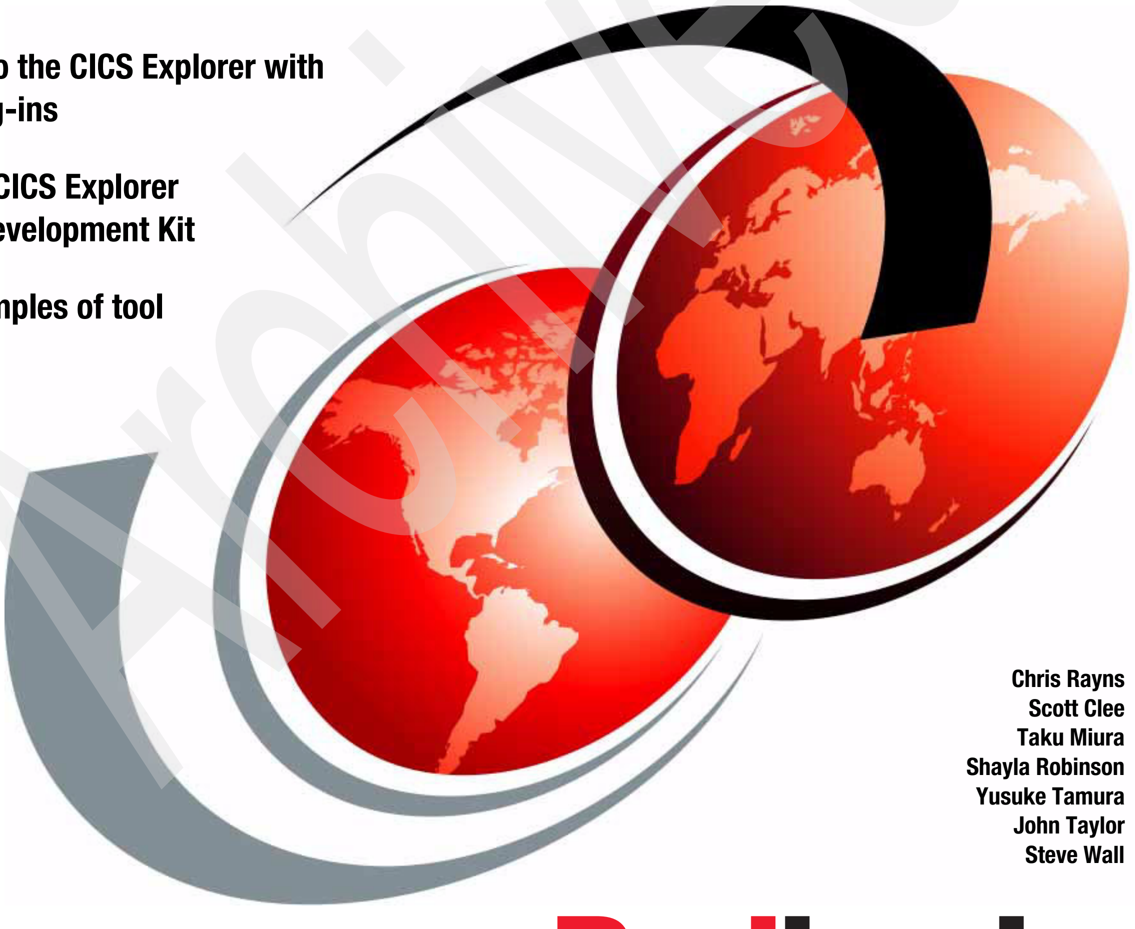
IBM

Extend the CICS Explorer
A Better Way to Manage Your CICS



Add value to the CICS Explorer with Eclipse plug-ins

Unlock the CICS Explorer Software Development Kit

Follow examples of tool integration

Chris Rayns
Scott Clee
Taku Miura
Shayla Robinson
Yusuke Tamura
John Taylor
Steve Wall

Redbooks

ibm.com/redbooks

International Technical Support Organization

Extend the CICS Explorer: A Better Way to Manage Your CICS

October 2009

SG24-7819-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (October 2009)

This edition applies to Version 4, Release 1, of CICS Transaction Server.

Contents

Notices

Preface

CICS® Explorer™ is the latest significant evolution in the management and analysis of your CICS environment. It is a statement of intent from the CICS Development organization, which is determined to ensure you can manage your CICS estate in a simple and easily extensible way, using a combination of the following approaches:

- Tried and trusted CICS expertise and technology
- The widely accepted user interfaces and integration power of the open source Eclipse platform
- Web 2.0 and RESTful programming (this technology underpins the CICS Explorer concept)

This IBM® Redbooks® publication shows how you can use the extensible design of CICS Explorer to complement the functionality already provided, with added functionality tailored to the needs of your business. We show you how to perform the following tasks:

- Install the CICS plug-in SDK into your eclipse environment
- Develop a simple plug-in for the CICS Explorer
- Deploy the plug-in into CICS Explorer

We provide several useful examples of plug-ins that we developed during the residency using the methodology we describe.

The starting point for the book is that you already have CICS Explorer installed and configured with connectivity to your CICS region or CICSPlex®, and that you are looking for ways to customize CICS Explorer.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Chris Rayns is an IT Specialist and the CICS project leader at the ITSO, Poughkeepsie Center. He writes extensively on all areas of CICS. Before joining the ITSO, Chris worked in IBM Global Services in the United Kingdom as a CICS IT Specialist.

Scott Clee is the CICS Test Architect for IBM in the United Kingdom. He frantically architects the face of test by looking for new ways to push Testing techniques, process, and technology. His areas of expertise include CICS, Java™, Common Business Oriented Language (COBOL), Linux®, and Testing. He has a degree in Mathematics and Computing from the University of Bath, UK. Check out his blog at TestingBlues.com.

Taku Miura is a Project Manager for ISOL (IBM Global Services Japan Solution and Services Company, Ltd.: a subsidiary of IBM Japan). He has 11 years of experience of software development and 3 years of project management, especially for mainframe system. He holds a degree in Productive System Engineering from Otaru University of Commerce, Hokkaido in Japan.

Shayla Robinson is a Staff Software Engineer in CPSM Level II Technical Support at Research Triangle Park, NC. Her areas of expertise include CICS, CICSPlex SM, z/VSE™ and Java. She has a Bachelors degree in Computer Science with a minor in Mathematics from Fayetteville State University in Fayetteville, NC and is currently working towards a Masters degree in Information Systems from Aspen University in Denver, CO.

Yusuke Tamura is an Assistant Manager for DIR (Daiwa Institute of Research Ltd (Japan)). Since joining DIR in 2003, he is involved in the design and development of self-developed software for online transaction processing. He holds a Bachelor of Arts in Law from the University of Tokyo.

Steve Wall is a Software Engineer in CICS Development at Hursley Park, UK. He has 13 years of experience as an Application Programmer, Tester, CICS Developer, and OMEGAMON® XE for CICS Developer with IBM. His areas of expertise include CICS, z/OS®, Java, OMEGAMON, and various forms of tooling. He has a Bachelors degree in Digital Systems Engineering from the University of the West of England, Bristol and a Masters degree in Software Engineering from the University of Oxford.

is an IT specialist working in the System z® Benchmark Center. He worked for the CICS Transaction Server Development organization at Hursley, United Kingdom, for over 20 years before joining the PSSC. Steve has a degree in Linguistic and International Studies from the University of Surrey. He has written and taught extensively about CICS e-business enablement using CICS Web Support and the CICS Transaction Gateway.

Thanks to the following people for their contributions to this project:

Richard M Conway
International Technical Support Organization, Poughkeepsie Center

John Knutson, System z Product Marketing Manager - CICS Tools
IBM Hursley

Joe Winchester, CICS Explorer, Interdependency Analyzer, Configuration Manager, Performance Analyzer Software Developer
IBM Hursley

Matthew Webster, CICS Tools UI Strategist: CICS Explorer
IBM Hursley

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Introduction

In part 1 we review the evolution of CICS management before taking a look at the new CICS Explorer. We provide an overview of CICS Explorer and Eclipse.

1

CICS Evolution

In this chapter we begin by taking a sideways look at the history of CICS System Management, and the benefits that CICS Explorer brings to the CICS systems management world. To develop new extensions to CICS Explorer, you need understanding of how Eclipse presents information, and how CICS Explorer uses the facilities that Eclipse provides. We therefore take a high-level overview of the external aspects of CICS Explorer and Eclipse that you need to understand to write your own CICS Explorer plug-ins.

1.1 Evolution of CICS system management

CICS Explorer is the latest significant evolution in the management and analysis of your CICS environment, but CICS has a long history of building on, and improving its system management tools.

CICS is now over 40 years old. Over the years, there have been significant changes in the way in which CICS is managed. Here are several of the important milestones in the evolution of CICS systems management.

