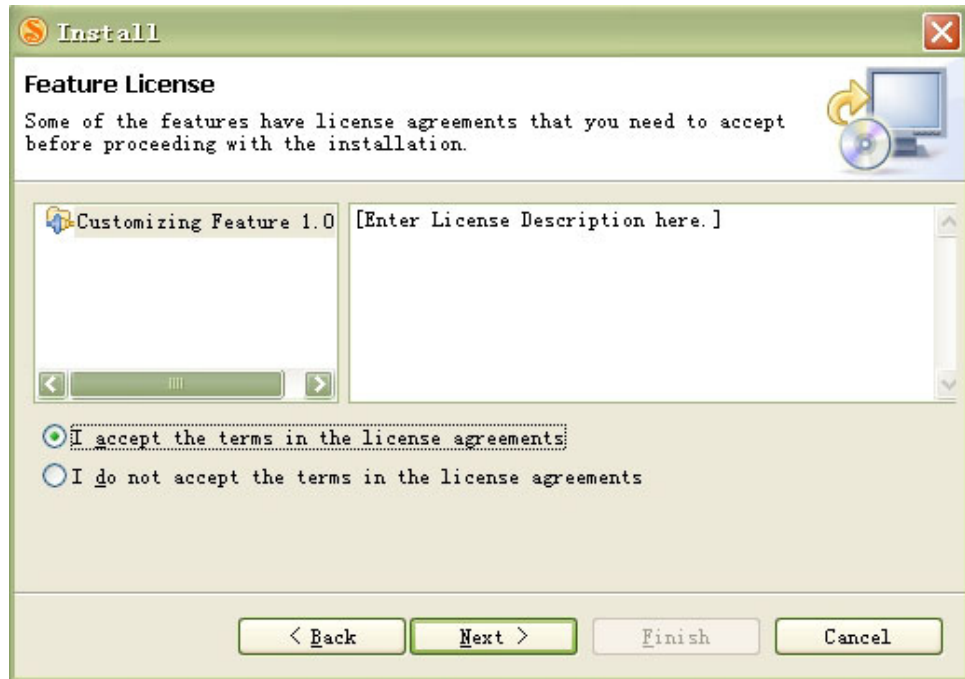
- Click **Add Folder Location** and select the update site project from the local file system, then click **Finish** button.



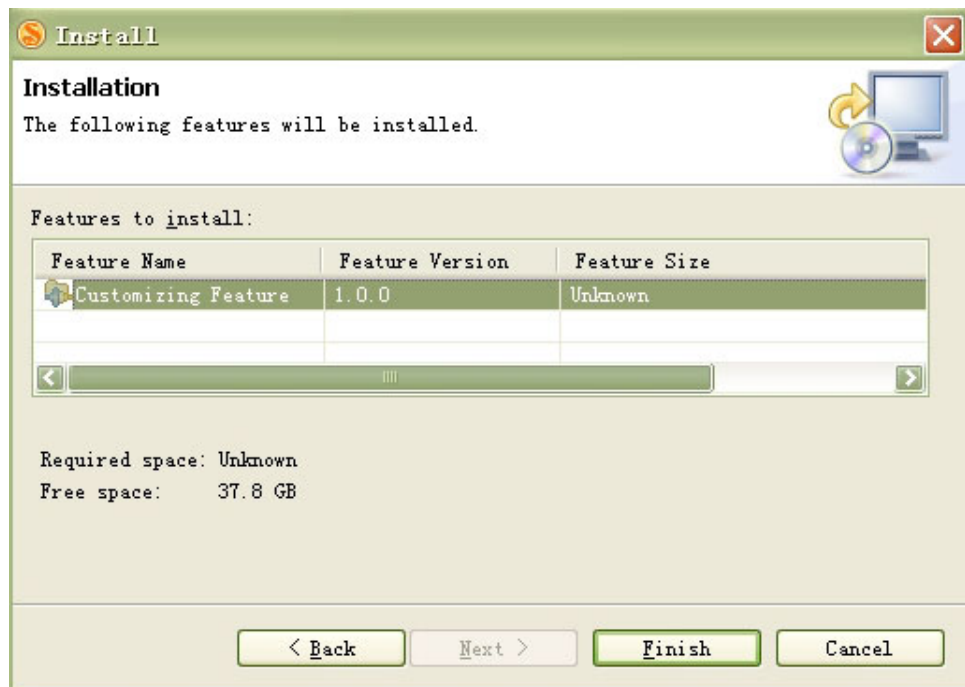
4. In Updates window, select the update site, and then click Next.



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



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



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

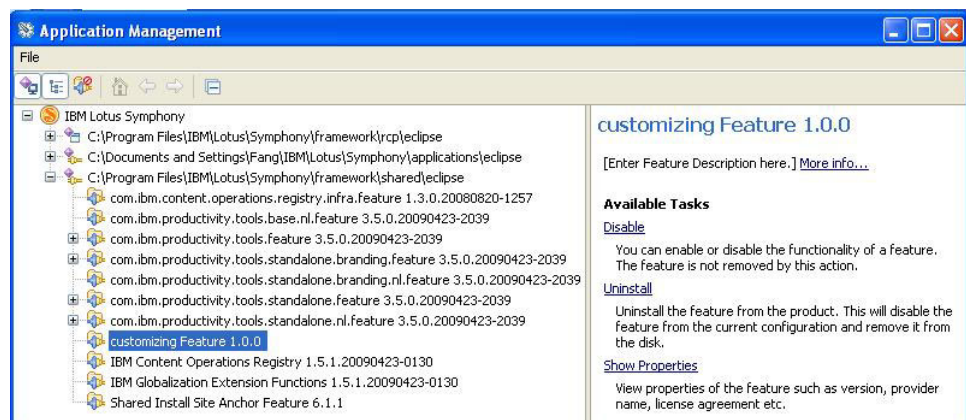
5.4 Disable or enable custom Lotus Symphony applications

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

- Click **File > Application > Application Management**.



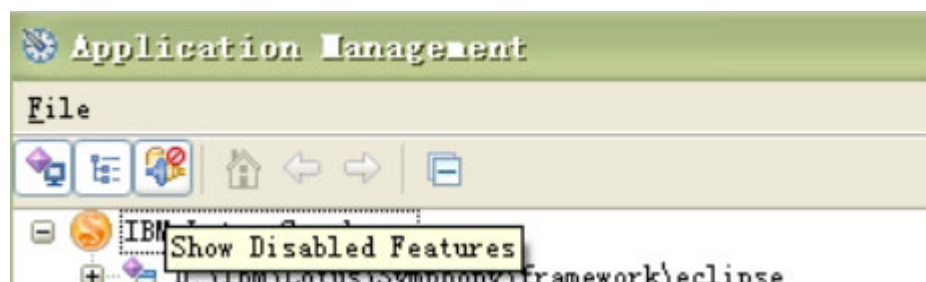
2. In the navigator, click <Symphony Install Home>\framework\shared\eclipse, and find the custom application that you want to view and make changes to.



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

To enable a disabled application, do the following steps:

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



3. Click <Symphony Install Home>\framework\shared\eclipse, to find the custom application that you want to view and make changes to.

4. Click **Enable** in right pane and accept the restart operation, to enable the selected application.

5.5 Uninstall custom Lotus Symphony application

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

Part 5. The Lotus Symphony Object Model

The Lotus Symphony Object Model provides APIs to control Lotus Symphony, such as opening documents, presentations and spreadsheets, or manipulating tables, paragraphs, sections of a document. The Lotus Symphony Object Model supports both Java and LotusScript on Windows and Linux. LotusScript support for Lotus Symphony object model is only available on Lotus Notes integrated with Lotus Symphony.

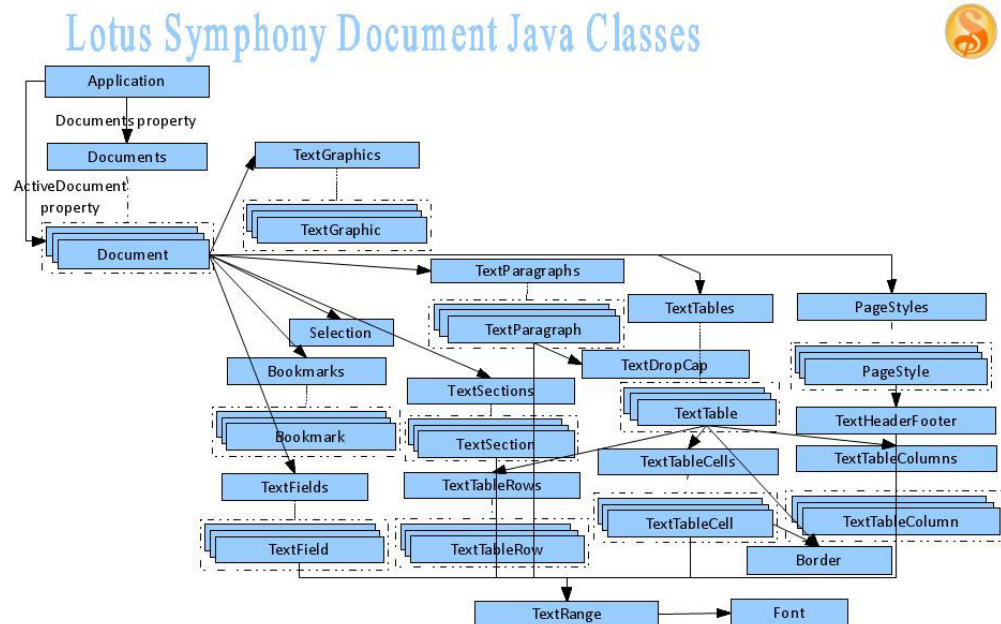
The two languages share the same models, which means that the object models in Java and LotusScript have the same properties and methods. This part only focus on the Lotus Symphony object model in Java.

To use Lotus Symphony object model APIs in Java, you must add plug-in `com.ibm.symphony.javaapi` to the dependencies of your plug-ins.

Chapter 1. Lotus Symphony Document Object Model

1.1 Overview

The Lotus Symphony Document Object Model handles the document content. It provides access to paragraphs, sections, tables, and fields. The following figure shows the structure of the Lotus Symphony Document Object Model.



1.2 Handling Lotus Symphony documents

Creating documents

Use the following method in `com.ibm.symphony.javaapi.document.Documents`:

```
public Document addDocument(String template, boolean asTemplate,
    boolean visible) throws SymphonyException
```


The parameter **template** is the name of the template to be used for the new document. When template is used, parameter **asTemplate** is used to specify whether to load the template for editing or create a new document based on the template. A value of true creates a new untitled document based on the template, and a value of false loads the template for editing. If no template is used, both true and false create a new untitled document. The parameter **visible** specifies whether to open the template or new document in a visible window or tab.

The following example creates a Symphony document base on a template visibly.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.addDocument("D:\\test.ott", true, true);
```

Accessing existing documents

The class `com.ibm.symphony.javaapi.document.Documents` provides a method to access an existing Lotus Symphony Document:

```
public Document openDocument(String fileName,
                             boolean visible) throws SymphonyException
```

The following example opens a document visibly.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("D:\\test.odt", true);
```

Saving and exporting documents

The class `com.ibm.symphony.javaapi.document.Document` provides methods to save and export documents to other format:

```
public void saveDocument()
public void exportPDF(String fileName) throws SymphonyException
public void saveAsDocument(String fileName,
                           String fileFormat) throws SymphonyException
```

Parameter **fileFormat** of method `saveAsDocument` specifies the format of the document. The following file formats are supported :

File format	File type
OpenDoc Text 1.0	.odt file
OpenDoc Text Template 1.0	.ott file
MS Word 97	.doc file
Rich Text Format	.rtf file
Text	.txt file

The following example saves a new created document.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.addDocument("", true, true);
document.saveAsDocument("C:\\work.odt", "OpenDoc Text 1.0");
```

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("D:\\test.odt", true);
document.exportPDF("C:\\work.pdf");
document.saveAsDocument("C:\\work.doc", "MS Word 97");
```

Closing Document

The class `com.ibm.symphony.javaapi.document.Document` provides this method to close the current document:

```
public void closeDocument(boolean saveChanges)
```

If the value of parameter **saveChanges** is true, the document saves changes before closing. If it is false, the document closes without saving changes.

The class `com.ibm.symphony.javaapi.document.Documents` provides a method to close all opened documents:

```
public void closeDocuments(boolean saveChanges)
```

Using documents in invisible mode

Lotus Symphony Document Object Model provides ways to control documents in invisible mode: creating or opening document invisibly. Invisible mode is useful when operating documents in the background without breaking the user's view. The following example exports an existing document to a PDF file.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("D:\\test.odt", false);
document.exportPDF("C:\\work.pdf");
```

1.3 Working with Lotus Symphony document Paragraphs

In Lotus Symphony Document Object Model, you can insert and remove paragraphs, and change the content and style of a paragraph.

Accessing paragraphs

The `item` method of `com.ibm.symphony.javaapi.document.TextParagraphs` is used to get a paragraph of a document

The following example gets the third paragraph of the document.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("C:\\Scenario\\TextParagraph.odt", true);
TextParagraph paragraph = document.getParagraphs().item(3);
```

Inserting paragraph

The `com.ibm.symphony.javaapi.document.TextRange` class provides a method to insert paragraph:

```
public TextParagraph insertParagraph(Object tableOrSection,  
    boolean before) throws SymphonyException
```

The type of parameter **tableOrSection** can be `com.ibm.symphony.javaapi.document.TextTable` or `com.ibm.symphony.javaapi.document.TextSection`, which means that you can only insert paragraphs before or after a table or a section. The following example inserts a paragraph before a table.

```
Application application = Application.getInstance();  
Documents documents = application.getDocuments();  
Document document = documents.openDocument("C:\\Scenario\\TextParagraph.odt", true);  
TextTables tables = document.getTables();  
TextTable table = tables.item(2);  
TextRange range = document.getContent();  
TextParagraph paragraph = range.insertParagraph(table, true);
```

The following example insert a paragraph after paragraph 3.

```
TextRange range = document.getParagraphs().item(3).getRange();  
range.insertBreak(range.getEnd(), Constant.SYMPHONY_TEXT_BREAK_PARAGRAPH);
```

Removing paragraph

The `remove` method of `com.ibm.symphony.javaapi.document.TextParagraph` is used to remove the paragraph.

The following example removes the second paragraph.

```
document.getParagraphs().item(2).remove();
```

Formatting paragraph

The `com.ibm.symphony.javaapi.document.TextParagraph` class has several properties to format paragraph. The following table shows these properties:

Table 1. Properties of a paragraph object

Properties	Description
AutoFirstLineIndent	Specifies whether to set the first line indentation automatically. If set to true, the first line indentation is the length of two characters, and the property <code>FirstLineIndent</code> does not take effect.
BackColor	The background color of the paragraph. This property only takes effect when set <code>BackTransparent</code> to false.
BackTransparent	Specifies whether the background is transparent.
DropCap	Represents a dropped capital letter at the beginning of a paragraph.
FirstLineIndent	The indentation for the first line (in 1/100 mm). If the value is negative, set a hanging indentation.
HoriAlignment	Constant. Specifies the horizontal alignment of the paragraph.

Table 1. Properties of a paragraph object (continued)

Properties	Description
LineSpacing	Constant. Specifies the line spacing type of a paragraph.
LineSpacingHeight	<p>Returns or sets the height in regard to the LineSpacing property. This property only takes effect when the value of property LineSpacing is</p> <p>Constant.SYMPHONY_LINESPACING_MODE_PROP or Constant.SYMPHONY_LINESPACING_MODE_FIX .</p> <ul style="list-style-type: none"> The value of the property LineSpacing is Constant.SYMPHONY_LINESPACING_MODE_PROP, the value of the property LineSpacingHeight is a proportional value of the line height, and the range of its value is 50-200. The value of the property LineSpacing is Constant.SYMPHONY_LINESPACING_MODE_FIX, the value of the property LineSpacingHeight is a fixed height, in 1/100 mm.
PageBreakBefore	Returns or sets true to allow page breaks between this and the following paragraph, or false to prevent page breaks between this and the following paragraph.
Range	Returns the range of the paragraph. It provides access to the text of the paragraph, and methods to insert text in the paragraph.