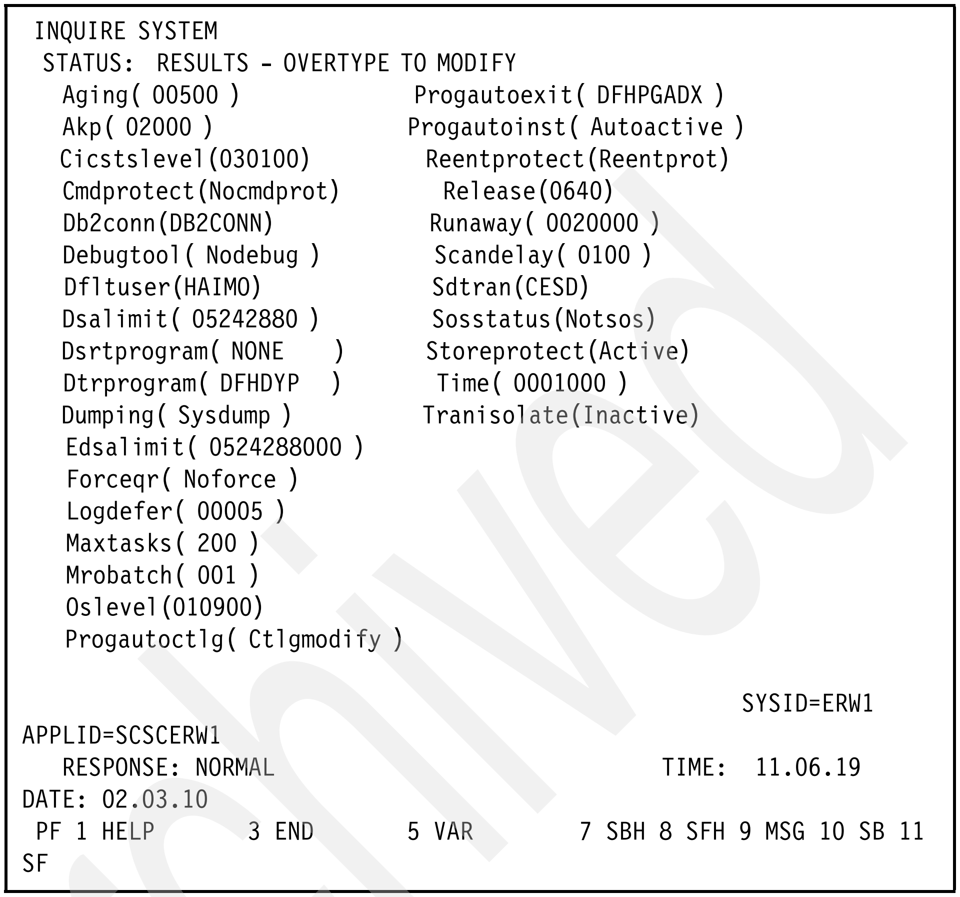
1.1.1 The dark ages: CSMT and Assembler macros

When CICS first appeared, there were no nice graphical interfaces. CSMT is the CICS transaction that was executed on these prehistoric devices. In these early days there was no such thing as "CEMT PERF SHUT". It was CSMT you used to close down your CICS region. If you try to run CSMT on your brand new CICSTS 4.1 region, it tells you that it is not defined. Dig around in the CICS-supplied CSD definition and you will find it. Today, the only thing that remains of transaction CSMT is the transient data queue to which many CICS messages are still routed.

As far as defining new CICS resources such as files, transactions, and programs, look no further than a set of CICS resource tables coded using Assembler macros. If you needed to install a new terminal, program, or transaction, you needed to restart your CICS.

1.1.2 The middle ages: CEMT and the first GUI

With the advent of 3270 green panel devices, we had the ability to format entire displays of data, and a new state of the art, full panel CICS management tool was born. CEMT arrived, with its state of the art graphics interface (See Figure 1-1 on page 5). With CEMT, the systems programmer can see the state of the CICS region, and the resources installed in that region.



```
 INQUIRE SYSTEM
  STATUS:  RESULTS - OVERTYPE TO MODIFY
    Aging( 00500 )              Progautoexit( DFHPGADX )
    Akp( 02000 )               Progautoinst( Autoactive )
    Cicstslevel(030100)         Reentprotect(Reentprot)
    Cmdprotect(Nocmdprot)        Release(0640)
    Db2conn(DB2CONN)            Runaway( 0020000 )
    Debugtool( Nodebug )        Scandelay( 0100 )
    Dfltuser(HAIMO)             Sdtran(CESD)
    Dsalimit( 05242880 )        Sosstatus(Notsos)
    Dsrtprogram( NONE    )      Storeprotect(Active)
    Dtrprogram( DFHDYP   )       Time( 0001000 )
    Dumping( Sysdump )          Tranisolate(Inactive)
    Edsalimit( 0524288000 )
    Forceqr( Noforce )
    Logdefer( 00005 )
    Maxtasks( 200 )
    Mrobatch( 001 )
    Oslevel(010900)
    Progautoctlg( Ctlgmodify )

                                                    SYSID=ERW1
APPLID=SCSCERW1
   RESPONSE: NORMAL                              TIME:  11.06.19
DATE: 02.03.10
 PF 1 HELP       3 END       5 VAR       7 SBH 8 SFH 9 MSG 10 SB 11
SF
```

Figure 1-1 Graphics interface

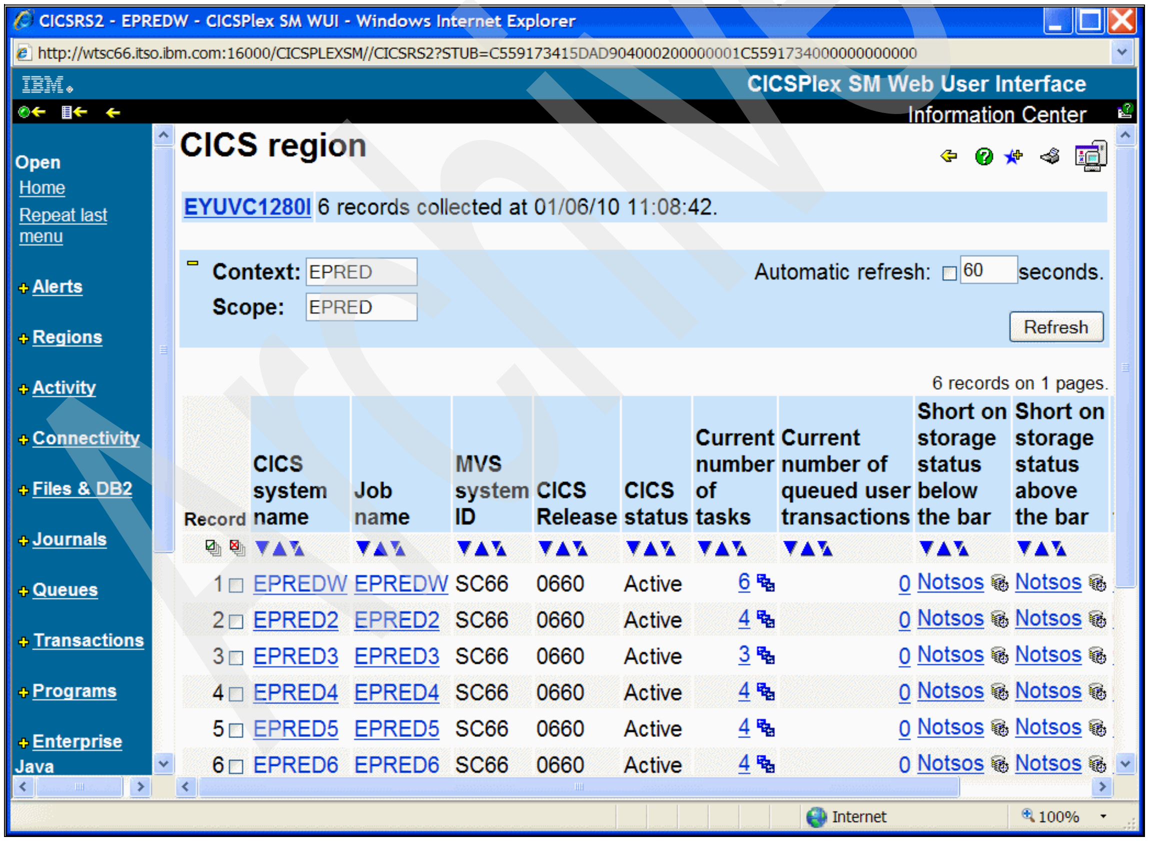
1.1.3 The renaissance: CEDA and CICS System Definition

CEMT was a great tool, but we were still obliged to assemble and compile our DFHPCT, DFHPPT, DFHFCT, and DFHTCT macros (to mention but a few) to modify or install new resources into our CICS region. The next great leap forward in CICS systems management was the CICS System Definition (CSD) and a new CICS-supplied transaction CEDA. Now we can add new files, programs, transactions, and terminals to our CICS region with a few keystrokes, and without having to stop and restart our CICS region.

1.1.4 The enlightenment: CICSPlex SM

CEDA and CEMT made the job of the CICS system programmer easier, but as the size of customer CICS installations grew, it became evident that a new approach was needed. To execute CEDA or CEMT in each of tens or hundreds of CICS regions was a logistical nightmare. The introduction of CICSPlex SM, with its new APIs and system management tools, allowed system administrators to group together CICS regions according to their particular requirements, and manage those groups of CICS regions as single entities, installing, controlling, and monitoring those CICS regions without the need to log on to each CICS region simultaneously.

CICSPlex SM used the advanced features of the latest 3270 terminal technology to bring a rich user interface to the management of complex CICS environments. With the advent of the Internet and the Web, the 3270 CICSPlex SM interface has been superseded by a Web Browser interface called the Web User Interface. Figure 1-2 shows a CICSPlex SM WUI screen showing CICS regions on MVS image SC66.