The following example sets a red background color for the second paragraph, and inserts some text at the beginning of the paragraph.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("C:\\Scenario\\TextParagraph.odt", true);
TextParagraph paragraph = document.getParagraphs().item(1);
paragraph.setAutoFirstLineIndent(true);
paragraph.setBackTransparent(false);
//set background color to red
paragraph.setBackgroundColor(application.RGB(255, 0, 0));
paragraph.setHoriAlignment(Constant.SYMPHONY_TEXT_HORI_ALIGN_LEFT);
//insert text at the beginning
paragraph.getRange().insertBefore("The beginning");
```

Tables

Inserting tables

The com.ibm.symphony.javaapi.document.TextTables class provides access to tables, and methods to insert and manipulate tables.

```
public TextTable item(int index) throws SymphonyException
public TextTable add(TextRange position ,int numOfRow,
                    int numOfColumn) throws SymphonyException
```

The following example add a table with five rows and five columns before the forth paragraph.

```
TextRange pos = document.getParagraphs().item(4).getRange().getStart();
TextTables textTables = document.getTextTables();
TextTable newTextTable = textTables.add(pos, 5, 5);
newTextTable.setName("table1");
```

Accessing table objects

The `com.ibm.symphony.javaapi.document.TextTable` class provides access to table rows, columns, and cells, and a method to remove tables.

```
public TextTableCells getCells()
public TextTableColumns getColumns()
public TextTableRows getRows()
public void remove()
```

The following example removes the third table.

```
TextTables textTables = document.getTables();
TextTable table = textTables.item(3);
table.remove();
```

Inserting and removing table rows

The `com.ibm.symphony.javaapi.document.TextTableRows` class provides methods to access, insert and remove table rows:

```
public TextTableRow add(int rowIndex, int count) throws SymphonyException
public TextTableRow item(int index) throws SymphonyException
public void remove(int beginIndex, int count) throws SymphonyException
```

The parameter **rowIndex** of the `add` method is the index of the row before which the new rows are inserted. The parameter **count** is the number of rows to be added. The parameter **beginIndex** of the `remove` method is the index of the first row to be removed. The parameter **count** is the number of rows to be removed. The following example shows how to add and remove table rows.

```
TextTables textTables = document.getTables();
TextTable textTable = textTables.item(1);
TextTableRows rows = textTable.getRows();
TextTableRow row = rows.add(1, 1);
System.out.println(rows.getCount());
rows.remove(1, 2);
System.out.println(rows.getCount());
```

Inserting and removing table columns

The `com.ibm.symphony.javaapi.document.TextTableColumns` class provides methods to access, insert, and remove table columns:

```
public TextTableColumn add(int index, int count) throws SymphonyException
public TextTableColumn item(int index) throws SymphonyException
public void remove(int beginIdx, int count) throws SymphonyException
```

The following example shows how to add and remove table columns.

```
TextTables textTables = document.getTables();
TextTable textTable = textTables.item(1);
TextTableColumns columns = textTable.getColumns();
//add two column before the first column
TextTableColumn column = columns.add(1, 2);
System.out.println(columns.getCount());
columns.remove(1, 2);
System.out.println(columns.getCount());
```

Accessing table cells

The `com.ibm.symphony.javaapi.document.TextTable` class provides methods to access table cells:

```
public TextTableCell cell(int rowIndex, int columnIndex) throws SymphonyException
public TextTableCell cell(String name) throws SymphonyException
```

The following example gets a cell by row index and column index.

```
TextTables textTables = document.getTables();
TextTable textTable = textTables.item(1);
TextTableCell cell = textTable.cell(2, 1);
```

The Text, Value, and Formula properties of cell

The value of a cell can be a string value or a double value. The property `Text` is used to set the string value for a cell. The property `Value` is used to set the double value for a cell. The value for the `Text` property and `Value` property can overwrite each other's value.

- When you set the value for the `Text` property, the value of the `Value` property is the numeric conversion value of the `Text` property's value.
- When you set the value for the `Value` property, the value of the `Text` property is the string conversion value of the `Value` property's value.

For example, if we set the `Text` property's value as `Test`, the `Value` property's value is 0; if we set the `Value` property's value as `26.7`, the `Text` property's value is `26.7`.

The property `Formula` is used to calculate the cell value automatically by a formula. The following example shows how to set a formula for a cell.

```
TextTable table = textTables.add(document.getContent().getEnd(), 7, 5);
table.cell(1, 2).setValue(2);
table.cell(1, 3).setValue(3);
table.cell(1, 4).setFormula("=<B1>*<C1>");
table.cell(2, 2).setValue(3);
table.cell(2, 3).setValue(4);
table.cell(2, 4).setFormula("=<B2>*<C2>");
table.cell(3, 2).setFormula("=Sum<B1:B2>");
```

Sections

The `com.ibm.symphony.javaapi.document.TextSections` class provides access to sections, and a method to insert sections:

```

public TextSection add(String name, TextRange position,
    int count) throws SymphonyException
public TextSection item(int index)

```

The parameter **position** is the position before which the new section is inserted. The parameter **count** specifies the number of columns of the new section.

The `com.ibm.symphony.javaapi.document.TextSection` class provides access to the text range of the section, sets link for a section, and a method to remove section:

```

public String getLink()
public void setLink(String link)
public void remove()
public TextRange getRange()

```

The following example adds a new section, and sets the second section as the new section's link section.

```

Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("C:\\Scenario\\TextSection.odt", true);
TextSections sections = document.getSections();
TextSection section = sections.item(1);
TextSection newSection = sections.add("New section",
    document.getParagraphs().item(1).getRange(), 1);
newSection.setLink(section.getName());
newSection.setLeftIndent(500);

```

Field

The Lotus Symphony Document Object Model supports adding and removing user and page number fields for documents.

The class `com.ibm.symphony.javaapi.document.TextFields` provides access to fields and a method to add a field:

```

public TextField item( int index ) throws SymphonyException
public TextField add( TextRange position, String type,
    String Name ) throws SymphonyException

```

The parameter **position** of the add method is the position of the new field. The parameter **type** is the type of the field, of value `User` or `PageNumber`.

The class `com.ibm.symphony.javaapi.document.TextField` provides ways to remove field, set name, content or value for the user field:

```

public String getContent()
public void setContent(String content)
public String getName()
public void setName(String name)
public double getValue()
public void setValue(double value)
public void remove()

```

The value of a user field can be a string value or a double value. The property `Content` is used to set the string value for a cell. The property `Value` is used to set

the double value for a user field. The value for the Content property and Value property can overwrite each other's value.

- When you set the value for the Content property, the value of the Value property is the numeric conversion value of the Content property's value.
- When you set the value for the Value property, the value of the Content property is the string conversion value of the Value property's value.

For example, if we set the Content property's value as Test, the Value property's value is 0; if we set the Value property's value as 26.7, the Content property's value is 26.7.

The following example adds a user field at the beginning of the fourth paragraph.

```
Application application = Application.getInstance();
Documents documents = application.getDocuments();
Document document = documents.openDocument("C:\\Scenario\\TextSection.odt", true);
TextFields fields = document.getFields();
TextRange range = document.getParagraphs().item(4).getRange().getStart();
TextUserField field = fields.add(range, "User", "Vendor Name");
field.setContent("Peter");
```

Range

The `com.ibm.symphony.javaapi.document.TextRange` object is contained by all the objects with text as its content, such as document, paragraph and section. It provides access to the text, font, start, and end of range, and methods to insert text at the beginning or end of a range, and a method to insert breaks and paragraphs.

Accessing the range start and end

Access the start and end of the range using the properties `Start` and `End`. The start and end position of a range is the beginning or the end of the range of which text length is always 0.

Inserting text, breaks and paragraphs

`com.ibm.symphony.javaapi.document.TextRange` provides the following methods to insert text, breaks and paragraphs:

```
public void insertBefore(String text)
public void insertAfter(String text)
public void insertBreak(TextRange position, int type) throws SymphonyException
public TextParagraph insertParagraph(Object tableOrSection,
    boolean before) throws SymphonyException
```

The following example inserts a line break at the end of the third paragraph.

```
TextRange range = document.getParagraphs().item(3).getRange();
range.insertBreak(range.getEnd(), Constant.SYMPHONY_TEXT_BREAK_LINE);
```

The following example inserts a paragraph before the first table, and inserts text for the paragraph.

```
TextTables tables = document.getTables();
TextTable table = tables.item(1);
TextParagraph paragraph = document.getContent().insertParagraph(table, true);
paragraph.getRange().insertBefore("my paragraph");
```

Importing file

Use the method `importFile` in class `TextRange` to import a doc, txt, or odt file into a paragraph or section.

The following example imports a txt file into a paragraph.

```
TextRange range = document.getParagraphs().item(2).getRange();
range.importFile("c:\\import.txt");
```

Selection

The class `com.ibm.symphony.javaapi.document.Selection` represents the current selection of a document. It provides methods to select range, copy, cut the selected range to a clipboard, and paste data to the selected range.

```
public void setRange( TextRange startRange, TextRange endRange )
public void cut()
public void copy()
public void paste()
```

Use the following and methods in `SymphonyDocumentSelection`:

```
SetRange(StartRange, EndRange)
Copy()
Cut()
Paste()
```

The parameter **startRange** of the method `setRange` is the start position of the selection. The parameter **endRange** is the end position of the selection. The following example replaces the content of second paragraph with the content of first paragraph.

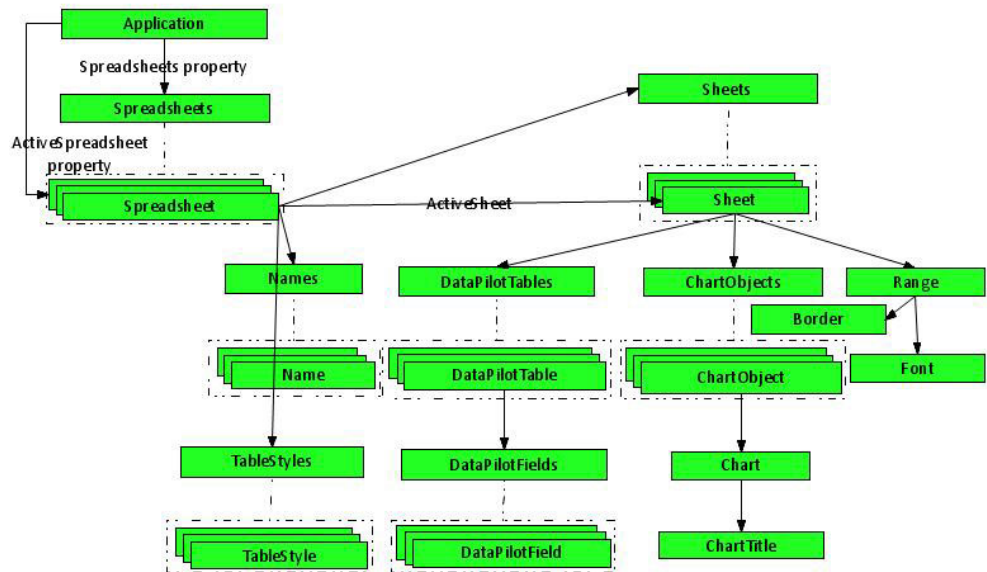
```
Selection selection = document.getSelection();
TextRange range = document.getParagraphs().item(1).getRange();
selection.setRange(range.getStart(), range.getEnd());
selection.copy();
TextRange range2 = document.getParagraphs().item(2).getRange();
selection.setRange(range2.getStart(), range2.getEnd());
selection.paste();
```

Chapter 2. Lotus Symphony Spreadsheet Object Model

2.1 Overview

Lotus Symphony Spreadsheet Object Model handles spreadsheet documents. It provides access to spreadsheets, sheets, DataPilot tables, chart objects, and ranges. The following figure shows the structure of the Lotus Symphony Spreadsheet Object Model.

Lotus Symphony Spreadsheet Java Classes



2.2 Handling Lotus Symphony spreadsheets

Creating spreadsheets

To create a Lotus Symphony spreadsheet, use the following method in the class `com.ibm.symphony.javaapi.spreadsheet.Spreadsheets`:

```
public Spreadsheet addSpreadsheet(String template, boolean newTemplate,  
    boolean visible) throws SymphonyException
```

The parameter **template** is the name of the template to be used for the new spreadsheet. When **template** is used, the parameter **newTemplate** is used to specify whether to load the template for editing or create a new spreadsheet based on the template. A value of `true` creates a new untitled spreadsheet based on the template, and a value of `false` to load the template for editing. If no template is used, both `true` and `false` create a new untitled spreadsheet. The parameter **visible** specifies whether to open the template or new spreadsheet in a visible window or tab.

The following example creates a Lotus Symphony spreadsheet base on a template in visible mode.

```
Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();  
Spreadsheet spreadsheet = spreadsheets.addSpreadsheet("D:\\test.ots", true, true);
```

Accessing existing spreadsheets

The class `com.ibm.symphony.javaapi.spreadsheet.Spreadsheets` provides method to access an existing Lotus Symphony spreadsheet:

```
public Spreadsheet openSpreadsheet(String fileName, boolean visible)
    throws SymphonyException
```

The following example opens a spreadsheet in visible mode.

```
Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();
Spreadsheet spreadsheet = spreadsheets.
    openSpreadsheet("D:\\test.ods", true);
```

Saving and exporting spreadsheets

The class `com.ibm.symphony.javaapi.spreadsheet.Spreadsheet` provides methods to save and export a Lotus Symphony spreadsheet to other formats:

```
public void saveSpreadsheet()
public void saveAsSpreadsheet(String fileName, String fileFormat)
    throws SymphonyException
public void exportPDF(String fileName) throws SymphonyException
```

The parameter **fileFormat** of the method `saveAsSpreadsheet` specifies the format of the spreadsheet to save. The following file formats are supported:

File Format	File Type
OpenDoc SpreadSheet 1.0	.ods file
OpenDoc SpreadSheet Template 1.0	.ots file
MS Excel 97	.xls file
Text - txt - csv (StarCalc)	.csv file

The following example saves a new spreadsheet.

```
Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();
Spreadsheet spreadsheet = spreadsheets.addSpreadsheet("", true, true);
spreadsheet.saveAsSpreadsheet("c:\\work.ods", "OpenDoc SpreadSheet 1.0");
```

```
Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();
Spreadsheet spreadsheet = spreadsheets.openSpreadsheet("D:\\test.ods", true);
spreadsheet.exportPDF("C:\\work.pdf");
spreadsheet.saveAsSpreadsheet("C:\\work.xls", "MS Excel 97");
```

Closing spreadsheets

The class `com.ibm.symphony.javaapi.spreadsheet.Spreadsheet` provides a method to close the current spreadsheet:

```
public void closeSpreadsheet(boolean saveChange)
```

If the value of parameter **saveChange** is true, the spreadsheet saves changes before closing. If it is false, the spreadsheet closes without saving changes.

The class `com.ibm.symphony.javaapi.spreadsheet.Spreadsheets` provides a method to close all opened spreadsheets:

```
public void closeSpreadsheets(boolean saveChange)
```

Using spreadsheet in invisible mode

Lotus Symphony Spreadsheet Object Model provides ways to use spreadsheets in invisible mode: creating or opening spreadsheets invisibly. The following example exports a spreadsheet to a PDF file.

```
Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();  
Spreadsheet spreadsheet = spreadsheets.openSpreadsheet("D:\\test.ods", false);  
spreadsheet.exportPDF("C:\\work.pdf");
```

2.3 Working with Lotus Symphony spreadsheets Sheets

Accessing sheets

To access the sheets of a spreadsheet, use the item method in class `com.ibm.symphony.javaapi.spreadsheet.Sheets`. You can also use property `Previous` and `Next` in the class `com.ibm.symphony.javaapi.spreadsheet.Sheet` to access the previous and next sheet of current sheet.