CICSRS2 - EPREDW - CICSPlex SM WUI - Windows Internet Explorer
http://wtsc66.itso.ibm.com:16000/CICSPLEXSM//CICSRS2?STUB=C559173415DAD904000200000001C5591734000000000000
IBM.
CICSPlex SM Web User Interface
Information Center
Open
Home
Repeat last menu
Alerts
Regions
Activity
Connectivity
Files & DB2
Journals
Queues
Transactions
Programs
Enterprise Java
Refresh
Internet
100%

CICS region

EYUVC1280I 6 records collected at 01/06/10 11:08:42.

Context: EPRED
Scope: EPRED
Automatic refresh: 60 seconds.

6 records on 1 pages.

Record	CICS system name	Job name	MVS system ID	CICS Release	CICS status	Current number of tasks	Current number of queued user transactions	Short on storage status below the bar	Short on storage status above the bar
1	EPREDW	EPREDW	SC66	0660	Active	6	0	Notsos	Notsos
2	EPRED2	EPRED2	SC66	0660	Active	4	0	Notsos	Notsos
3	EPRED3	EPRED3	SC66	0660	Active	3	0	Notsos	Notsos
4	EPRED4	EPRED4	SC66	0660	Active	4	0	Notsos	Notsos
5	EPRED5	EPRED5	SC66	0660	Active	4	0	Notsos	Notsos
6	EPRED6	EPRED6	SC66	0660	Active	4	0	Notsos	Notsos

Figure 1-2 CICSPlex SM WUI panel

Although the face of CICSPlex SM changed with the advent of the WUI, the underlying philosophy of CICSPlex SM is probably more important today than when it was first conceived. It remains a key element in the management of CICSPlexes big and small across the world. We see that CICS Explorer relies on CICSPlex SM to provide it with timely and accurate information about the state of the CICS resources it manages.

1.1.5 The baby boom: The CICS Tools Suite

In recent years, the CICS tools toolset has significantly improved the facilities at the disposal of the CICS System Administrator, allowing them to simplify the management and monitoring of their CICS real estate, and giving them the ability to extract the underlying structure of their CICS applications even though the programmers, the documentation, and sometimes the source code, of those applications are long gone. CICS configuration manager took the best of CEDA and CICSPlex SM resource management and created something simpler and easier. CICS interdependencies analyzer allowed administrators to understand the implications of proposed changes to their application suites. CICS Performance Analyzer gave new insight into the performance of those applications. However, each of the tools had its own user interface, its own set of panels. CICS Explorer brings a common user interface and user experience to the whole CICS toolset in a single (free) product. Do not forget the Tivoli® Omegamon family of products, which also contain a rich set of functions for the management of CICS resources.

1.1.6 The brave new world: CICS Explorer

All the systems management tools described previously have their own separate user interfaces - CICS 3270 transactions, Web browser, TSO ISPF panels, batch jobs. When you consider that there is also a whole host of third-party, vendor, or customer home-grown CICS systems management tools, each with their own user interface and implementation, you can see why CICS system administrators and programmers tend to regard with suspicion yet another tool with yet another user interface to learn.

CICS Explorer signals an end to this confusion. It allows all the tools and applications that a user needs to perform a business task to present the same structure and user interface. Tools that are built by IBM, vendors, and customers extend and enhance CICS Explorer in a consistent way, adding value to CICS management without the need to reinvent or redefine CICS Explorer's user interface.

CICS Explorer has achieved this by embracing the Eclipse open source community's extensible development platform, runtimes, and application frameworks for building, deploying, and managing software.

1.2 Overview of CICS Explorer

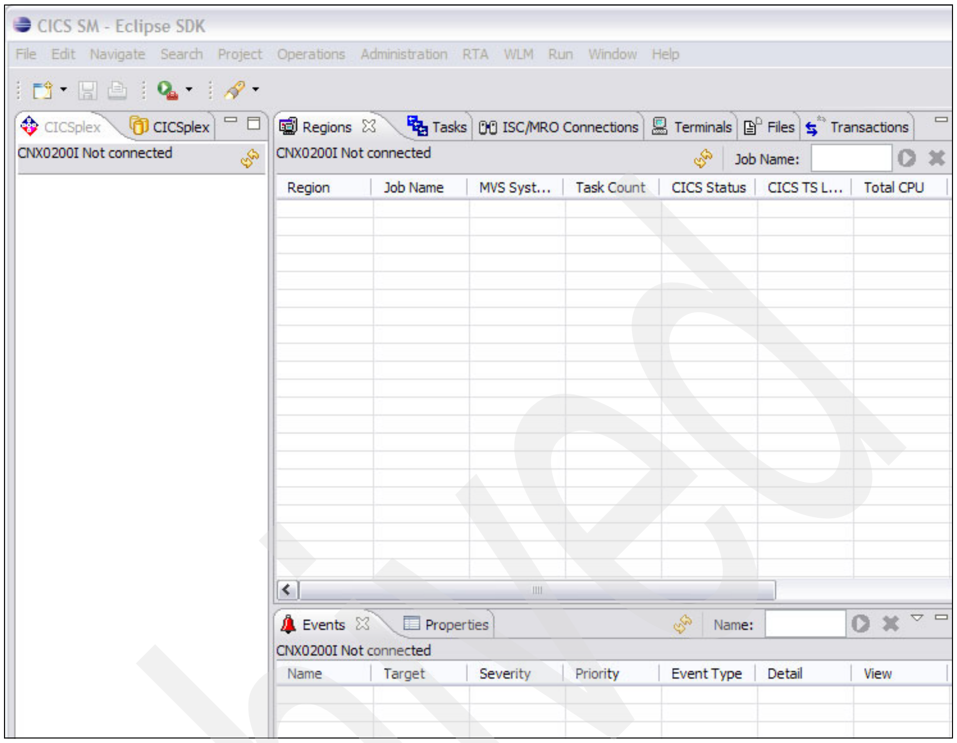
It is worth spending a little time looking at the Eclipse user interfaces, and how they are used by CICS Explorer. The better you understand these interfaces, the easier it is for you to create useful new CICS Explorer plug-ins.

Note: To learn in detail about CICS Explorer and how you can use and customize it without the need for programming, see IBM Redbooks publication *CICS Explorer*, SG24-7778.

The intention here is to focus on the elements and concepts of Eclipse, which are going to be interesting from a plug-in programming point of view. We can break the Eclipse display down into a number of constituent parts.

1.2.1 Workbenches and Perspectives

When you start your CICS Explorer for the first time, you are presented with the the CICS Explorer Workbench, shown in Figure 1-3 on page 9.



CNX0200I Not connected

Region	Job Name	MVS Syst...	Task Count	CICS Status	CICS TS L...	Total CPU

Name	Target	Severity	Priority	Event Type	Detail	View

Figure 1-3 CICS Explorer Workbench CICS SM perspective

Each Workbench window contains one or more perspectives. In Figure 1-3, we see the default perspective for the CICS Explorer, provided by the CICS Explorer. Think of an Eclipse Perspective as being a set of related tools grouped together for a specific task or role. For example, the perspective used by a CICS Systems Programmer is different from that used by a COBOL Application Developer, a Java Application Developer, or a Web designer.

If you click **Window** → **Open Perspective** → **Other** you can see the list of perspectives supplied with this version of CICS Explorer. The Resource perspective is also provided by CICS Explorer. All the other perspectives you see come with Eclipse RCP. In this book we look at and mostly work with the CICS SM perspective. The Resource perspective is used to create and manage CICS Event bindings and objects. Perspectives are a powerful way of creating displays that are customized to show the tasks and information required for a specific job or role (with no need for programming).

1.2.2 Views

Think of views as the primary Eclipse interface for displaying the information you want to see. A perspective can be made up of many views, or of a single view.

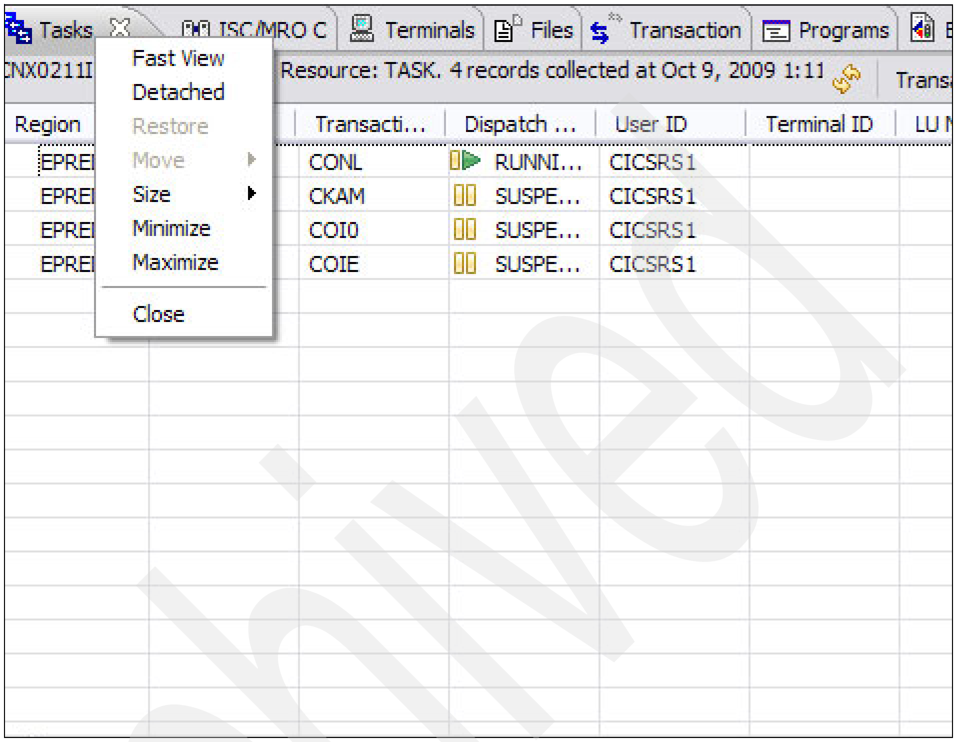
Tip: If you are in a perspective with multiple views, and you want to concentrate on a single view, double-click that view to get it in full-panel mode. After you have finished, double-click its tab again, and you return to the original perspective.

Views can occur singly, or in what is called a *tab group*, or *tabbed notebook*. For example, if you look at the default CICS SM perspective, you can see that the welcome view is on the right hand side on its own. On the left, you can see that there is a tab group made up of the CICSPlex Explorer and CICSPlex Repository views. In the center, there is another tab group, which contains views for CICS resources managed by CICS Explorer. Add a new view of another CICS resource to the central tabbed notebook by simply clicking (for example) **Operations → Web Services**.

If you click the CICS regions listed in the CICSPlex Explorer view, the resources displayed in the upper central tab group are changed to reflect that you are now viewing the data associated with that region.

As a general rule, when using Eclipse, the left tab group to navigate around the information available in the workspace, and the information you are interested appears in the lower central tab group. This is a general rule that applies to most Eclipse RCP implementations. Similarly, there is usually a tab group in the lower central portion of the workspace, which tends to be used to signal problems, errors, or (in the case of our CICS SM view) events that might present a potential problem. The default view in this panel is for CICSPlex SM alerts. If one of our CICS Regions raises an alert (for example, goes short on storage, or reaches maxtask), the appropriate alert is displayed in this view.

Each view can have a menu of related actions or views associated with it. You display this menu by right-clicking the view tab (see Figure 1-4).



CNX0211I Resource: TASK. 4 records collected at Oct 9, 2009 1:11

Region	Transacti...	Dispatch ...	User ID	Terminal ID	LU N
EPREI	CONL	RUNNI...	CICSRS1		
EPREI	CKAM	SUSPE...	CICSRS1		
EPREI	COI0	SUSPE...	CICSRS1		
EPREI	COIE	SUSPE...	CICSRS1		

Figure 1-4 View tab menu

The actions we see in the Tasks view are the defaults provided by Eclipse. There are no CICS-specific ones unless we choose to add a plug-in that adds an action we want to implement.

1.2.3 Menus and the menu bar

Each perspective has a menu bar, which has been designed and customized to detail a menu of actions or views that are related to a specific menu item. So, for example, the menu bar of the CICS SM perspective looks like Figure 1-5.

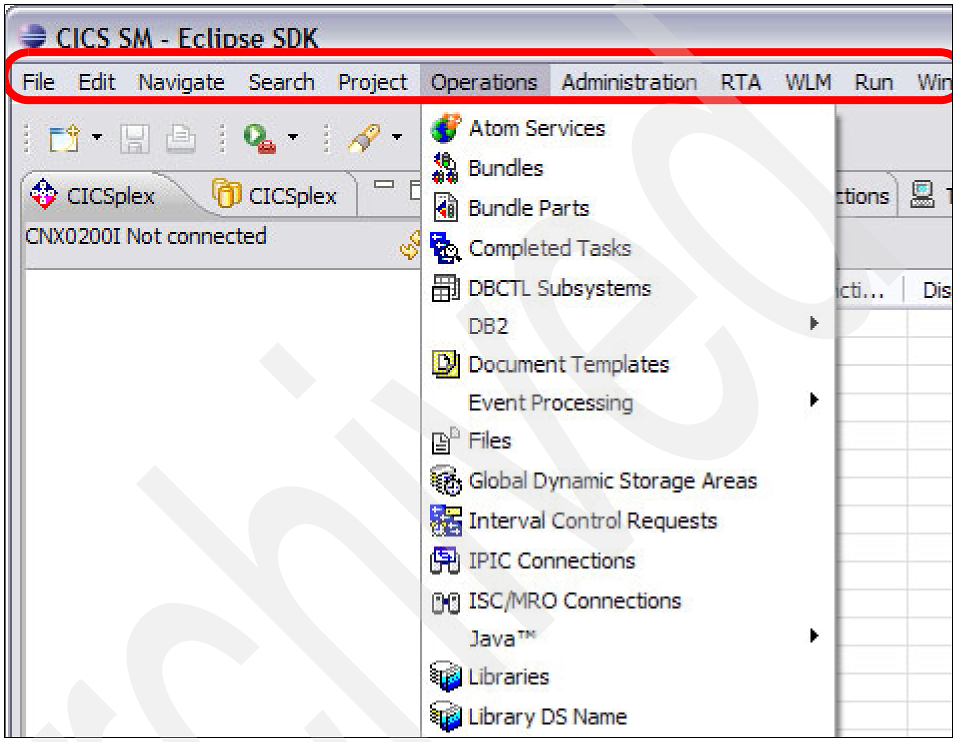


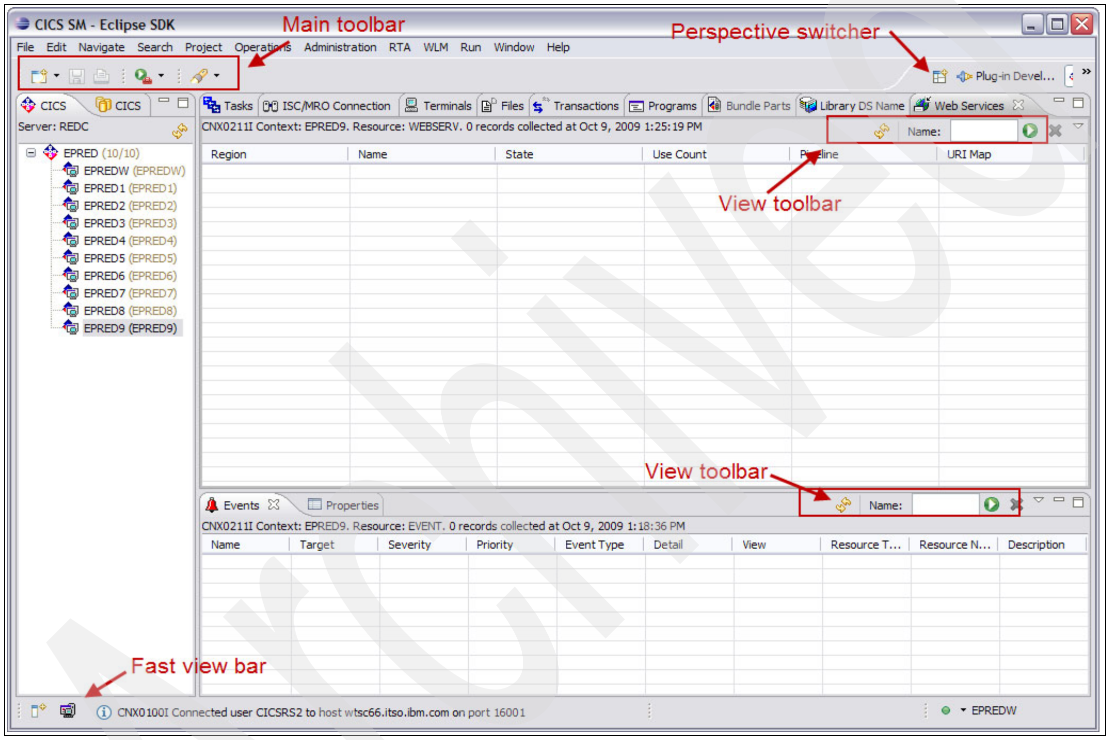
Figure 1-5 CICS SM perspective menu bar

As you click each item in the menu bar, a list of actions or views associated with that item is presented, and you can select the one you want. You can see that several of the menu items, such as File, or Edit are standard Eclipse user interface menu items, but others, such as WLM, RTA, Operations, are provided by plug-ins provided by CICS Explorer. If you want to add new menus that are appropriate to your installation, write a plug-in to do it. Later we show how you can create your own menus that you can add to a new or existing perspective's menu bar.

1.2.4 Toolbars

A toolbar is a collection of graphical icons that when clicked, typically execute a wizard associated with the task represented by the icon, or when right-clicked, display a menu of actions or views associated with that icon.

There are five separate kinds of toolbars in Eclipse, so when writing a new plug-in for a new tool, you need to give thought to the best place to invoke that new plug-in. Figure 1-6 shows the kinds of toolbars in Eclipse.


CICS SM - Eclipse SDK
Main toolbar
Perspective switcher
File Edit Navigate Search Project Operations Administration RTA WLM Run Window Help
Plug-in Devel...
CICS CICS
Tasks ISC/MRO Connection Terminals Files Transactions Programs Bundle Parts Library DS Name Web Services
Server: REDC
EPRED (10/10)
EPREDW (EPREDW)
EPRED1 (EPRED1)
EPRED2 (EPRED2)
EPRED3 (EPRED3)
EPRED4 (EPRED4)
EPRED5 (EPRED5)
EPRED6 (EPRED6)
EPRED7 (EPRED7)
EPRED8 (EPRED8)
EPRED9 (EPRED9)
Name:
Region Name State Use Count Pipeline URI Map
View toolbar
View toolbar
Events Properties
Name Target Severity Priority Event Type Detail View Resource T... Resource N... Description
Fast view bar
EPREDW

CNX0211I Context: EPRED9. Resource: WEBSERV. 0 records collected at Oct 9, 2009 1:25:19 PM

CNX0211I Context: EPRED9. Resource: EVENT. 0 records collected at Oct 9, 2009 1:18:36 PM

CNX0100I Connected user CICSRS2 to host wtsc66.itso.ibm.com on port 16001

Figure 1-6 Eclipse toolbars

The main toolbar (sometimes called the *workspace toolbar*) is associated with the perspective. By default, when you define a new item on the menu bar (called an action set, the Eclipse SDK generates a corresponding icon on the toolbar to graphically represent the item.