```
public Sheet previous()  
public Sheet next()
```

Insert, copy and move sheets

The class `com.ibm.symphony.javaapi.spreadsheet.Sheets` provides a method to insert new sheets:

```
public Sheet addTo(Sheet sheet, int count, boolean beforeOrAfter)  
    throws SymphonyException
```

The parameter **sheet** specifies the position before or after which the new sheets are inserted. The parameter **count** specifies the number of sheets to be added. The parameter **beforeOrAfter** specifies whether to insert the new sheets before or after the specified sheet.

The class `com.ibm.symphony.javaapi.spreadsheet.Sheet` provides methods to copy and move sheets:

```
public void copyTo(Sheet pre, boolean beforeOrAfter)  
    throws SymphonyException  
public void moveTo(Sheet sheet, boolean beforeOrAfter)  
    throws SymphonyException
```

The following example inserts a sheet before the first sheet, then copies the new sheet after the second sheet, and then moves it after the third sheet.

```

Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();
Spreadsheet spreadsheet = spreadsheets.openSpreadsheet("D:\\test.ods", true);
Sheets sheets = spreadsheet.getSheets();
sheets.addTo(sheets.item(1), 1, true);
Sheet sheet = sheets.item(1);
sheet.setName("new sheet");
sheet.copyTo(sheets.item(2), false);
sheet.moveTo(sheets.item(3), false);

```

Removing sheets

To remove an existing sheet, use the remove method in the class `com.ibm.symphony.javaapi.spreadsheet.Sheet`.

Protect sheet

The method `protect` and `unProtect` in the class `Sheet` locks and unlocks the sheet from UI. The property `Protected` returns whether the sheet is locked or not.

```

public void protect(String password)
public void unProtect(String password)
public boolean isProtected()

```

The following example locks the active sheet.

```

spreadsheet.getActiveSheet().protect("1234");
if(spreadsheet.getActiveSheet().isProtected()){
    System.out.println("This sheet can not be modified");
}

```

Importing files

The Lotus Symphony Spreadsheet Object Model supports importing data from csv files or spreadsheet files.

To import data from a csv file, use the method `importCSVFile` in `com.ibm.symphony.javaapi.spreadsheet.Sheet`. To import data from a spreadsheet file, use the method `importFile`.

```

public void importCSVFile(String FileName, String fieldSeparator,
    String Delimiter, int firstLine, int importMode) throws SymphonyException
public void importFile(String fileName, String SheetName,
    int firstLine, int importMode) throws SymphonyException

```

The following example imports data from a csv file into the first sheet, and imports data from a spreadsheet into the second sheet.

```

Spreadsheets spreadsheets = Application.getInstance().getSpreadsheets();
Spreadsheet spreadsheet = spreadsheets.openSpreadsheet("C:\\work.ods", true);
Sheet sheet = spreadsheet.getSheets().item(1);
sheet.importCSVFile("C:\\Scenario\\CsvFile.csv", ",", "", 1,
    Constant.SYMPHONY_SHEET_IMPORTMODE_NORMAL);
sheet = spreadsheet.getSheets().item(2);
sheet.importFile("C:\\Scenario\\Sheet.ods", "A", 3,
    Constant.SYMPHONY_SHEET_IMPORTMODE_NORMAL);

```

Accessing cells

Use the class `com.ibm.symphony.javaapi.spreadsheet.Range` to access cells of a sheet. A `Range` object represents a range that contains one cell or a collection of cells in a sheet.

The class `com.ibm.symphony.javaapi.spreadsheet.Sheet` provides methods to get range objects:

```
public Range cells(int rowIndex, int columnIndex) throws SymphonyException
public Range columns(int index) throws SymphonyException
public Range rows(int index) throws SymphonyException
public Range range(Range range1, Range range2) throws SymphonyException
public Range range(String name)
```

The method `cells` is used to access a cell by a column index and a row index. The method `columns` and `rows` are used to access the cells of a row and the cells of a column.

The method `range(String name)` is used to access cells by name. The parameter **name** can be a name of cell, for example, "C5", or the name of a range of cells, for example, "A1:D5".

The method `range(Range range1, Range range2)` is used to access cells between the top-left of range **range1** and top-left of range **range2**.

The following example shows the ways to access cells.

```
//get one cell by row index and column index
Range oneCell = sheet.cells(5, 5);
//get all cells of column 5
Range oneColumn = sheet.columns(5);
//get all cells of row 5
Range oneRow = sheet.rows(5);
//get one cell by name
oneCell = sheet.range("B15");
//get a range of cells by name
Range range = sheet.range("A2:E20");
Range range1 = sheet.range("A2:E10");
Range range2 = sheet.range("B6:F20");
//get cells between top-left of range1 and top-left of range2
range = sheet.range(range1, range2);
```

Range

The class `com.ibm.symphony.javaapi.spreadsheet.Range` is used to handle a cell or a range of cells. It provides ways to set values for cells, format cells, and access the copy-cut-paste function.

Text, Value and Formula

The value of a cell can be a string value or a double value. The property `Text` is used to set the string value for a cell. The property `Value` is used to set the double value for a cell. The value for the `Text` property and `Value` property can overwrite each other's value.

- When you set the value for the `Text` property, the value of the `Value` property is the numeric conversion value of the `Text` property's value.

- When you set the value for the Value property, the value of the Text property is the string conversion value of the Value property's value.

For example, if we set the Text property's value as Test, the Value property's value is 0; if we set the Value property's value as 26.7, the Text property's value is 26.7.

The property Formula is used to calculate values automatically by a formula. The following example shows how to set formula for a cell.

```
Range range = sheet.range("B7");
range.setValue(25.4);
range = sheet.range("C7");
range.setValue(32.4);
range = sheet.range("D7");
range.setFormula("=B7+C7");
System.out.println(range.getValue());
```

Formating range

A range object has several properties to format cells, as the following table shows:

Table 2. Properties of a range object

Properties	Description
BackColor	The background color of the range. The value for this property is the decimal value of a color. Get the decimal value using the RGB method in the class Application.
ColumnWidth	The column width in 1/100th mm of the range. If the return value is 65537, the width of the columns are different from each other.
RowHeight	The row width (in 1/100th mm) of the range. If return 65537, the height of rows are different from each other.
Font	The font of the range.
WrapText	Specifies whether to wraps text in the range.

The Range class provides two methods to format range:

```
public void autoFit()
public void autoFormat( String tableStyleName )
```

The method autoFit is used to set a reasonable size for each cell of the range automatically. The method autoFormat is used to apply a table style for the range. A table style is a style format defined in Lotus Symphony previously. Use item method of the class com.ibm.symphony.javaapi.spreadsheet.TableStyles to access table styles.

The following example automatically format a range.

```
Range range = sheet.range("A1:E10");
range.setText("Hello World");
range.autoFormat(spreadsheet.getTableStyles().item(2).getName());
range.autoFit();
```

Copy, cut , paste and replace operations

The class com.ibm.symphony.javaapi.spreadsheet.Range provides methods to copy, cut, paste, and replace data.

```
public void cut()
public void copy()
public void paste()
public int replace( String what, String replacement, boolean matchcase )
```

The following example copies a range and pastes its data to another range.

```
Range range = sheet.range("A101:B102");
range.setText("copy");
range.copy();
Range dst = sheet.range("C101:D102");
dst.paste();
```

In the following example, the word "forReplace" is changed to "replaced" matching case.

```
Range range = sheet.range("A1:B2");
range.setText("forreplace");
range = sheet.range("C1:D2");
range.setText("forReplace");
range = sheet.range("E1:F2");
range.setText("ForReplace");
range = sheet.range("A1:F2");
range.replace("forReplace","replaced",true);
```

Datapilot table

To create a DataPilot table, use the add method of the class `com.ibm.symphony.javaapi.spreadsheet.DataPilotTables`.

```
public DataPilotTable add(Range source, String name,
                          Range destination) throws SymphonyException
```

The parameter **source** is the source range that contains the data. The parameter **destination** is the position to display the table.

After creating a DataPilot table, use the `addFields` method of the class `com.ibm.symphony.javaapi.spreadsheet.DataPilotTable` to add fields for the DataPilot table:

```
public void addFields(String[] rowfields, String[] colfields, String[] datafields,
                     String[] pagefields,int function) throws SymphonyException
```

- **rowfields**, specifies that the data in the fields is added as rows.
- **colfields**, specifies that the data in the fields is added as columns.
- **datafields**, specifies that the data in the fields is used to calculate the table's data area.
- **pagefields**, specifies that the data in the fields is added in the table's page area, where single values from the fields can be selected.

The following example creates a DataPilot table.

```
String[] rowFieldsString = { "Requested By" };
String[] colFieldsString = { "Item" };
String[] dataFieldsString = { "Total Price" };
String[] pageFieldsString = { "P.O. #" };
Range src = sheet.range("A1:E15");
Range dst = sheet.cells(13, 3);
DataPilotTable table = sheet.getDataPilotTables().add(src, "DPT_NEW",dst);
table.addFields(rowFieldsString, colFieldsString, dataFieldsString,
    pageFieldsString, Constant.SYMPHONY_GENERAL_FUNCTION_SUM);
```

Chart

The class `com.ibm.symphony.javaapi.spreadsheet.ChartObject` handles the embedded chart object. To add an embedded chart object, use the following method in class `com.ibm.symphony.javaapi.spreadsheet.ChartObjects`.

```
public ChartObject add(String name,int left, int top, int width, int height,
    boolean rowHeader, boolean columnHeader) throws SymphonyException
public ChartObject add(String name,int left, int top, int width,
    int height) throws SymphonyException
```

The `chartWizard` method in the class `com.ibm.symphony.javaapi.spreadsheet.Chart` creates the chart on the embedded chart object.

```
public void chartWizard(Range range, String gallery, int plotBy,
    String title, boolean hasLegend) throws SymphonyException
```

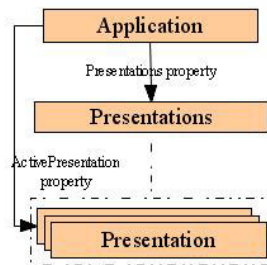
The parameter **range** is the source range of the char. The parameter **gallery** is the chart type. The following example creates a chart for the range (A1:E15).

```
Range range = sheet.range("A1:E15");
ChartObject chartObject = sheet.getChartObjects().add("Total Report", 6000,
    8000, 8000, 6000);
chartObject.getChart().chartWizard(range, Constant.SYMPHONY_CHART_PIEDIAGRAM,
    Constant.SYMPHONY_CHART_PLOTBY_COLUMNS, "total", false);
```

Chapter 3. Lotus Symphony Presentation object model

3.1 Overview

Lotus Symphony Presentation Object Model provides APIs to open, save, close, and export presentations to other formats. The following figure shows the structure of the Lotus Symphony Presentation Object Model.



3.2 Handling Lotus Symphony presentations

Creating presentations

The class `com.ibm.symphony.javaapi.presentation.Presentations` provides a method to create a presentation:

```
public Presentation addPresentation(String template, boolean newTemplate,
    boolean visible) throws SymphonyException
```

The parameter **template** is the name of the template to be used for the new presentation. When **template** is used, the parameter **newTemplate** is used to specify whether to load the template for editing or create a new presentation based on the template. A value of `true` creates a new untitled presentation based on the template, and a value of `false` loads the template for editing. If no template is used, both `true` and `false` create a new untitled presentation. The parameter **visible** specifies whether to open the template or new presentation in a window or tab in visible mode.

The following example creates a presentation base on a template in visible mode.

```
Application application = Application.getInstance();
Presentations presentations = application.getPresentations();
Presentation presentation = presentations.addPresentation(
    "D:\\test.otp", true, true);
```

Accessing existing presentations

The class `com.ibm.symphony.javaapi.presentation.Presentations` provides a method to access a presentation:

```
public Presentation openPresentation(String fileName,  
    boolean visible) throws SymphonyException
```

The following example opens a presentation in visible mode.

```
Application application = Application.getInstance();  
Presentations presentations = application.getPresentations();  
Presentation presentation = presentations.openPresentation("D:\\test.odp", true);
```

Saving and exporting presentations

The class `com.ibm.symphony.javaapi.presentation.Presentations` provides methods to save and export presentations to other formats:

```
public void savePresentation()  
public void exportPDF(String fileName) throws SymphonyException  
public void saveAsPresentation(String fileName,  
    String fileFormat) throws SymphonyException
```

The parameter **fileFormat** of the method `saveAsPresentation` specifies the format of the presentation when it is saved. The following file formats are supported:

File Format	File Type
OpenDoc Presentation 1.0	.odp file
OpenDoc Presentation Template 1.0	.otp file
MS PowerPoint 97	.ppt file

The following example saves a new presentation.

```
Application application = Application.getInstance();  
Presentations presentations = application.getPresentations();  
Presentation presentation = presentations.addPresentation("", true, true);  
presentation.saveAsPresentation("C:\\work.odp", "OpenDoc Presentation 1.0");
```

```
Application application = Application.getInstance();  
Presentations presentations = application.getPresentations();  
Presentation presentation = presentations.openPresentation("D:\\test.odp", true);  
presentation.exportPDF("C:\\work.pdf");  
presentation.saveAsPresentation("C:\\work.ppt", "MS PowerPoint 97");
```

Closing presentations

The class `com.ibm.symphony.javaapi.presentation.Presentation` provides a method to close the current presentation:

```
public void closePresentation(boolean saveChange)
```

If the value of parameter **saveChange** is true, the presentation saves changes before closing. If it is false, the presentation closes without saving changes.

The class `com.ibm.symphony.javaapi.presentation.Presentations` provides a method to close all opened presentations:

```
public void closePresentations(boolean saveChange)
```

Using Presentations in invisible mode

Lotus Symphony Presentation Object Model provides ways to use presentations in invisible mode: creating or opening presentation invisibly. The following example exports a presentation to a PDF file invisibly.

```
Application application = Application.getInstance();  
Presentations presentations = application.getPresentations();  
Presentation presentation = presentations.openPresentation("D:\\test.odp", false);  
presentation.exportPDF("C:\\work.pdf");
```

Part 6. Lotus Expeditor and UNO Programming

Chapter 1. Developing Lotus Expeditor applications

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

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

1.1. Lotus Expeditor toolkit documentation

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

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

1.2. Debugging and testing applications

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

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

1.3. Packaging and deployment for local testing

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

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

1.4. Securing applications and data

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

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

Chapter 2. UNO Programming

2.1 Getting the global service factory

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

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

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

To get the `ServiceManager`, use the following sample code:

```

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

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

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

        // Query the initial object for its main factory interface

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

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

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

        // Get the default context from the editor service.

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

        return (XMultiServiceFactory) UnoRuntime.queryInterface(
            XMultiServiceFactory.class, context.getServiceManager());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

```

2.2 Using the import and export functions

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

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

Loading new or existing components

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

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

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

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

Storing documents

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

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

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

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

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

Exporting documents and drawing objects

Writer documents and Spreadsheet documents can be exported as HTML format files. Presentation documents can export drawing objects as graphics through the `com.sun.star.drawing.GraphicExportFilter` interface. After getting a `GraphicExportFilter` from the `ServiceManager`, use its `XExporter` interface to inform the filter which page, shape, or shape collection to export.

Functions in this interface include:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        //some graphics, for example, the Windows Metafile does not have a
        //Media Type, for this case aProps[0].Name = "FilterName";
        //it is possible to set a FilterName aProps[0].Value = "WMF";
        java.io.File destFile = new java.io.File(fileName);
        StringBuffer destUrl = new StringBuffer("file:///");
        destUrl.append(destFile.getCanonicalPath().replace('\\', '/'));

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

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

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

```

PropertyValue aProps[] = new PropertyValue[3];

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

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

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

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

```

Using the print function

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

2.3 Text documents

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

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

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

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

- Text (core content)

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

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

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

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

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

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

Word processing

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

- Editing text
Method `setAttributes()` of `com.sun.star.accessibility.XAccessibleEditableText` might not work because the valid char index range of a character string might be beyond the length of the string.
- Inserting text files
Currently, Lotus Symphony documents does not support this function. Developers can create unexpected issues while using the associated APIs provided by OpenOffice 1.1.
- Auto text
The auto text function can be used to organize reusable texts, which is the same as OpenOffice 1.1.

Formatting

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

Navigating

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

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

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

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

Tables

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

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

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

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

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

Text fields

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

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

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

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

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

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

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

Bookmarks

A bookmark is a kind of text content that marks a position inside of a paragraph or a text selection that supports the `com.sun.star.text.TextContent` service. The text document model provides the interface `com.sun.star.text.XBookmarksSupplier` to retrieve and collect the bookmarks.

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

Indexes and index marks

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

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

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

Reference marks

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

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

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

Footnotes and endnotes

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

Shape objects in text

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

Overall document features

Styles

Styles apply document-wide and can differentiate segments in a document that are commonly formatted, and separate this information from the actual formatting. It is a good way to unify the appearance of a document, and customize the formatting of a document by altering a style, instead of using local format settings after the document has been completed. Styles are sets of attributes that can be applied to text or text contents in a text document in a single step.

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

Line and outline numbering

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

Text section

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

Page layout

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

Text document controller

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

Text view

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

TextViewCursor

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

2.4 Spreadsheets

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

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

Operations of spreadsheet documents are mainly in those interfaces :

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

Operations of single cells are in these interface:

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

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

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

User interface refresh

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

```
interface XCellRange;
```

```
void SyncDocument([in] boolean bEnable)
```

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

Sample code:

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

Import external data from a file

Interface XAreaLinks;

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

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

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

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

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

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

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

- Methods to import data from a Web server.

Charts

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

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

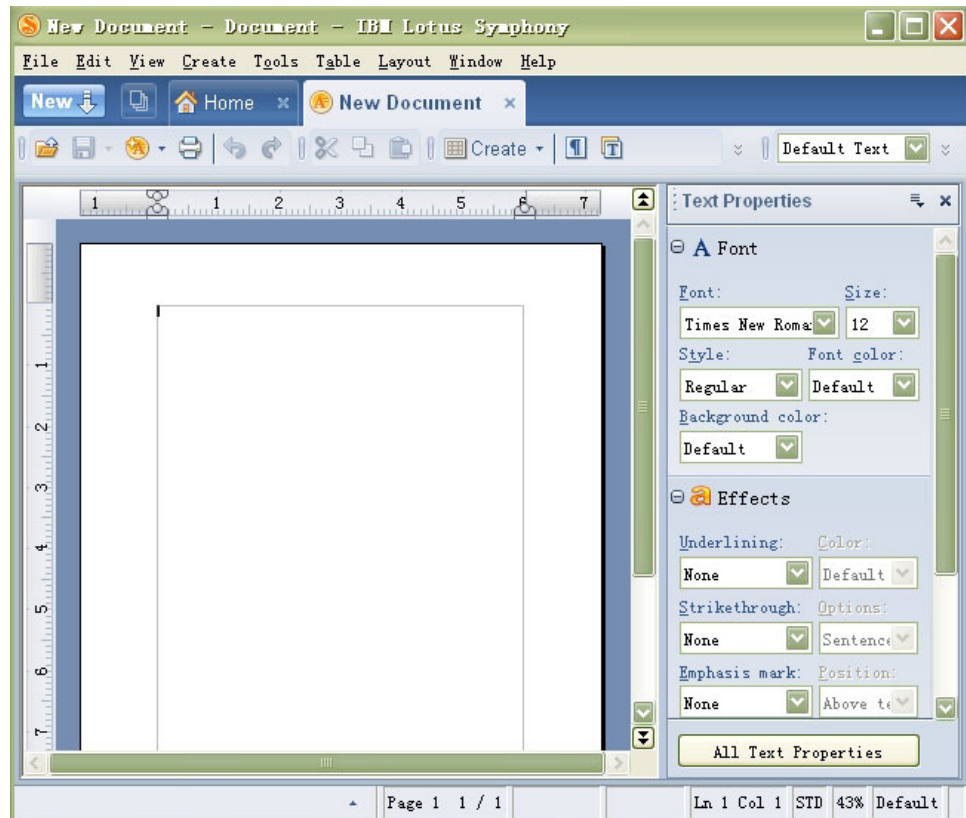
Part 7. Sample plug-ins

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

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



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



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

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

Chapter 1. Hello World sample plug-in

1.1 Creating a new plug-in

Launch the Eclipse development environment

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

1.2 Adding the plug-in dependency

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

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

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

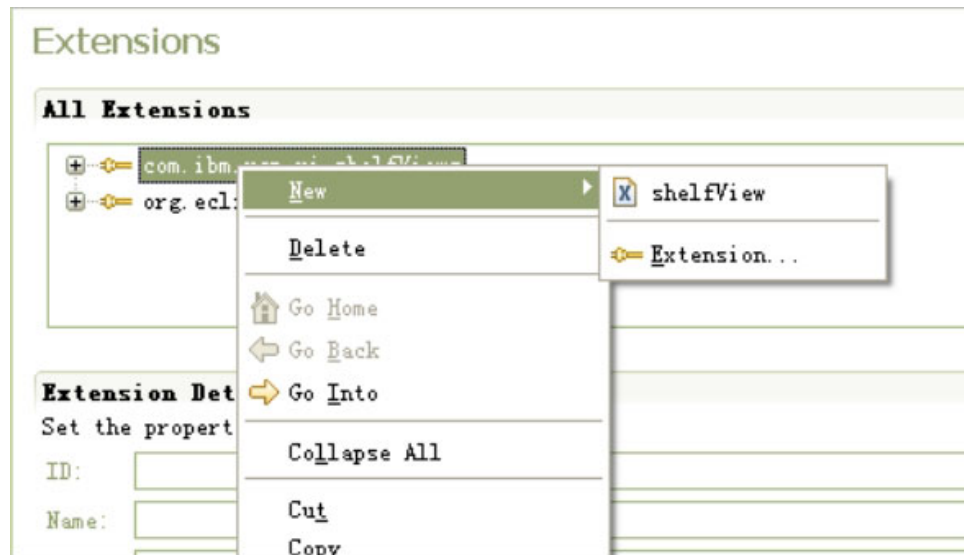
1. Click the **Dependencies** tab of the Hello world plug-in manifest.
2. Click **Add**.
3. Add the following plug-ins:
 - `com.ibm.productivity.tools.ui.views`
 - `com.ibm.productivity.tools.core`

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

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

1.3 Adding a side shelf element

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



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



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

6. Click the **plugin.xml** tab

7. Copy and paste the following into the plugin.xml file.

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

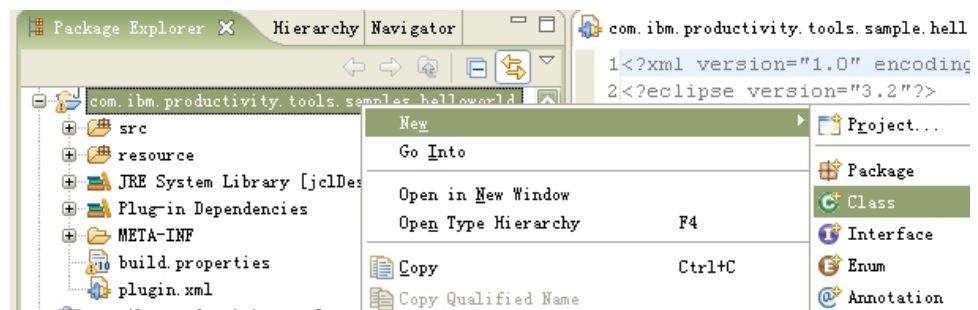
    <view
        name="Hello World"
        icon="resource/Helloworld.gif"
        category="com.ibm.productivity.tools.sample"
        class="com.ibm.productivity.tools.samples.helloworld.ShelfView"
        id="com.ibm.productivity.tools.samples.helloworld.view">
    </view>

</extension>
```

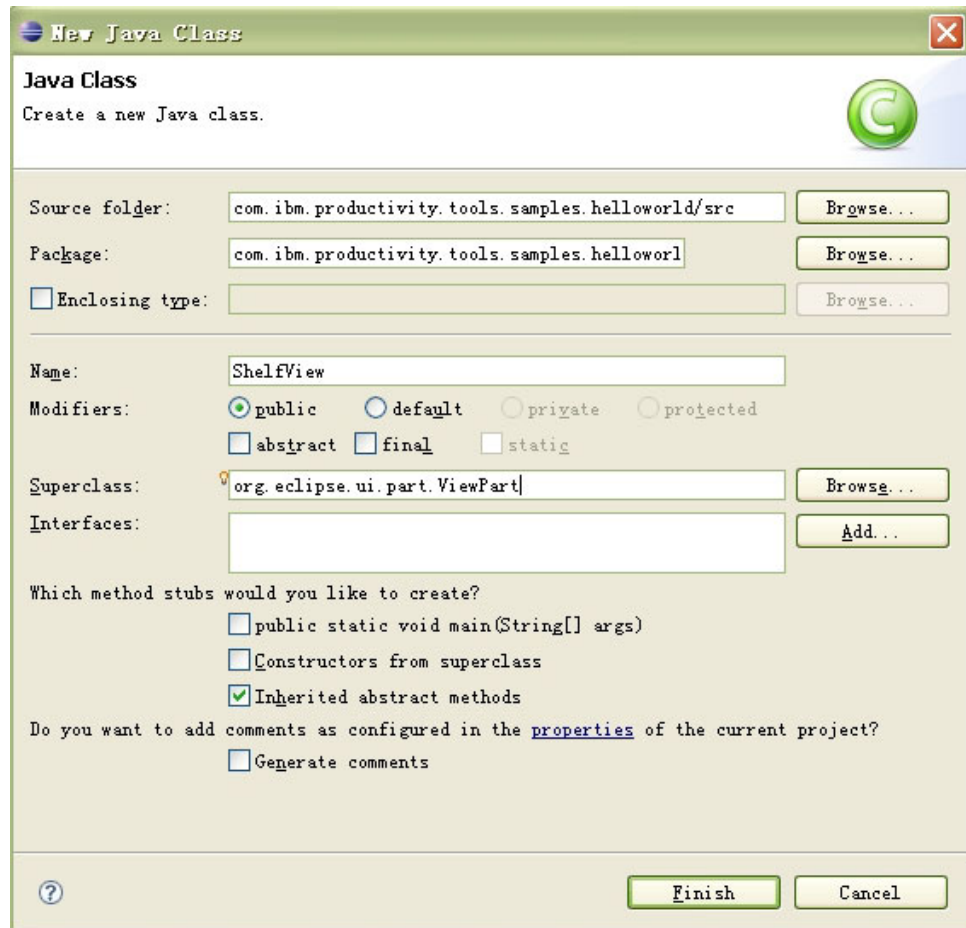
The view attribute of the **<shelfView>** tag in the com.ibm.rcp.ui.shelfViews extension must match the id attribute of the **<view>** tag in the org.eclipse.ui.views extension exactly. That is, the side-shelf content is defined by the extensions **<view>** / **<shelfView>** pairs.

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

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



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



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

1.4 Running the application

1. Check your plug-in.

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

The plugin.xml file is like the following:

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

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

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

    </extension>

  </plugin>
```

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

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

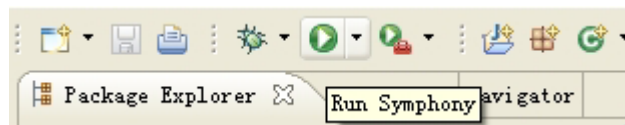
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.part.ViewPart;

public class ShelfView extends ViewPart {

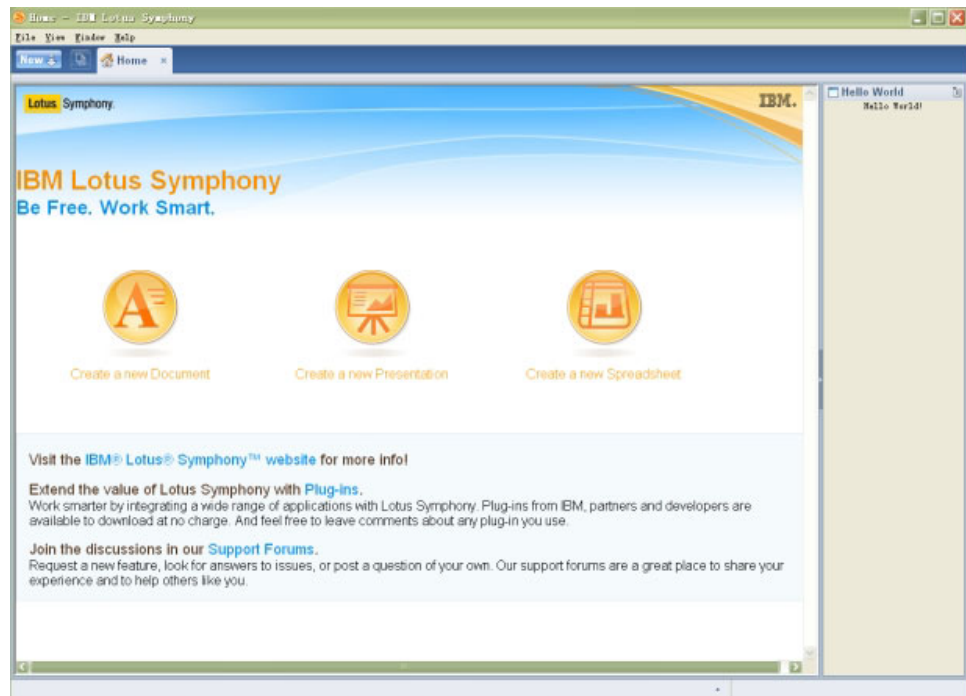
    public void createPartControl(Composite parent) {
        parent.setLayout(new FillLayout());
        Label helloLabel = new Label(parent, SWT.CENTER);
        helloLabel.setText("Hello World!");
    }

    public void setFocus() {
    }
}
```

3. Click **Run** from toolbar:



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



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

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

Chapter 2. Editor View sample plug-in

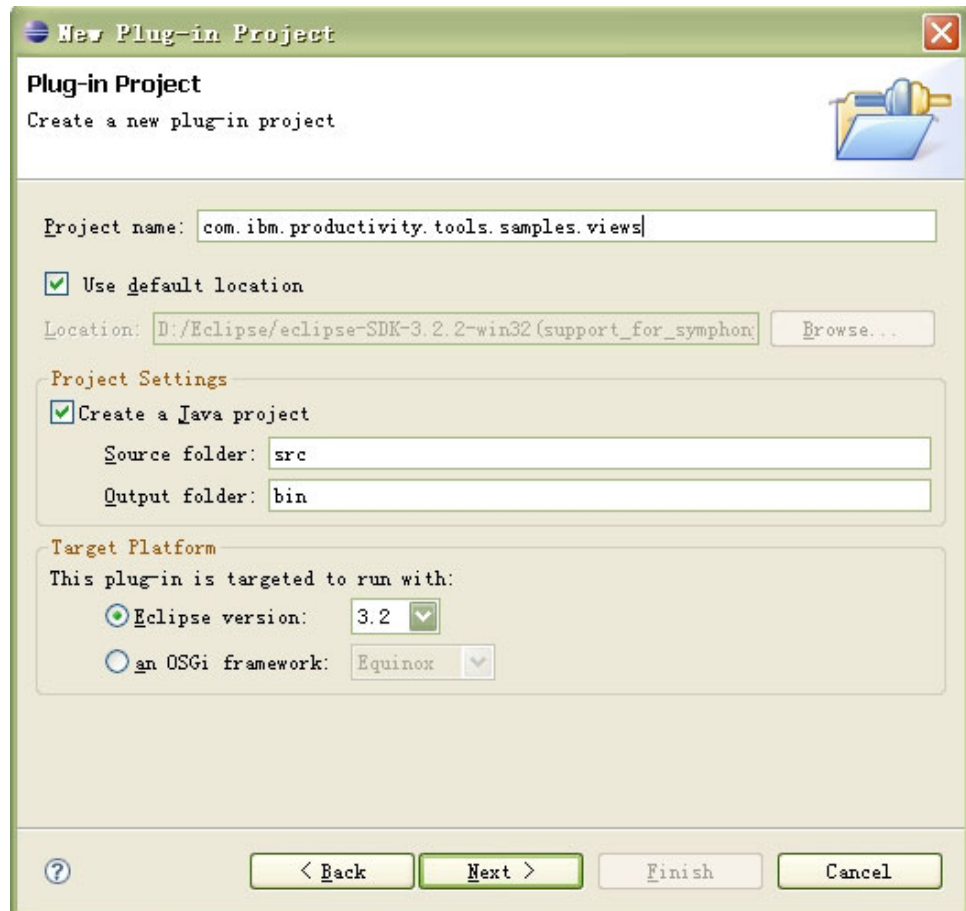
This sample demonstrates how to create a simple editor. When launched from within the new button group, the editor is showed as a view part in a new perspective.