A new plug-in that defines an action or view associated to a particular view can be defined to be displayed in a view toolbar.

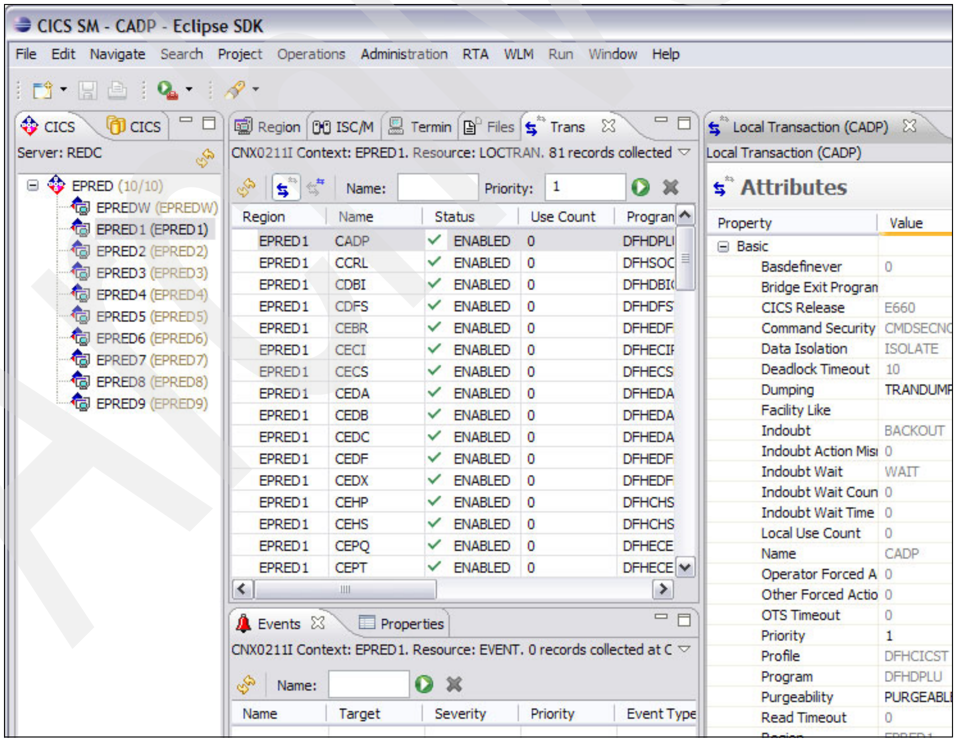
The perspective switcher is a tool that allows you to move quickly from one perspective to another, with each open perspective represented by an icon.

To get instant access to your most commonly used views, add them to the fast view toolbar by right-clicking the View tab and selecting **Fast View**.

1.2.5 Editors

When you want to define or modify Eclipse resources, depending on the resource, Eclipse opens an appropriate editor in the Editor Area (usually the center panel in the workspace). If the Eclipse resource is a plain text file, it is an editor that works with plain text. If it is an XML file, Eclipse contains an editor with functionality for working with XML data (such as syntax checking). As you develop new plug-ins, you might need to invoke an editor.

In the case of the CICS Explorer, the CICS Explorer has a specific editors for working with CICS resources. If you double-click a CICS resource (for example a CICS region), CICS Explorer opens an editor session to the right of the current view, showing the attributes for that resource. This is shown in Figure 1-7.


CICS SM - CADP - Eclipse SDK
File Edit Navigate Search Project Operations Administration RTA WLM Run Window Help
CICS
CICS
Region
ISC/M
Termin
Files
Trans
Local Transaction (CADP)
Events
Properties

Server: REDC

- EPRED (10/10)
  - EPREDW (EPREDW)
  - EPRED1 (EPRED1)
  - EPRED2 (EPRED2)
  - EPRED3 (EPRED3)
  - EPRED4 (EPRED4)
  - EPRED5 (EPRED5)
  - EPRED6 (EPRED6)
  - EPRED7 (EPRED7)
  - EPRED8 (EPRED8)
  - EPRED9 (EPRED9)

CNX0211I Context: EPRED1. Resource: LOCTRAN. 81 records collected

Name: Priority: 1

Region	Name	Status	Use Count	Progran
EPRED1	CADP	✓ ENABLED	0	DFHDPLI
EPRED1	CCRL	✓ ENABLED	0	DFHSOC
EPRED1	CDBI	✓ ENABLED	0	DFHDBI(
EPRED1	CDFS	✓ ENABLED	0	DFHDFS
EPRED1	CEBR	✓ ENABLED	0	DFHEDF
EPRED1	CECI	✓ ENABLED	0	DFHECIF
EPRED1	CECS	✓ ENABLED	0	DFHECS
EPRED1	CEDA	✓ ENABLED	0	DFHEDA
EPRED1	CEDB	✓ ENABLED	0	DFHEDA
EPRED1	CEDC	✓ ENABLED	0	DFHEDA
EPRED1	CEDF	✓ ENABLED	0	DFHEDF
EPRED1	CEDX	✓ ENABLED	0	DFHEDF
EPRED1	CEHP	✓ ENABLED	0	DFHCHS
EPRED1	CEHS	✓ ENABLED	0	DFHCHS
EPRED1	CEPQ	✓ ENABLED	0	DFHECE
EPRED1	CEPT	✓ ENABLED	0	DFHECE

CNX0211I Context: EPRED1. Resource: EVENT. 0 records collected at C

Name:

Name	Target	Severity	Priority	Event Type

Local Transaction (CADP)

Attributes

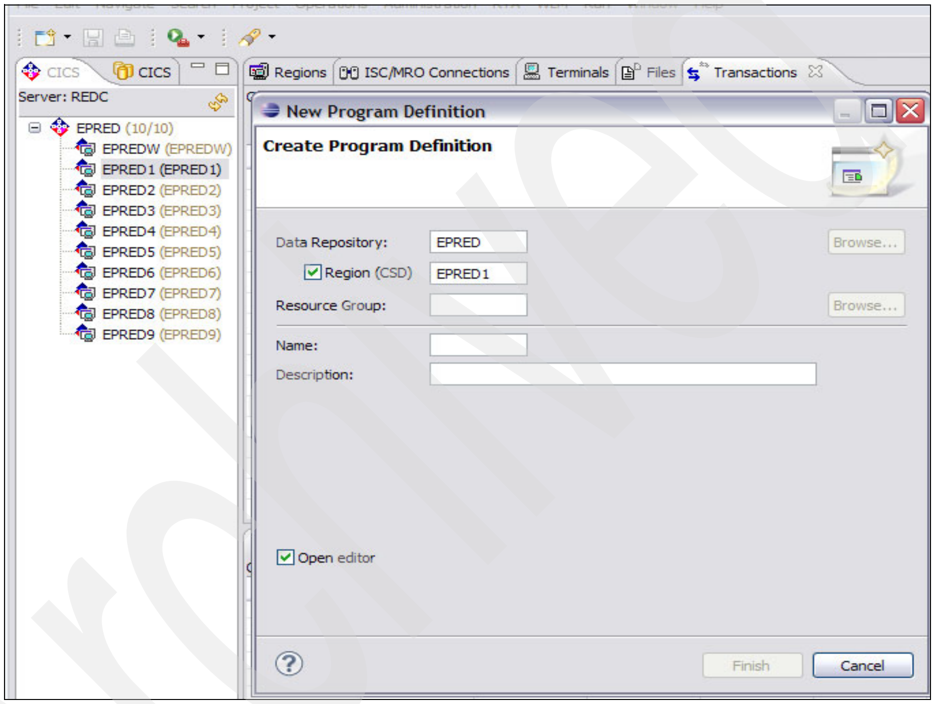
Property	Value
Basic	
Basdefinever	0
Bridge Exit Progran	
CICS Release	E660
Command Security	CMDSECNC
Data Isolation	ISOLATE
Deadlock Timeout	10
Dumping	TRANDUMP
Facility Like	
Indoubt	BACKOUT
Indoubt Action Mis	0
Indoubt Wait	WAIT
Indoubt Wait Coun	0
Indoubt Wait Time	0
Local Use Count	0
Name	CADP
Operator Forced A	0
Other Forced Actio	0
OTS Timeout	0
Priority	1
Profile	DFHCICST
Program	DFHDPLU
Purgeability	PURGEABL
Read Timeout	0

Figure 1-7 Double-click region object to invoke CICS Editor to display attributes

1.2.6 Wizards

When you choose to create or modify a CICS resource, a wizard opens in a separate pane, to the right of the tabular views. The wizard guides you through the creation of the new resource, prompting you for the correct information.

For example, if you want to create a new Program definition for a CICS region, you can right-click that region, and select **New** → **Program Definition** and a wizard opens to allow you to provide the necessary information. See Figure 1-8.



Create Program Definition

Data Repository: EPRED

[✓] Region (CSD) EPRED1

Resource Group:

Name:

Description:

[✓] Open editor

Figure 1-8 CICS Explorer CICS Resource Editor

There might be scenarios where you want to do something similar for your own business needs. and you can do this by writing your own plug-in.