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

```
$symphony_sdk/samples/eclipse/plugins/  
com.ibm.productivity.tools.samples.views.
```

2.1 Creating a plug-in

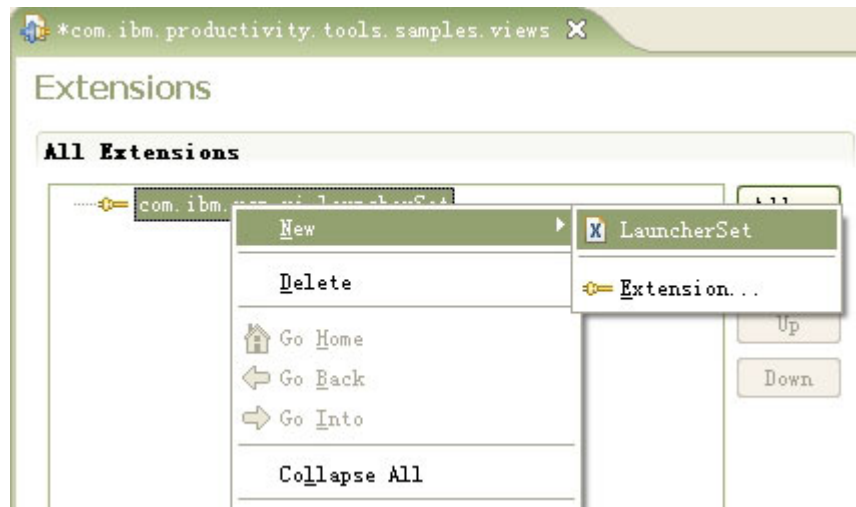
1. Set up the integrated development environment as discussed in Part 4 Chapter 1.
2. New a plug-in project named `com.ibm.productivity.tools.samples.views`, for detailed refer to Create a new plug-in in HelloWorld sample plug-in in Part 6 Chapter 1.



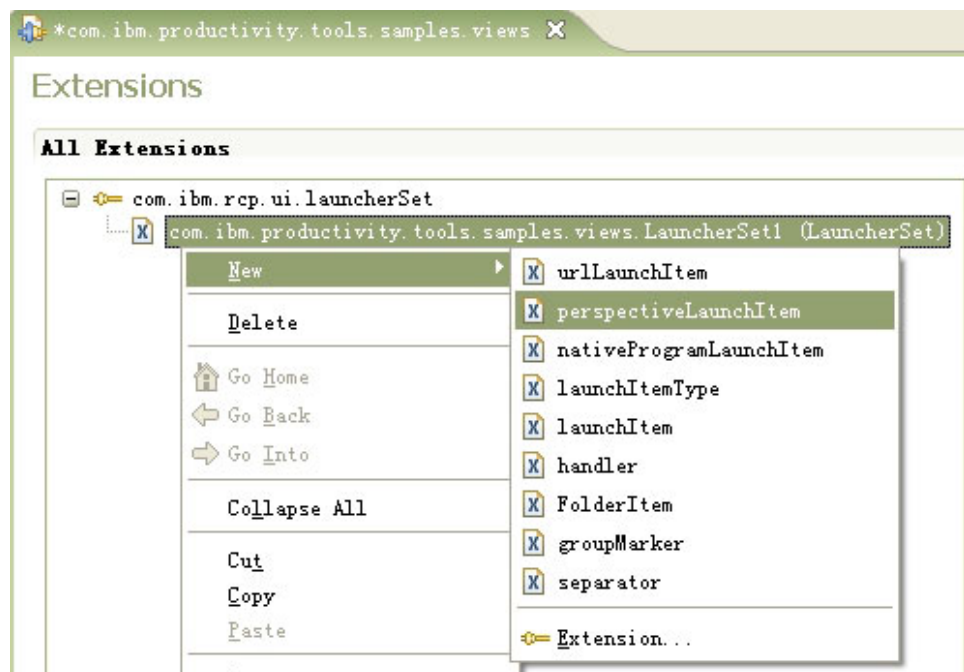
3. Add dependencies. Select the **Dependencies** tab, and click **Add** to add the required plug-ins:
 - com.ibm.rcp.ui
 - com.ibm.productivity.tools.ui.views

2.2 Creating a new button

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



3. Leave the **id** and **label** properties of the LauncherSet unchanged, and save the plugin.xml file.
4. Right-click com.ibm.productivity.tools.samples.views.LauncherSet1, and select **New > perspectiveLaunchItem**.



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

```

<extension
    point="com.ibm.rcp.ui.launcherSet">
    <LauncherSet
        id="com.ibm.productivity.tools.samples.views.LauncherSet1"
        label="%sample.editor2.launcherSet1">
        <perspectiveLaunchItem
            autoStart="false"
            iconUrl="resources/AtualizarDoc.png"
            iconUrlLarge="resources/AtualizarDoc.png"
            iconUrlMedium="resources/AtualizarDoc.png"
            id="com.ibm.productivity.tools.samples.views.perspectiveLaunchItem1"
            label="%sample.editor.spreadsheet"
            perspectiveId="com.ibm.productivity.tools.samples.views
                .WriterPerspective">
        </perspectiveLaunchItem>
    </LauncherSet>
</extension>

```

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



6. Add an extension at extension point `org.eclipse.ui.perspectives`.
7. Select the **plugin.xml** tab. Change the extension declaration of the added perspectives extension point as shown in the following sample code:

```

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

```

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

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

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

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

    public static final String PERSPECTIVE_ID =
        "com.ibm.productivity.tools.samples.views.WriterPerspective";

    public void createInitialLayout( IPageLayout layout ) {
        //set editor area to invisible so that our view can show maximized.
        layout.setEditorAreaVisible(false);

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

2.3 Creating an editor view part

1. Select the **Extensions** tab and click **Add**.
2. Add new extensions org.eclipse.ui.views, then click **Finish**.
3. Select the **plugin.xml** tab, and add the markup as shown in the following sample code:

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

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

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

Name: WriterView

Superclass:

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

and then click **Finish**.

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

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

import org.eclipse.swt.widgets.Composite;

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

public class WriterView extends DefaultRichDocumentView {

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

    public WriterView() {
        super();
    }

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

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

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



Chapter 3. Spreadsheet sample plug-in

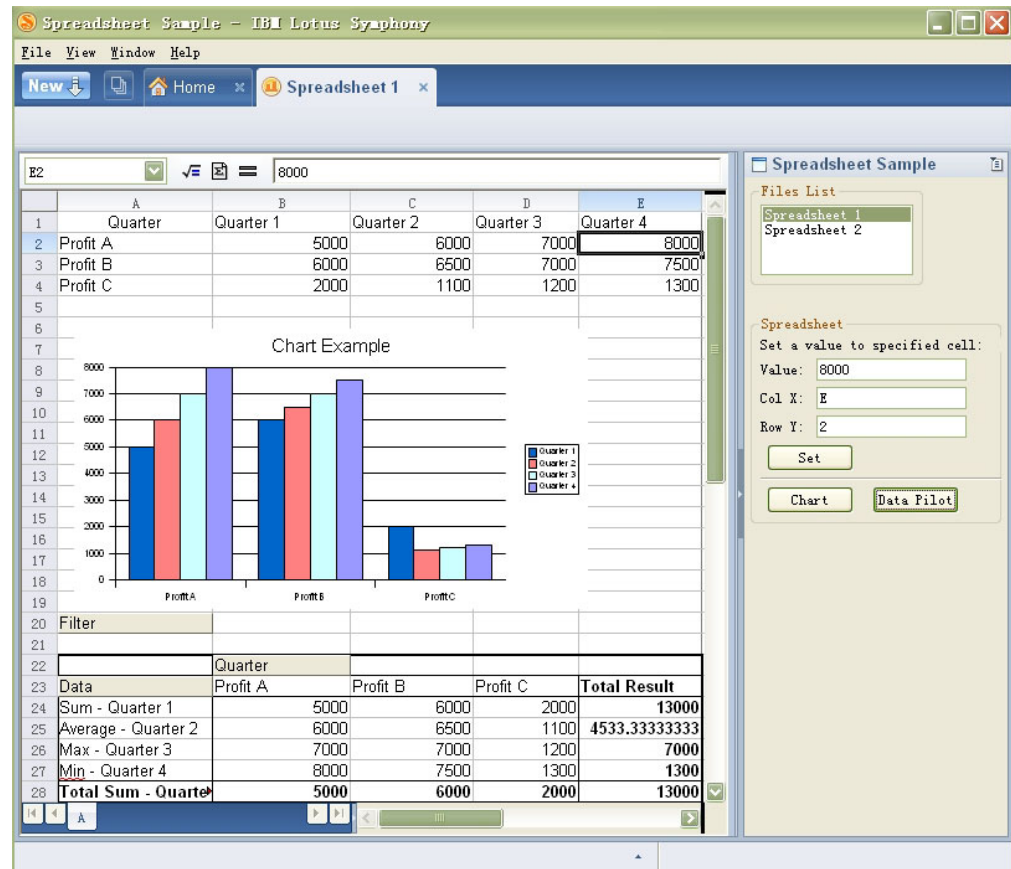
This chapter shows how to begin to add a customized Lotus Symphony spreadsheet UI plug-in and operate the Spreadsheet on a Lotus Symphony side shelf.

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

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

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

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



3.1 Introduction to the scenario

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

3.2 Preview of the result

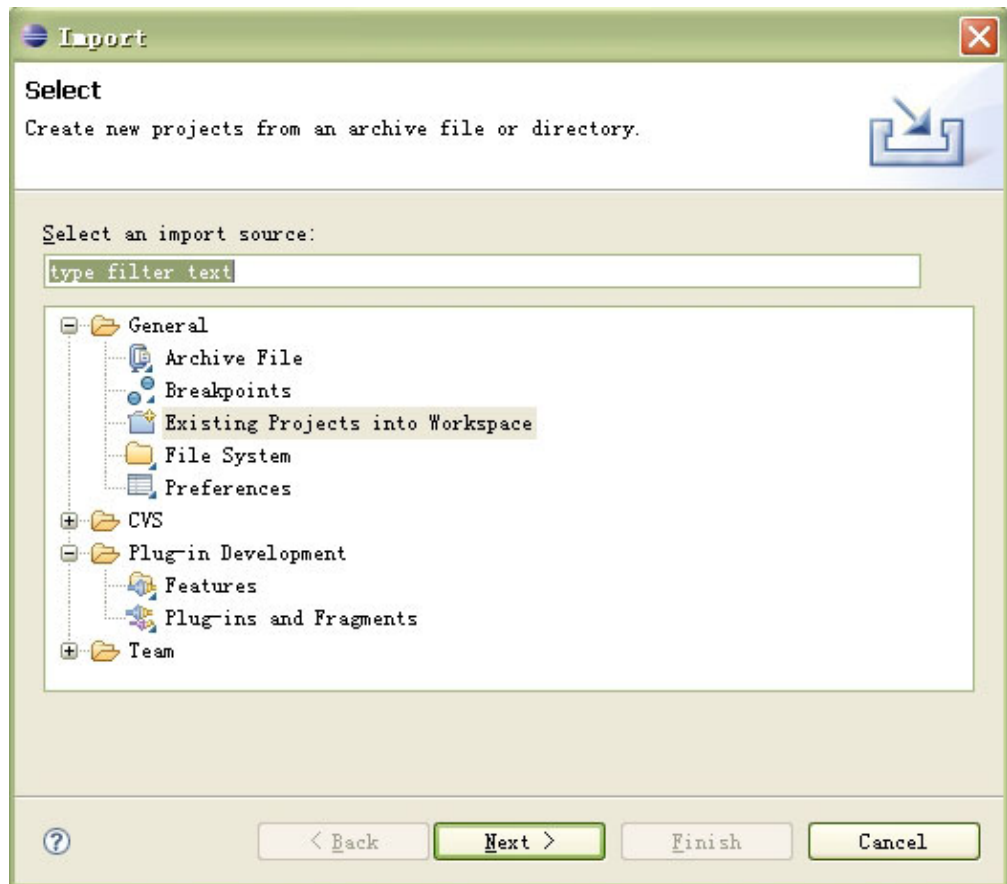
According to the scenario above, this plug-in first creates a shelf view, then adds a list view on the side shelf to show the spreadsheet file list. Then it adds three text fields and a button for setting data to a specified cell. It adds a button for creating a chart and a button for creating a data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

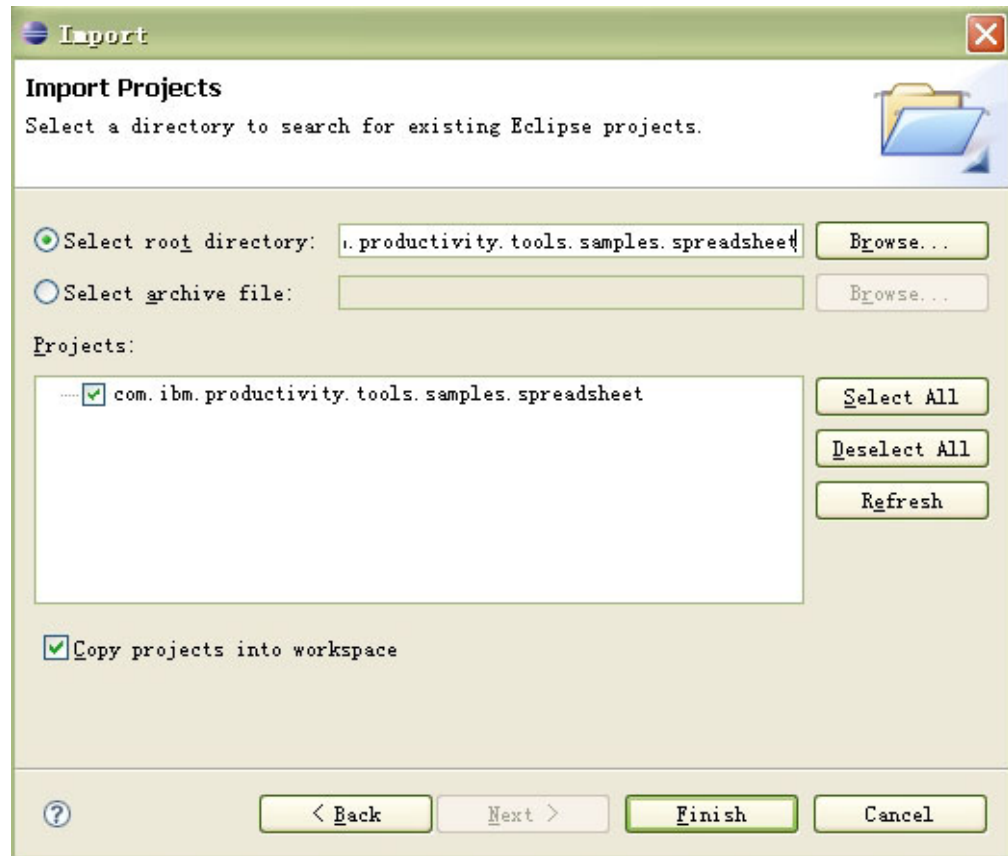
3.3 Prepare your development environment

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

3.4 Deploying the sample

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





3.5 Creating the sample

Creating a new plug-in

1. Launch the Eclipse development environment.
2. New a plug-in project named `com.ibm.productivity.tools.samples.spreadsheet`, for detailed refer to Create a new plug-in in Part 6 Chapter 1 HelloWorld sample plug-in.

New Plug-in Project

Plug-in Content
Enter the data required to generate the plug-in.

Plug-in Properties

Plug-in ID:

Plug-in Version:

Plug-in Name:

Plug-in Provider:

Classpath:

Plug-in Options

☒ Generate an activator, a Java class that controls the plug-in's life cycle
Activator:

☒ This plug-in will make contributions to the UI

Rich Client Application

Would you like to create a rich client application? ☐ Yes ☒ No

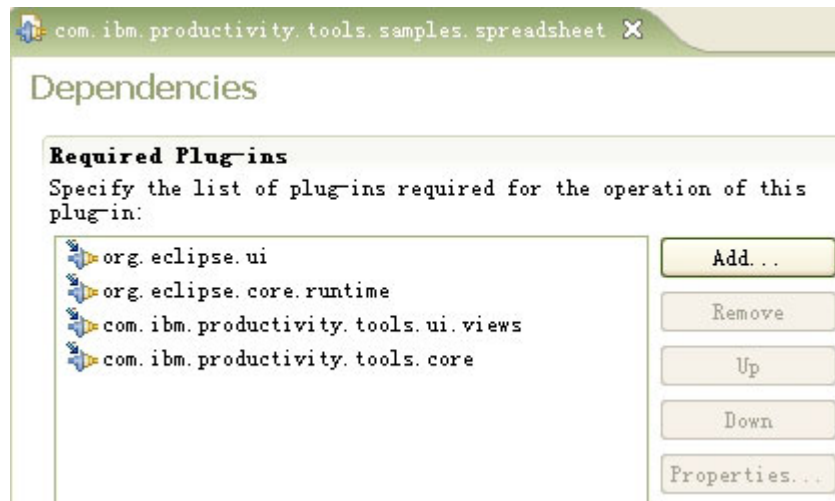
Adding the plug-in dependency

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

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

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

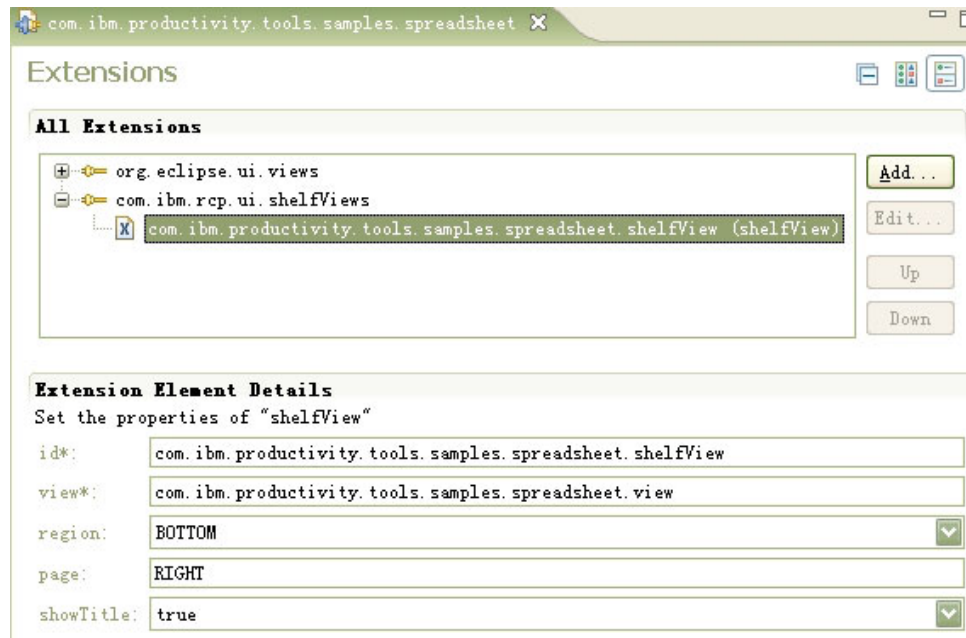
1. Click the **Dependencies** tab of the Spreadsheet sample plug-in manifest.
2. Click **Add**.
3. Add the following plug-ins:
 - com.ibm.productivity.tools.ui.views
 - com.ibm.productivity.tools.core



Adding an element to the side shelf

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

```
<plugin>
<extension
  point="org.eclipse.ui.views">
  <category
    name="Sample Category"
    id="com.ibm.productivity.tools.sample">
  </category>
  <view
    name="Spreadsheet Sample"
    icon="resource/spreadsheetview.gif"
    category="com.ibm.productivity.tools.sample"
    class="com.ibm.productivity.tools.samples.spreadsheet.ui.ShelfView"
    id="com.ibm.productivity.tools.samples.spreadsheet.view">
  </view>
</extension>
<extension
  point="com.ibm.rcp.ui.shelfViews">
  <shelfView
    id="com.ibm.productivity.tools.samples.spreadsheet.shelfView"
    page="RIGHT"
    region="BOTTOM"
    showTitle="true"
    view="com.ibm.productivity.tools.samples.spreadsheet.view"/>
  </extension>
</plugin>
```



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

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

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

3.6 Core code demonstration

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

1. Add a side shelf to the Lotus Symphony.
Refer to the section **Adding a side shelf element** of Chapter 1 Hello World sample plug-in .
2. Open a spreadsheet file and get this sheet's model.

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

3. Set a value in a cell:

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

```

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

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

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

```

4. Create a chart for this sheet.

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

```

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

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

```

5. Create a data pilot for this sheet:

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

```

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

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

```

3.7 Extending the sample

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

Chapter 4. Writer Sample plug-in

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

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

6. Creating user-defined fields

4.1 Introduction to the scenario

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

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

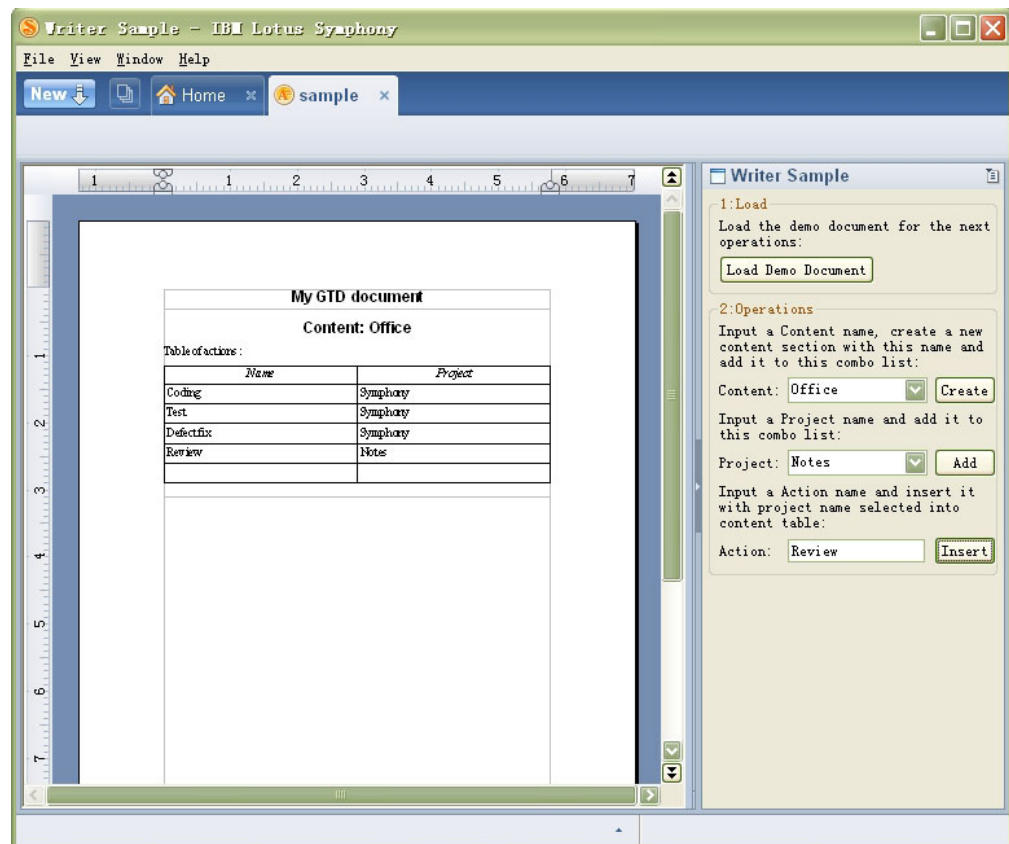
1. Context. A context refers to locations or situations, such as home, computer, work and errands, that are suitable for doing a certain kind of to-dos.
2. Project. A project can be, for example, Repaint bedroom or Review report.
3. Action. An action is a to-do item.

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

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

4.2. Preview of the result

The UI of this plug-in to manipulate the writer document and a GTD document this plug-in created are as follows:



From this figure we can see that:

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

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

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

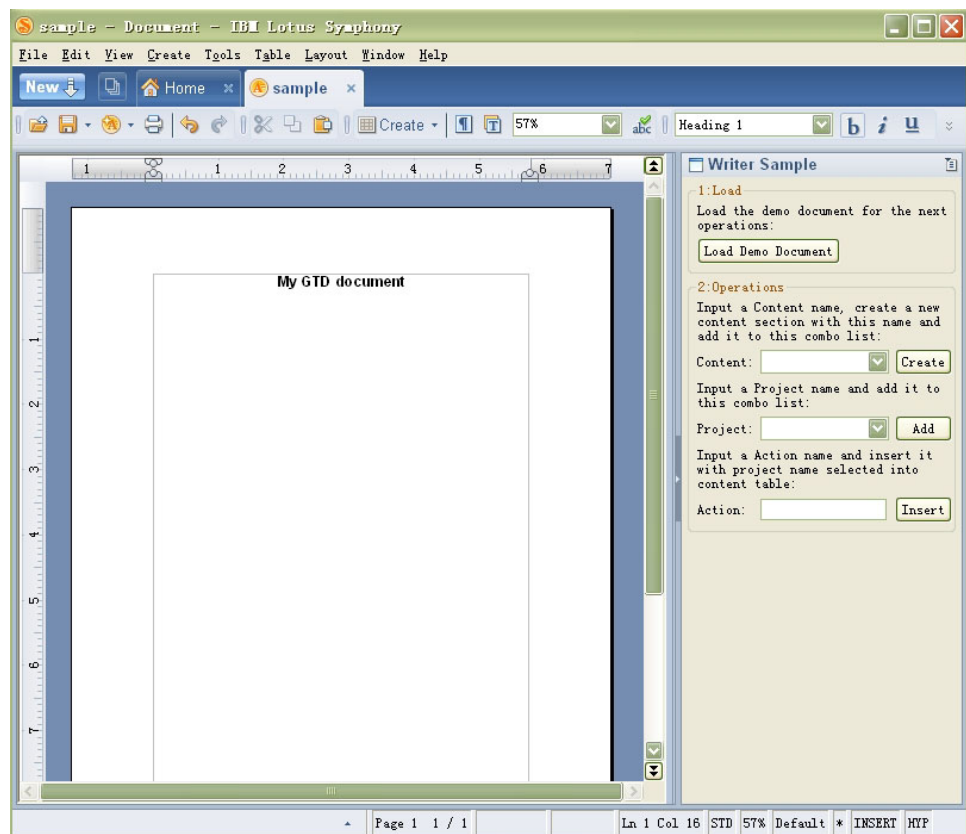
4.3 Deploying the sample

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

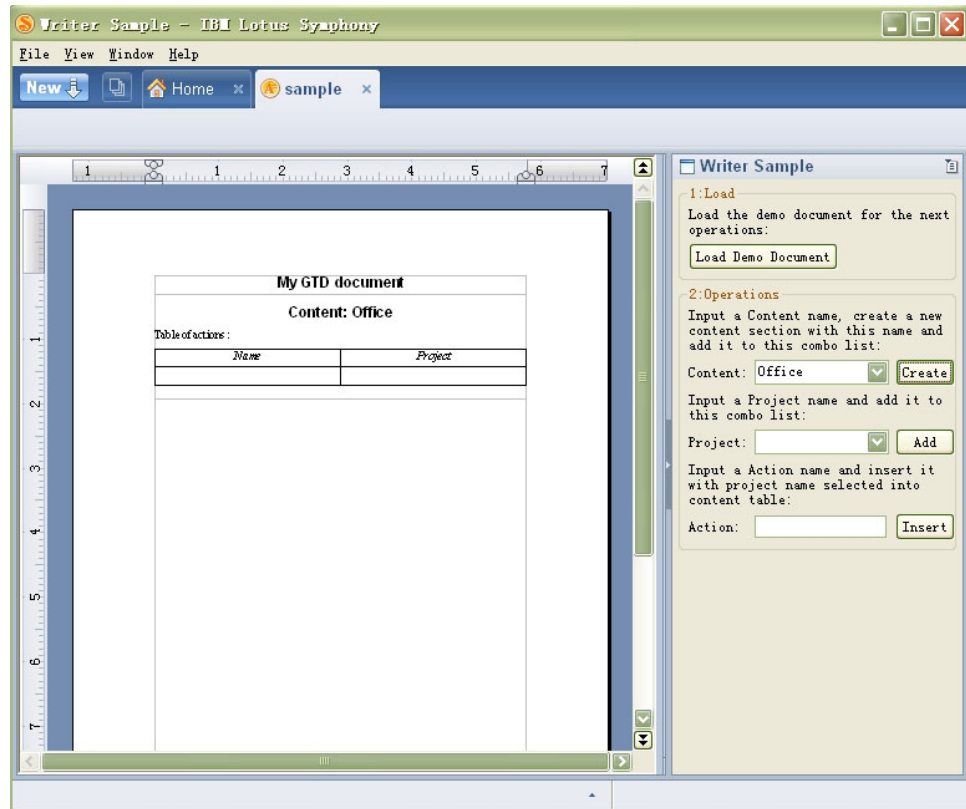
4.4 Using the sample

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

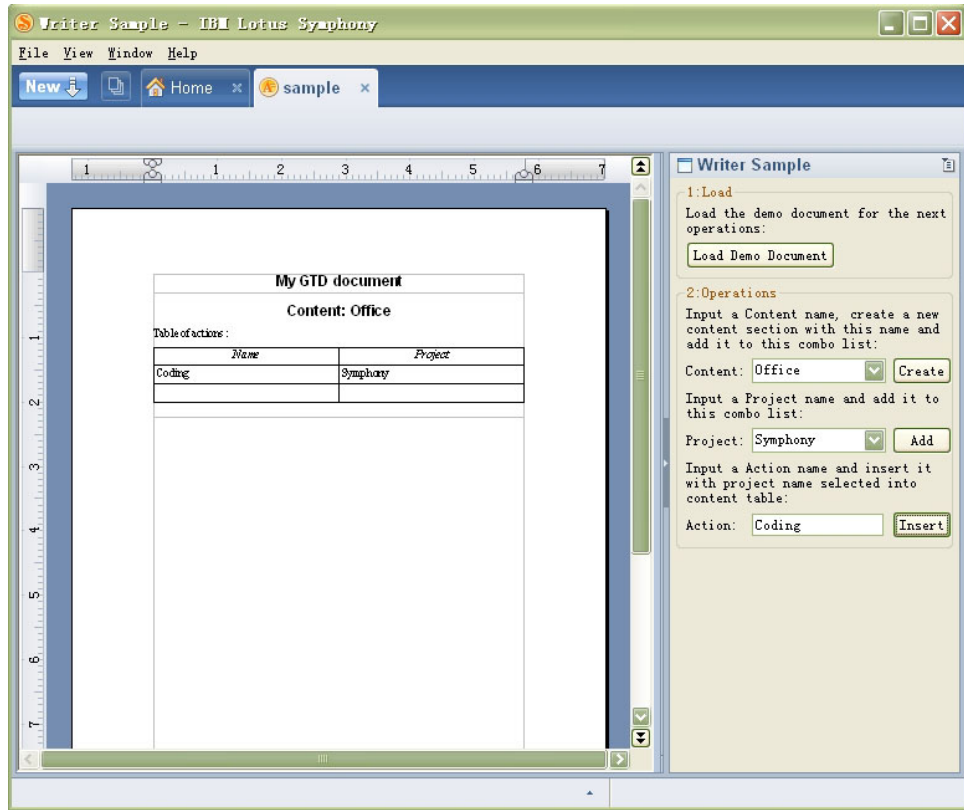
1. Click **Load Demo Document**. An empty GTD document is opened in a new page.