1.3 Overview of Eclipse

Because CICS Explorer is based on Eclipse, it is important to have idea of the relationship between the two. In this section we take a high-level view of aspects of Eclipse of which we need to be aware.

The Eclipse community has several separate ongoing focuses of activity. The one on which CICS Explorer is based is called *Eclipse Rich Client Platform* (RCP).

A rich client is a fat client. That means we have a fat, rich client. That is to say, it is an application that uses the windowing and GUI features of the operating system (for example, native widgets, drag and drop) and is integrated with the operating system's component model (for example, ActiveX). This is what we are going to focus on.

Figure 1-9 is a description of Eclipse RCP taken from the following Web page:

http://www.eclipse.org/downloads/download.php?file=/technology/phoenix/talks/What-is-Eclipse-and-Eclipse-RCP-3.2.6.ppt

```
A consistent and native look and feel across applications and features
Provides common application services
Native look and feel
Window management
Standardized component model
Pervasive extensibility
Update Manager
Help system
First-class development tools
Middleware for building rich client applications
Allows programmers to focus on core application not the plumbing
Don't reinvent the wheel
```

Figure 1-9 Eclipse description

Sounds like a good fit with the aims of CICS Explorer? That is what the CICS development organization thought. Eclipse RCP provides a set of base functions and services that handle, among other things, user interfaces, help panels, and update management, as shown in Figure 1-10.

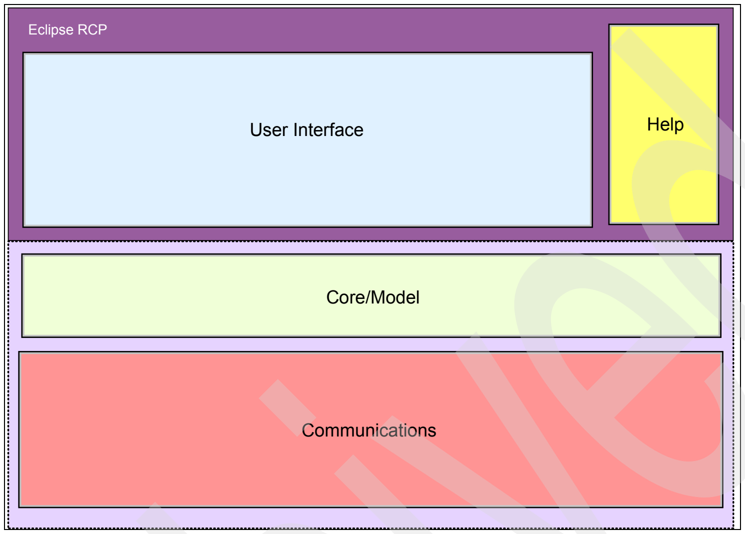


Figure 1-10 Eclipse RCP structure

The base functions provided by Eclipse RCP are designed to be extensible, and available for use by new components (plug-ins) that implement functionality specific to a particular technical or business need. The plug-in developers use the provided interfaces to write their own components, which can provide functionality used by other plug-ins, as shown in Figure 1-11.

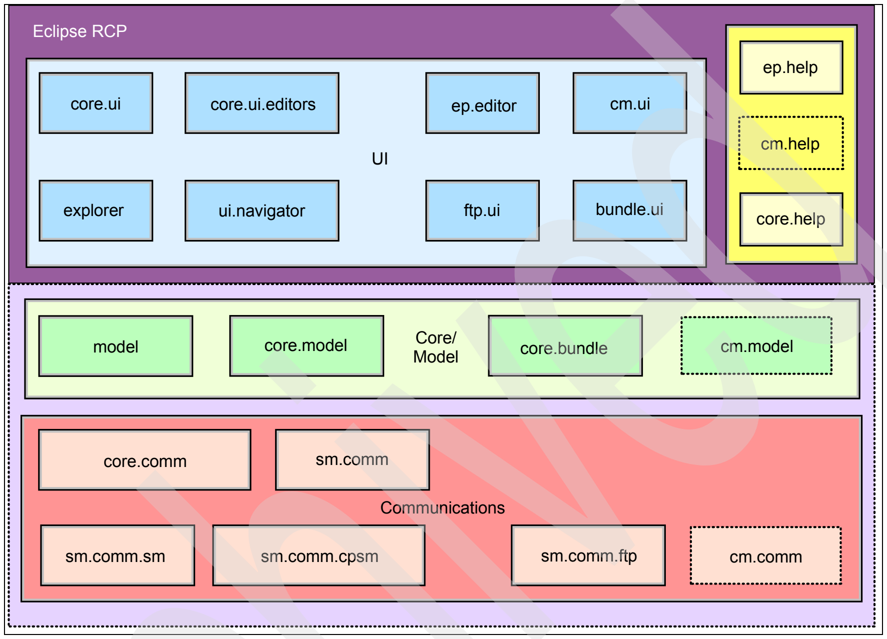


Figure 1-11 Eclipse RCP component model

To summarize, Eclipse RCP is a runtime environment providing a set of commonly used services for use by components (plug-ins) developed to run inside that runtime environment (sound familiar?).

Where it is necessary to connect to a remote system to retrieve information, a plug-in can use the communications facilities provided by Eclipse, or implement its own communication services. This might be made available to other plug-ins, as shown in Figure 1-12.

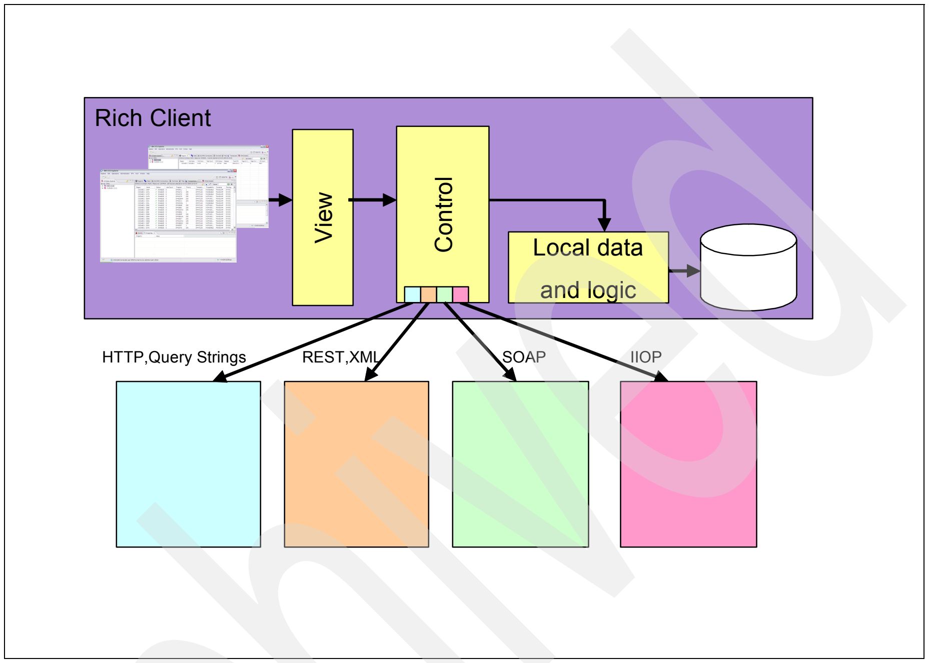


Figure 1-12 Eclipse RCP Connectivity

As we see in more detail later, CICS Explorer uses the communications facilities provided by Eclipse RCP to connect to CICS TS to retrieve and modify the data it uses. The list of protocols to be used is not exclusive, you simply write a plug-in to implement the transport protocol you need to use.

As well as the Eclipse RCP run-time itself, Eclipse provides a development environment specifically for plug-in development. This development environment is called the Eclipse plug-in SDK. It is also referred to as a Plug-in Development Environment (PDE). We are using the Eclipse plug-in SDK to develop and test our CICS Explorer plug-ins in this Redbooks publication.

When developing Eclipse plug-ins, it is important to remember that you usually deal with two separate Eclipse environments:

- The Eclipse plug-in SDK, where you develop and test your plug-in (it contains a runtime environment to allow you to test).
- The target Eclipse environment, where you deploy the finished (and tested) Eclipse RCP application. In our case the target environment is the CICS Explorer runtime.

As a general rule, the default active view (the one you can see) on the left contains a list of resources related to the perspective we are using. In this case, the default view shown in pane 1 is a list of the plug-in projects that exist in the Eclipse workspace in which you are working. Because you have not yet created any plug-ins, the pane is blank. However, if we click the Plug-ins tab, to bring that view to the front, we can see all the plug-ins that come with the Eclipse SDK.

When working with the CICS Explorer, it is important to remember that it is still a work in progress. However, it is seen by the CICS development organization as a critical element in the future of CICS. The plan is to roll out new versions of CICS Explorer, with new functionality, several times per year.

We have already seen significant progress since the CICS Explorer first appeared. In its original form, CICS Explorer is not able to update CICS resources, only to view them. New in CICS TS version 4 was the ability to modify and create CICS resources. Originally, you could only connect to a CICSPlex SM WUI CICS region, but you can now connect to a single standalone CICS region.

We see in the course of this book that there is already a rich set of functions in CICS Explorer, which are there to be used in new and innovative ways. We hope that after you have read this book, and tried the samples, you will share your ideas for new ways to use CICS Explorer with the wider CICS community through the CICS Explorer forum at the following Web page:

`http://www.developerworks/forums/forum.jspa?forumID=1475`

If you have any questions about CICS Explorer, or about the topics discussed in this book, you can post them on the forum.

Part 2

Exploring CICS Explorer

In this part of this Redbooks publication we show how CICS Explorer uses the CICS Client Management Interface (CMCI). CMCI is the strategic interface for CICS Explorer communications with CICS TS. We also we take an in depth look at the CICS Explorer SDK.

2

CICS Explorer and the CICS Client Management Interface

In this chapter we take a more detailed look at the way CICS Explorer uses the CICS Client Management Interface (CMCI). CMCI is the strategic interface for CICS Explorer communications with CICS TS.

If you are simply using CICS Explorer as a user, the way CICS Explorer uses CMCI is transparent to you.

If you plan to write additional functions to run inside CICS Explorer, you need to understand how CICS Explorer uses the CMCI to create efficient new CICS Explorer plug-ins, how it communicates with CICS, how it stores information about the CICS environment, and how it is possible to manipulate the data that CICS Explorer uses.

Note: CICS Explorer initially used a separate (non-RESTful) implementation of HTTP to communicate with a CICS WUI region over TCP/IP, but CMCI is the strategic interface, so it is what we focus on here.

We also compare the CICS Explorer with the Web user interface to understand how they differ, and why CICS Explorer is considered to be a better solution.

2.1 CMCI

CMCI allows anyone to develop HTTP client applications that manage, install, and define CICS resources. CMCI is a simple but extremely powerful tool. The biggest user of the CMCI today is CICS Explorer.

CMCI is CICSPlex SM code. It runs either in a CICSPlex SM WUI server, or in a stand-alone CICS region (in which case it is known as Systems Management Single Server [SMSS]). Like CICS Explorer and CICSplex SM, you can divide CICS Explorer activities into two categories:

- Operational activities

  Enabling, disabling, or modifying installed resources, or system settings (analogous to CEMT).

- Administration activities

  Defining, installing, modifying or deleting CICS resources (analogous to CEDA or CICSPlex SM's Business Application Services (BAS)). When used in CICSPlex mode CMCI works with CICS resources managed by BAS; when used in a single standalone region it works with the CICS CSD. The repository used for the resources being managed is completely transparent to the user.

For a detailed description of the CMCI implementation and the API, enter the search argument "CMCI" at the following Web page:

`http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp`

2.1.1 CMCI is a RESTful implementation

CMCI is a new system management API, based on Representational State Transfer (RESTful) principles. The speed and flexibility of CICS Explorer are derived largely from CMCI's RESTful implementation. A RESTful application makes use of the following methods (as defined in the HTTP Protocol) to define actions that can be performed (in the case of CMCI, on a CICS resource):

- `POST`

  Create resources on the data repository.

- `GET`

  Retrieve information about resources.

- `PUT`

  Update existing resources in the data repository, set attributes, or perform actions on installed resources.

- `DELETE`

  Remove resources from the data repository or discard installed resources.

The key factor in a RESTful implementation is efficient use of the HTTP methods, URI strings, and query strings.

Note: REST stands for Representational State Transfer. This is not the most informative acronym, and this book is not the place for a lengthy discussion of REST. For the official description, and useful background, try the following Web pages:

- `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`
- `http://www.ibm.com/developerworks/web/library/wa-ajaxarch/`
- `http://www.surfscranton.com/architecture`
- `http://rest.blueoxen.net/cgi-bin/wiki.pl?whatIsREST`

2.1.2 CMCI and CICSPlex SM resource tables

Almost all CICS and CICSPlex SM resources that are managed by CICSPlex SM are described and managed by means of a set of resource tables. CMCI uses these resource tables. The CICSPlex SM resource table architecture has been extended to allow it to work with the CMCI's RESTful approach:

- Each resource type now has a new external resource name section. This external name is the one specified by the CMCI client to select that resource type. For example, the LOCFILE CICSPlex SM resource type (local VSAM file) is defined with an external resource name of CICSLocalFile.
- Each resource type now has a valid CICS Management Client Interface HTTP methods section, which defines the HTTP methods valid for the resource.
- The error codes returned by CICSPlex SM when accessing resource tables have been re-ordered to make things simpler for CMCI users

2.2 CMCI caching

CMCI uses caching technology to provide faster access to data and reduce the bandwidth and network latency associated with CMCI requests. The CMCI client (CICS Explorer in our case) chooses to use this caching by specifying the NODISCARD=NODISCARD parameter on the query string, which it uses to articulate its CMCI request to CMCI. Read the sections that follow, which provide examples of CMCI flows, and accessing the cache, to see how it is used. The default behavior of CICS Explorer is to use caching.

Note: Cached results are stored in the CICS WUI region in its above the bar domain storage pool WU_64, so make sure you set MEMLIMIT and the EDSA limit carefully

Use both indexing and ranges to target your requests for cached data. For example, to show the first five resource records from a previously created cache, starting at record nine, the URI query string looks like this:

```
../CICSResultCache/C46DB57FAE2D8F64/9/5?NODISCARD
```

Where the string C46DB57FAE2D8F64 identifies a cache created earlier (see the examples of the flows for separate activities).

You can also use the range value when creating the cache, to limit the cache to the first ten records. The URI you submit looks like this:

```
/CICSSystemManagement/CICSProgram/EPRED///10?NODISCARD=NODISCARD&SUMMONLY
```

2.2.1 Defining a CICS resource definition with the CMCI

The resource upon which the action is to be performed is specified in the URI of the HTTP request received by CMCI. So for example, if we look at the creation of a URIMAP, with name MYURI, in BAS, we see the request flow in from CICS Explorer over the CMCI, as shown in Example 2-1.

Example 2-1 HTTP request for to define a CICS resource using CMCI

```
POST /CICSSystemManagement/CICSDefinitionURIMap/EPRED HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: wtsc66.itso.ibm.com:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 141
```

```
<request>
   <create>
     <attributes USAGE="PIPELINE" PIPELINE="MYPIPE" PATH="/path"
HOST="host.ibm.com" NAME="MYURI" DEFVER="1"/>
   </create>
</request>
```

Because we are creating a URIMAP resource, we use the HTTP verb POST. The resource we are working with is a URIMAP definition so the URI specifies:

```
/CICSSystemManagement/CICSDefinitionURIMAP/EPRED
```

The EPRED at the end of the URI specifies the CICSPlex SM context (in our case this resource is associated with the EPRED CICSplex). The body of the HTTP request contains the XML data providing the attributes of the object we want to create.

If we look at the HTTP response to our create request returned by the CICS WUI region we see the information shown in Example 2-2.

Example 2-2   CMCI HTTP Response for defining a URIMAP

```
HTTP/1.1 200 OK..Cache-Control: no-store
Date: Wed, 14 Oct 2009 12:53:40 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive

147
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int

http://wtsc66.itso.ibm.com:16001/CICSSystemManagement/schema/CICSSystem
Management.xsd"
          version="1.0"
          connect_version="0410">
<resultsummary
      api_response1="1024"
      api_response2="0"
      api_response1_alt="OK"
      api_response2_alt=""
      recordcount="1"
      displayed_recordcount="1" />
<records>
2A4
<cicsdefinitionurimap _keydata="D4E8E4D9C9404040014040404040404040"
                      analyzer="NO" atomservice="" authenticate="NO"
                      certificate=""
                      changeagent="DREPAPI" changeagrel="0660"
changetime="2009-10-14T08:53:40"
                      changeusrid="CICSRS1" characterset="" ciphers=""
converter=""
                      createtime="2009-10-14T08:53:40" csdgroup=""
defver="1" desccodepage="37"
                      description="" hfsfile="" host="host.ibm.com"
hostcodepage=""
                      location="" mediatype="" name="MYURI"
path="/path" pipeline="MYPIPE"
                      port="NO" program="" redirecttype="NONE"
scheme="HTTP" status="ENABLED"
                      tcpipservice="" templatename="" transaction=""
usage="PIPELINE"
                      userdata1="" userdata2="" userdata3="" userid=""
webservice="" />
D
</records>

B
</response>..
```

We can see that the XML schemas for the CICS XML data are specified on the response tag, and can be retrieved for the CICS WUI region if required.

The attributes of the `resultsummary` tag show us whether our create was successful or not. `api_response1="1024"` means the operation was successful. `api_response1_alt="OK"` is the response code in human-readable form. The number of new CICS Definition records created is also returned (`recordcount=1`).

The record element gives us the details of all the new resources created. Each record's first attribute is `_keydata`. This is a value that uniquely identifies this record. It is also used to link records to feedback. In our case, we have a `cicsdefinitionurimap` with all the attributes of our newly created URIMAP. The records element provides a fully externalized interpretation of almost all CICS resources and their attributes.

Note: If you look at a CICS trace of the CMCI flows, you see that the various parts of both the HTTP request and the HTTP response are split across a number of Sockets domain SEND and RECEIVE calls.

2.2.2 Viewing a CICS resource definition using CMCI

The HTTP verb used for inquiring on a resource is GET. As with all CMCI calls, the resource upon which the action is to be performed is specified in the URI of the HTTP request received by CMCI. See Example 2-3.