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



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



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

4.5 Building the sample

Prepare your development environment

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

Creating the sample

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

```

<extension
    point="com.ibm.rcp.ui.shelfViews">
    <shelfView
        id="com.ibm.productivity.tools.samples.writer.shelfView1"
        page="RIGHT"
        region="TOP"
        showTitle="true"
        view="com.ibm.productivity.tools.samples.writer.demoView"/>
    </extension>

```

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

```

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

```

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

```

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

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

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

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

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

```

```

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

```

```

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

```

```

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

```

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

Chapter 5. Presentation sample plug-in

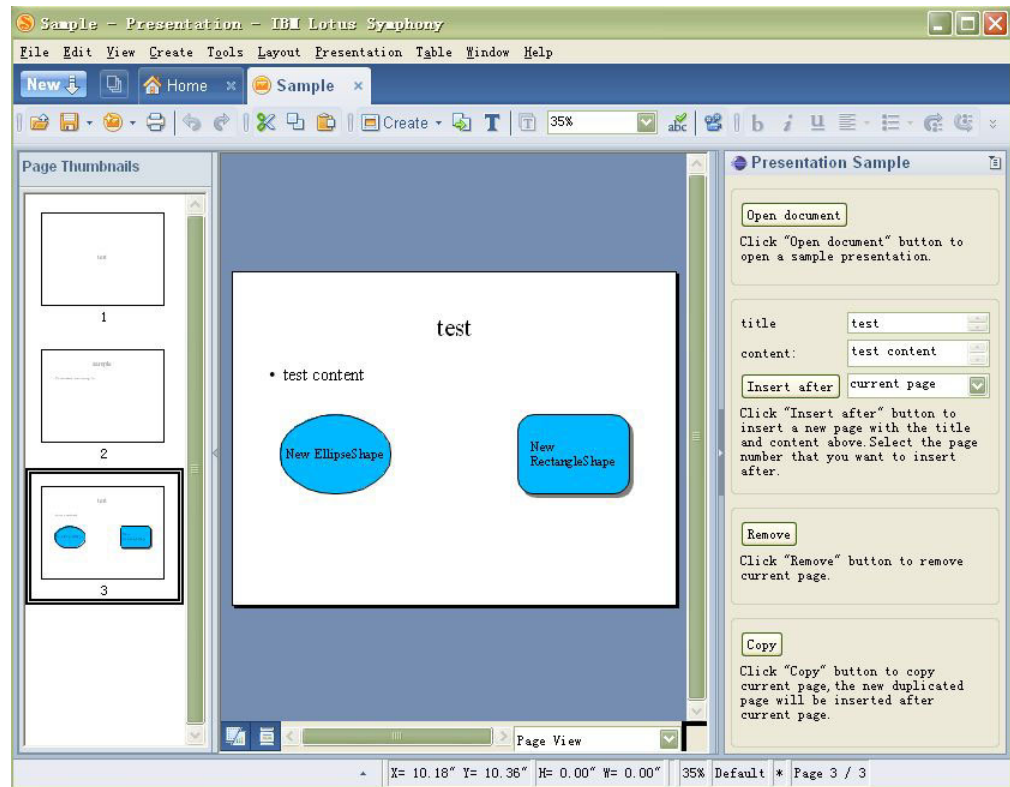
This chapter shows how to work with presentation documents.

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

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

1. Add a customized shelf view.
2. Open a presentation.
3. Add a `ISelectionListener` to listen for selection change.
4. Insert a draw page into the document.
5. Create shapes.
6. Set master page for a draw page.
7. Remove a draw page.
8. Copy a draw page.

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



5.1 Introduction to the scenario

You might want to generate a presentation from a database or files. First you need to open the presentation and get its model before you can insert a new page. Also you might want to set the master page for the new inserted page, or copy and remove some pages. The basis operations for presentations include opening the document, inserting, removing, or copying a page, and setting the master page.

5.2 Preview of the result

According to the scenario above, this plug-in first creates a shelf view, then adds four groups respectively for opening the document, inserting draw pages, removing draw pages and copying draw pages. There are two text fields for inputting the title and content of a new page in the insert page group.

5.3 Prepare your development environment

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

5.4 Deploying the sample

If you already have this plug-in, you can import it into Eclipse from an existing project by using the Eclipse import function. For how to import this plug-in, refer to Part 6 Chapter 3 section 3.4 *Deploying the sample*.

5.5 Creating the sample

Creating a new plug-in

1. Launch the Eclipse development environment.

2. New a plug-in project named `com.ibm.productivity.tools.samples.presentation`. For detailed instructions refer to *Create a new plug-in* in Part 6 Chapter 1 HelloWorld sample plug-in.

New Plug-in Project

Plug-in Content
Enter the data required to generate the plug-in.

Plug-in Properties

Plug-in ID: `com.ibm.productivity.tools.samples.presentation`

Plug-in Version: `1.0.0`

Plug-in Name: `Presentation sample`

Plug-in Provider: `IBM`

Classpath:

Plug-in Options

☒ Generate an activator, a Java class that controls the plug-in's life cycle
Activator: `com.ibm.productivity.tools.samples.presentation.Activator`

☒ This plug-in will make contributions to the UI

Rich Client Application

Would you like to create a rich client application? ☐ Yes ☒ No

Adding the plug-in dependency

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

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

Add the plug-in dependency listed above. For detailed instructions, refer to interrelated content in Part 6 Chapter 3.

Adding an element to the side shelf

1. On the **Extensions** tab, add an extension on the extension point `com.ibm.rcp.ui.shelfViews`. Change the part of the `plugin.xml` file.

2. Copy and paste the following sample code into the plugin.xml file.

```
<plugin>

    <extension point="org.eclipse.ui.views">
        <category
            name="Sample Category"
            id="com.ibm.productivity.tools.sample">
        </category>
        <view
            name="Presentation Sample"
            icon="resource/icon/sample.gif"
            category="com.ibm.productivity.tools.sample"
            class="com.ibm.productivity.tools.samples.presentation.ui
                .SampleShelfView"
            id="com.ibm.productivity.tools.samples.presentation.ui
                .SampleShelfView">
        </view>
    </extension>

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

</plugin>
```

3. Create a folder named ui and a class named ShelfView which extends org.eclipse.ui.part.ViewPart under this folder. The main method in this class is shown in the following sample code:

```
public void createPartControl(Composite parent) {
    parent.setLayout(new GridLayout());
    createOpenGroup(parent);
    createInsertPageGroup(parent);
    createRemoveGroup(parent);
    createCopyGroup(parent);
    addSelectionListener();
    refreshView(xModel);
}
```

The method createOpenGroup() creates a button named **Open document** to open a sample presentation. The method createInsertPageGroup() creates two text fields and a button named **Insert after** to insert a page with title and content, and set the master page for the new inserted page. The method createRemoveGroup() creates a button named **Remove** to remove current page. The method CreateCopyGroup() creates a button named **Copy** to copy current page. The method addSelectionListener() add a listener for listening selection changes.

5.6 Core code demonstration

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

1. Add a side shelf to Lotus Symphony.

Refer to the section *Adding a side shelf element* in Part 6 Chapter 1 Hello world sample plug-in.

2. Open a presentation file.

```
// the parameter url is the presentation file's url.  
RichDocumentView tempView = RichDocumentViewFactory.openView(url, false);
```

3. Add a selection listener to listen for selection change, and get the selected document's model:

```
SelectionService service = PlatformUI.getWorkbench().  
    getActiveWorkbenchWindow().getSelectionService();  
service.addSelectionListener(new ISelectionListener() {  
    public void selectionChanged(IWorkbenchPart arg0, ISelection arg1) {  
        IAdaptable adaptable = (IAdaptable) arg1;  
        RichDocumentViewSelection selection = (RichDocumentViewSelection)  
            adaptable.getAdapter(RichDocumentViewSelection.class);  
        //get selected document  
        RichDocumentView view = selection.getView();  
        Object unoModel = view.getUNOModel();  
        xModel = (XModel)UnoRuntime.queryInterface(XModel.class, unoModel);  
        refreshView(xModel);  
    }  
});
```

4. Insert a draw page into a presentation document.

```
//first get the the container of draw pages.  
XDrawPagesSupplier xDrawPagesSupplier = (XDrawPagesSupplier)UnoRuntime  
    .queryInterface(XDrawPagesSupplier.class, xComponent);  
XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();  
  
//then insert a new page.  
XDrawPage xNewDrawPage = xDrawPages.insertNewByIndex(pageIndex);
```

5. Add a shape to a draw page.

The code below demonstrates how to create a shape and add a shape to a draw page. The parameters pos and size are position and size of the shape. The shapeType is the service name for the shape, such as `com.sun.star.drawing.EllipseShape`. The text is the text that shown on the shape.

```

XShape addShape(XComponent xComponent,XDrawPage page,Point pos,
Size size, String shapeType,String text){
    XShape xShape = null;
    try {
        XShapes xShapes = (XShapes)UnoRuntime.queryInterface( XShapes.class, page );
        XMultiServiceFactory xFactory = (XMultiServiceFactory )UnoRuntime
        .queryInterface(XMultiServiceFactory.class, xComponent );
        Object xObj = xFactory.createInstance( shapeType );
        xShape = (XShape)UnoRuntime.queryInterface(XShape.class, xObj );
        xShape.setPosition( pos );
        xShape.setSize( size );
        //add the shape
        xShapes.add( xShape );
        //set text for the shape
        XText xText = (XText)UnoRuntime.queryInterface( XText.class, xShape );
        xText.setString(text);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return xShape;
    }
}

```

6. Get a master page.

```

int index = ...
XMasterPagesSupplier xMasterPagesSupplier = (XMasterPagesSupplier) UnoRuntime
    .queryInterface(XMasterPagesSupplier.class, xComponent);
XDrawPages xMasterPages = xMasterPagesSupplier.getMasterPages();
XDrawPage masterPage = null;
try {
    Object obj = xMasterPages.getByIndex(index);
    masterPage = (XDrawPage) UnoRuntime.queryInterface(XDrawPage.class,obj);
} catch (IndexOutOfBoundsException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (WrappedTargetException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

```

7. Set a master page for a draw page.

```

XdrawPage xNewDrawPage = ...
//query for MasterPageTarget of draw page
XMasterPageTarget xMasterPageTarget = (XMasterPageTarget)UnoRuntime
    .queryInterface(XMasterPageTarget.class, xNewDrawPage);
//set master page
xMasterPageTarget.setMasterPage(masterPage);

```

8. Remove a draw page.

```

//first get the the container of draw pages.
XdrawPages XdrawPages = ...
XdrawPage xDrawPage = ...
// remove the draw page
xDrawPages.remove(xDrawPage);

```

9. Copy a draw page.

```
// get page duplicator
XDrawPageDuplicator duplicator= (XDrawPageDuplicator)UnoRuntime
    .queryInterface(XDrawPageDuplicator.class, xModel);
//duplicate draw page.
duplicator.duplicate(xDrawPage);
```

5.7 Extending the sample

Next, you can add more functions to operating a presentation, such as loading, saving, and closing a presentation. You can also create a sample chart or set presentation effects.

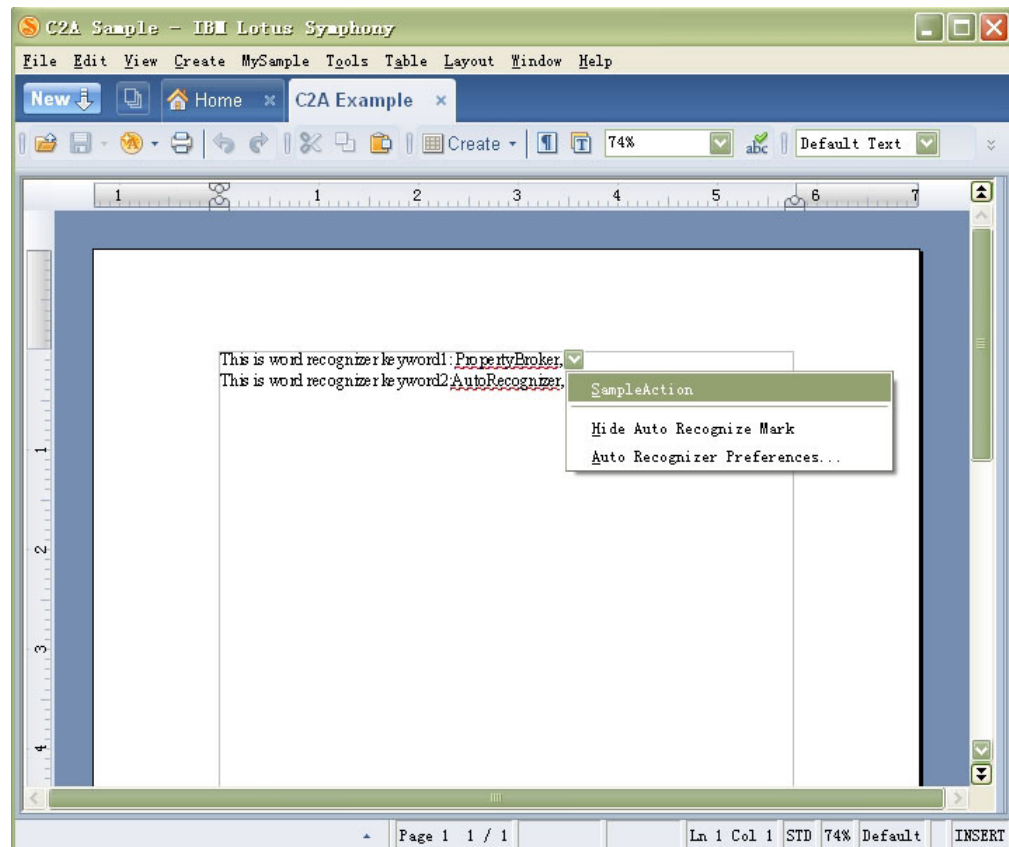
Chapter 6. C2A sample plug-in

This sample demonstrates how to develop Click to Action (C2A) applications based on Lotus Symphony.

Note: All sample code used within this chapter can be found in the Lotus Symphony toolkit, such as `$Symphony_sdk/samples/eclipse/plugins/com.ibm.productivity.tools.samples.C2A`. You can get this toolkit from the site: <http://symphony.lotus.com/software/lotus/symphony/developers.jspa>.

In the C2A sample plug-in, two keywords are predefined in the sample recognizer: `PropertyBroker` and `AutoRecognizer`. The auto recognizer engine calls the detector to check if there are such keywords found. When users click the context menu for recognized tags, a message window is opened.

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



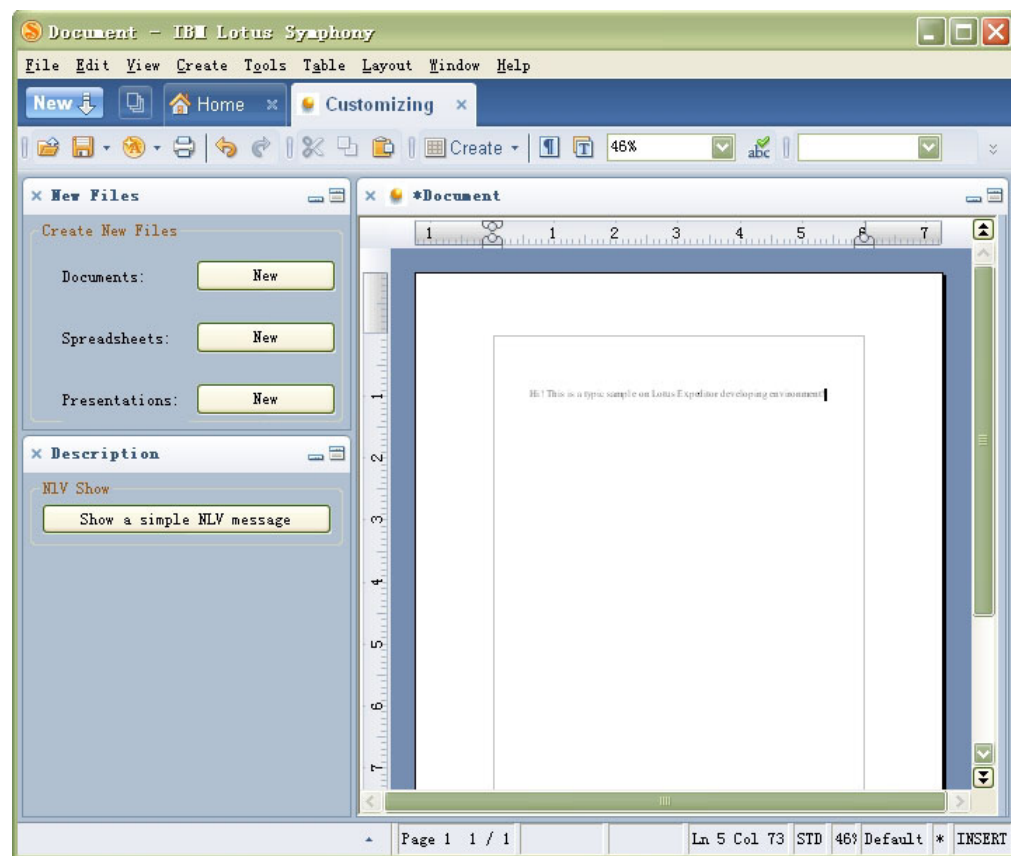
For more details about how to use the auto recognizer, please refer to Part 4 Chapter 2 Section 2.5.

Chapter 7. Customizing a Sample plug-in

In this sample plug-in, it shows :

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

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



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

7.1 Introduction to the scenario

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

7.2 Preview of the result

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

7.3 Prepare development environment

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

7.4 Deploying the sample

If you already have this plug-in, you can import it into Eclipse from an existing project using the Eclipse import function. For how to import this plug-in, refer to Part 6 Chapter 3 section 3.4 Deploying the sample.

7.5 Creating the sample

Create a new plug-in

1. Launch the Eclipse development environment.
2. New a plug-in project named `com.ibm.productivity.tools.samples.presentation`, for detailed refer to Create a new plug-in in Part 6 Chapter 1 HelloWorld sample plug-in.

New Plug-in Project

Plug-in Content
Enter the data required to generate the plug-in.

Plug-in Properties

Plug-in ID: `com.ibm.productivity.tools.samples.presentation`

Plug-in Version: `1.0.0`

Plug-in Name: `Presentation sample`

Plug-in Provider: `IBM`

Classpath:

Plug-in Options

☒ Generate an activator, a Java class that controls the plug-in's life cycle
Activator: `com.ibm.productivity.tools.samples.presentation.Activator`

☒ This plug-in will make contributions to the UI

Rich Client Application

Would you like to create a rich client application? ☐ Yes ☒ No

Buttons:

Add the plug-in dependency

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

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

Add the plug-in dependency list above, for detailed, refer to interrelated content in Part 6 Chapter 3.

Adding a perspective and views

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

7. Copy and paste the following sample code into the plugin.xml file.

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

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

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

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

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

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

```

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

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

</extension>

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

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

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

</plugin>

```

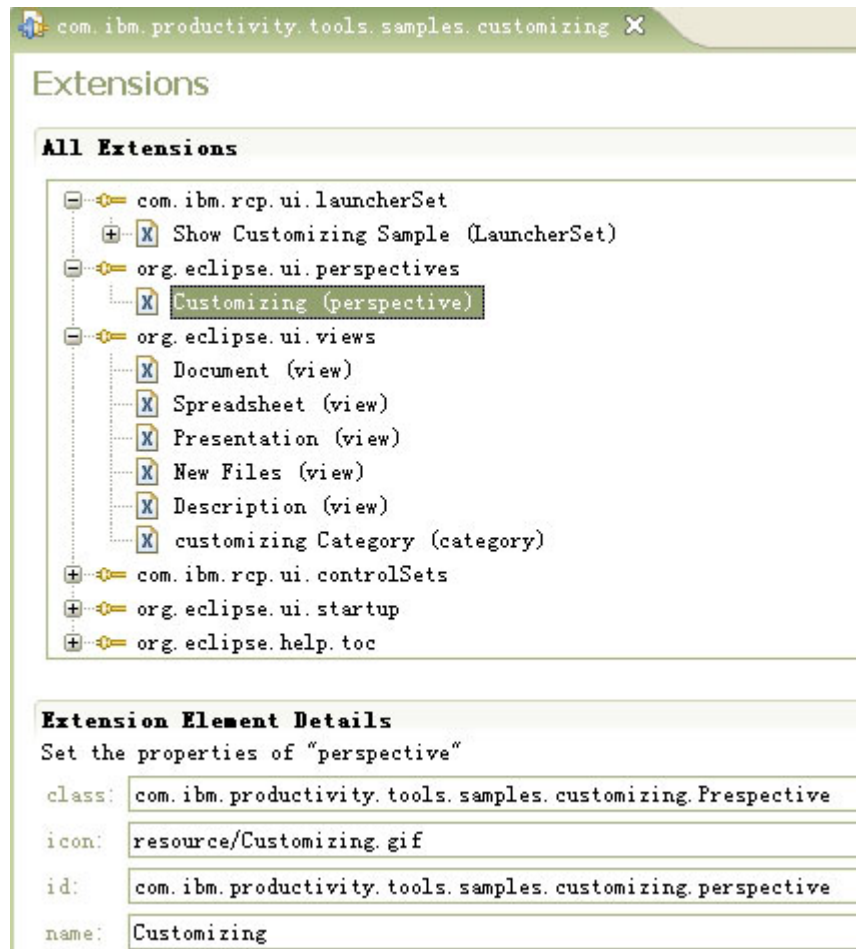
8. Create a class named `Prespective` which implements `IPerspectiveFactory`. The main method in this class is shown in the following sample code.

```

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

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

```



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

7.6 Core code demonstration

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

1. Add a launcher item to launch a perspective.

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

2. Add a custom view and Lotus Symphony views.

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

3. Add a status bar.

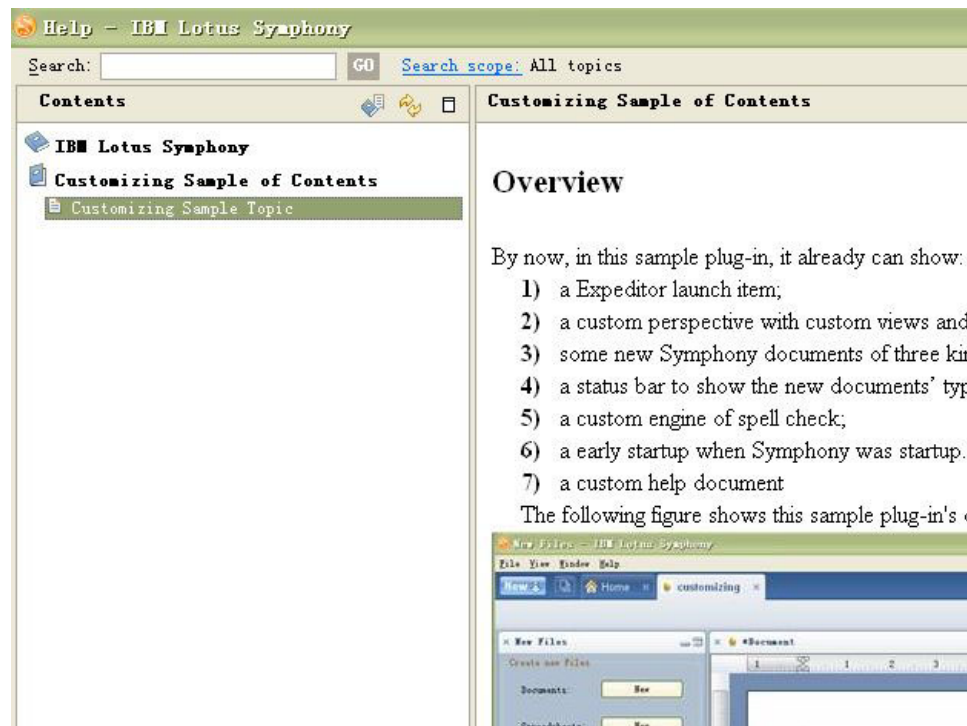
Refer to Part 4 Section 2.6 Chapter 2.

4. Add a custom early startup.

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

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

5. Add a custom help topic:



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

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

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

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

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

7.7 Extending the sample

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

Chapter 8. Convertor Sample plug-in

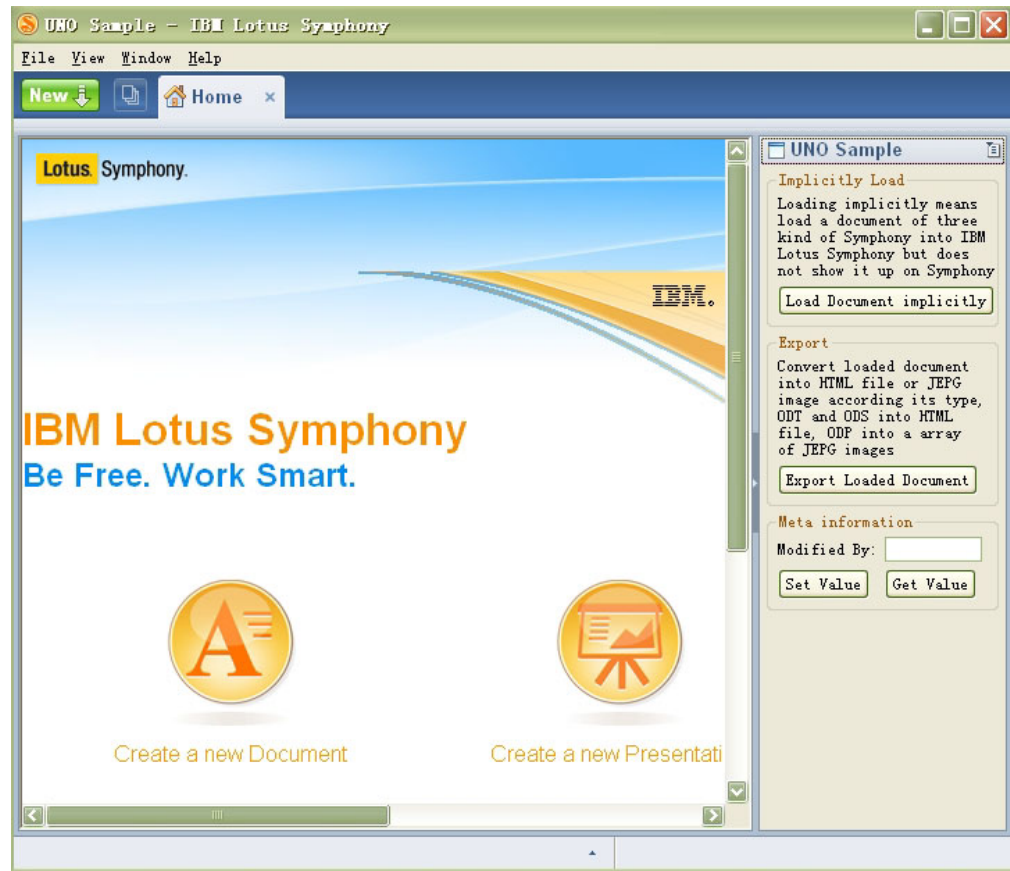
This plug-in sample shows a typical application of loading three kinds of Lotus Symphony documents implicitly, which means loading a document into Lotus Symphony but does not show it up on Lotus Symphony. The sample will export this loaded document into HTML or JPEG format according its type. A sample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

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

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

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

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



8.1 Introduction to the scenario

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

8.2 Preview of the result

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

8.3 Prepare development environment

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

8.4 Deploying the sample

If you already have this plug-in, you can import it into Eclipse from an existing project using the Eclipse import function. For how to import this plug-in, refer to Part 6 Chapter 3 section 3.4 Deploying the sample.

8.5 Design overview

This sample has these goals:

1. Add a side shelf.

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

8.6 Creating the sample

Creating a new plug-in

1. Launch the Eclipse development environment.
2. New a plug-in project named `com.ibm.productivity.tools.samples.convertor`, for detailed refer to Create a new plug-in in Part 6 Chapter 1 HelloWorld sample plug-in.

New Plug-in Project

Plug-in Content
Enter the data required to generate the plug-in.

Plug-in Properties

Plug-in ID: `com.ibm.productivity.tools.samples.convertor`

Plug-in Version: `1.0.0`

Plug-in Name: `Convertor sample`

Plug-in Provider: `IBM`

Classpath:

Plug-in Options

☒ Generate an activator, a Java class that controls the plug-in's life cycle
Activator: `com.ibm.productivity.tools.samples.convertor.Activator`

☒ This plug-in will make contributions to the UI

Rich Client Application

Would you like to create a rich client application? ☐ Yes ☒ No

Adding the plug-in dependency

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

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

Add the plug-in dependency list above, for detailed, refer to interrelated content in Part 6 Chapter 3 .

Adding shelf views

1. On the **Extensions** tab, add an extension on the extension point `com.ibm.rcp.ui.shelfViews`. Change the part of the `plugin.xml` file corresponding to the extension with:
2. Copy and paste the following sample code into the `plugin.xml` file.

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

8.7 Core code demonstration

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

1. Get the `com.sun.star.lang.XMultiServiceFactory` object reference. Refer to [Getting the global service factory](#).

2. Load the Lotus Symphony document by file path implicitly.

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

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

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

3. Resolve the document type.

The following sample code shows how to resolve document type.

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

4. Export the documents into a HTML file.

Refer to [Exporting documents and drawing objects](#).

5. Export the document into a JPEG image

Refer to [Exporting documents and drawing objects](#).

8.8 Extending the sample

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

Part 8. Troubleshooting and support

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

In the following chapters are some typical issues and solutions. If you have more questions, contact support at Lotus Symphony Web site <http://symphony.lotus.com/software/lotus/symphony/developers.nsf/home>. Common issues during the development on Louts Symphony are already documented and resolved.

Chapter 1. Troubleshooting the development environment

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

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

1. Eclipse 3.4.0
2. Lotus Expeditor toolkit 6.2.0
3. Lotus Symphony profile tool from the Lotus Symphony toolkit

Chapter 2. Troubleshooting during application development

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

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

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

Chapter 3. Troubleshooting during deployment

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

Solution: Perform the following steps to resolve the problem:

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

Chapter 4. Contacting support

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

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

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

Part 9. Appendixes

Appendix . References

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

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

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

For Lotus Notes 8, refer to the following site:

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

For composite applications, refer to the following sites:

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

[http://www-306.ibm.com/software/lotus/products/notes/
compositeapplications.html](http://www-306.ibm.com/software/lotus/products/notes/compositeapplications.html)

Appendix . Notices

Notices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

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

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol ([®] or [™]), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information (www.ibm.com/legal/copytrade.shtml).

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



Printed in USA