Example 2-3 Get request for CMCI view

```
GET
/CICSSystemManagement/CICSDefinitionURIMap/EPRED?NODISCARD=NODISCARD&SUMMONLY&C
RITERIA=((NAME='MYURI')) HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

We see in Example 2-3 that the URI identifies the type of resource. The query string `NODISCARD=NODISCARD&SUMMONLY&CRITERIA=((NAME='MYURI'))` identifies the name of the resource we want to access (MYURI), and that we only want a summary of its associated data. The `NODISCARD` name/value pair tells CMCI to cache the information we are retrieving. The default behaviour of Explore is to always request that retrieve data be cached by CMCI. If we look at the HTTP response returned by CMCI (Example 2-4), we see that the inquire was successful, but that CMCI has not returned the data for the inquire; instead it has returned a `cachetoken` which can be saved and used to request information about this and perhaps other URIMAPs.

Example 2-4 HTTP response returned by CMCI

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:20 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive

<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
```

```
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement.xsd"
version="1.0" connect_version="0410">
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
cachetoken="C4EF7130DF20DC09" />B
</response>
```

To get the details about our URIMAP CICS Explorer now issues a second GET HTTP request. This time the URI references the cached data identified by the cachetoken received earlier:

```
/CICSSystemManagement/CICSResultCache/C4EF7130DF20DC09/1/1
```

Note that the cachetoken is suffixed by `/1/1`. This tells CMCI to return only one record starting at record one. See Example 2-5.

Example 2-5   CMCI cached data HTTP GET request

```
GET ?NODISCARD=NODISCARD HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

This time the HTTP response contains the data we are looking for (Example 2-6).

Example 2-6   HTTP response to CICS Explorer inquire on URIMAP definition

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:20 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive
13D
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
AF
```

```
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
displayed_recordcount="1" cachetoken="C4EF7130DF20DC09" />
C
   <records>
2A4
      <cicsdefinitionurimap
_keydata="D4E8E4D9C94040400140404040404040" analyzer="NO"
atomservice="" authenticate="NO" certificate="" changeagent="DREPAPI"
changeagrel="0660" changetime="2009-10-14T11:26:19"
changeusrid="CICSRS1" characterset="" ciphers="" converter=""
createtime="2009-10-14T11:26:19" csdgroup="" defver="1"
desccodepage="37" description="" hfsfile="" host="host.ibm.com"
hostcodepage="" location="" mediatype="" name="MYURI" path="/path"
pipeline="MYPIPE" port="NO" program="" redirecttype="NONE"
scheme="HTTP" status="ENABLED" tcpipservice="" templatename=""
transaction="" usage="PIPELINE" userdata1="" userdata2="" userdata3=""
userid="" webservice="" />
D
   </records>
B
</response>
```

2.2.3 Altering a URIMAP resource definition using CMCI

To request CMCI to alter a resource definition, we use the PUT method on our HTTP request. The URI identifies the resource to be updated as a resource definition URIMAP (CICSDefinitionURIMap) in CICSPlex EPRED. The query string identifies the resource to be altered as MYURI. See Example 2-7.

Example 2-7 HTTP request to CMCI to alter a CICS resource definition

```
PUT
/CICSSystemManagement/CICSDefinitionURIMap/EPRED//?CRITERIA=((NAME='MYU
RI')%20AND%20(DEFVER='1')) HTTP/1.1
Authorization: Basic Q01DU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 76
```

```
<request><update><attributes DESCRIPTION="Hello World"
/></update></request>
```

The body of the HTTP request contains the XML message that details the change to be made to URIMAP. It is an update request to change the DESCRIPTION attribute of our URIMAP to `Hello World`. We can see the HTTP response sent on successful completion of the alter in Example 2-8 on page 32 with the good return code in the resultset element, and the updated description attribute of the cicsdefinitionurimap element.

Example 2-8 HTTP response from CMCI for an alter to a CICS resource definition

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:20 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive

13D
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
91
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
displayed_recordcount="1" />
C
   <records>
2AF
      <cicsdefinitionurimap
_keydata="D4E8E4D9C94040400140404040404040" analyzer="NO"
atomservice="" authenticate="NO" certificate="" changeagent="DREPAPI"
changeagrel="0660" changetime="2009-10-14T11:26:20"
changeusrid="CICSRS1" characterset="" ciphers="" converter=""
createtime="2009-10-14T11:26:19" csdgroup="" defver="1"
desccodepage="37" description="Hello World" hfsfile=""
host="host.ibm.com" hostcodepage="" location="" mediatype=""
name="MYURI" path="/path" pipeline="MYPIPE" port="NO" program=""
redirecttype="NONE" scheme="HTTP" status="ENABLED" tcpipservice=""
```

```
templatename="" transaction="" usage="PIPELINE" userdata1=""
userdata2="" userdata3="" userid="" webservice="" />
D
   </records>
B
</response>
```

2.2.4 Installing a CICS resource definition using CMCI

The HTTP method used to install a CICS resource definition is `PUT`. See Example 2-9.

Example 2-9 HTTP Put request for CMCI Install

```
PUT
/CICSSystemManagement/CICSDefinitionURIMap/EPRED/?CRITERIA=((NAME='MYURI')%20AN
D%20(DEFVER='1')) HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 131

<request>
   <action name="INSTALL">
      <parameter name="USAGE" value="LOCAL"/>
      <parameter name="TARGET" value="EPRED4"/>
   </action>
</request>
```

The URI identifies:

- The kind of resource we want to work with: `CICSDefinitionURIMAP`
- The CICSPlex context `EPRED.`

The query string identifies the resource we want to install:

`CRITERIA=((NAME='MYURI')%20AND%20(DEFVER='1')).`

The body of the HTTP request contains an XML message with more detail about the operation to be performed. The action to be performed is an install, and the target region for the install is CICS region EPRED4.

If the install for a resource is successful, CMCI returns the HTTP response shown in Example 2-10. We can see from the `api_response1` attribute of the resultset element that the operation completed normally. The details of the requested resource MYURI are returned in the records element.

Example 2-10 HTTP Response for CMCI Isntall

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:21 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive

13D
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
91
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
displayed_recordcount="1" />
C
   <records>
2AF
      <cicsdefinitionurimap
_keydata="D4E8E4D9C940404001404040404040404040" analyzer="NO"
atomservice="" authenticate="NO" certificate="" changeagent="DREPAPI"
changeagrel="0660" changetime="2009-10-14T11:26:20"
changeusrid="CICSRS1" characterset="" ciphers="" converter=""
createtime="2009-10-14T11:26:19" csdgroup="" defver="1"
desccodepage="37" description="Hello World" hfsfile=""
host="host.ibm.com" hostcodepage="" location="" mediatype=""
name="MYURI" path="/path" pipeline="MYPIPE" port="NO" program=""
redirecttype="NONE" scheme="HTTP" status="ENABLED" tcpipservice=""
templatename="" transaction="" usage="PIPELINE" userdata1=""
userdata2="" userdata3="" userid="" webservice="" />
D
   </records>
B
</response>
```

2.2.5 Inquiring on an installed CICS resource using CMCI

As well as installing and updating CICS resource definitions (either in the CSD or in BAS), CMCI allows you to inquire on the operational status of resources that are already installed and in use. The HTTP request shown in Example 2-11 shows an HTTP request sent by CICS Explorer to CMCI to inquire on the status of our installed URIMAP. You can see that the HTTP method specified is GET. As usual, the type of resource we are inquiring about is specified on the URI:

```
/CICSSystemManagement/CICSURIMap/EPRED/EPRED4
```

What do you notice about this URI compared to the one we have seen up until now?

Example 2-11 HTTP request to Inquire on ain installed CICS resource using CMCI

```
GET
/CICSSystemManagement/CICSURIMap/EPRED/EPRED4?NODISCARD=NODISCARD&SUMMO
NLY&CRITERIA=((NAME='MYURI')) HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Because we are inquiring on an installed resource rather than a resource definition, the resource type is different from the one we have used in previous examples. We are dealing with a CICSURIMap. The context is still CICSPlex EPRED, but the scope of this request is to only target CICS Region EPRED4.

Example 2-12 shows the HTTP response returned by CMCI to reply to our URIMAP inquiry. The body of the response shows that the operation was successful. The results of the inquiry have been cached, in the cache identified by the cachetoken attribute of the resultset attribute.

Example 2-12 HTTP response with cachetoken for CMCI Inquire

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:35 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive

13D
<?xml version="1.0"?>
```

```
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
95
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
cachetoken="C4EF713F93BD0F82" />
B
</response>
```

To retrieve the actual data corresponding to the inquire, CICS Explorer sends a second HTTP request. The requested object type is CICSResultCache, and the URI also contains the cachetoken returned to CICS Explorer by CMCI, and the fact that CICS Explorer is only interested in one URIMAP, beginning at the first one. See Example 2-13.

Example 2-13 HTTP Get request for data for CMCI Inquire

```
GET
/CICSSystemManagement/CICSResultCache/C4EF713F93BD0F82/1/1?NODISCARD=NO
DISCARD HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

This time the HTTP response contains the attributes of our installed URIMAP. See Example 2-14.

Example 2-14 HTTP response with data for CMCI inquire

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:35 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive

13D
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
AF
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
displayed_recordcount="1" cachetoken="C4EF713F93BD0F82" />
C
   <records>
37A
      <cicsurimap _keydata="D4E8E4D9C9404040" analyzerstat="NOANALYZER"
atomservice="" authenticate="NOAUTHENTIC" basdefinever="1"
certificate="" changeagent="DREPAPI" changeagrel="0660"
changetime="2009-10-14T11:26:20" changeusrid="CICSRS1" characterset=""
ciphers="" converter="" definesource="CPSMV01"
definetime="2009-10-14T11:26:19" enablestatus="ENABLED"
eyu_cicsname="EPRED4" eyu_cicsrel="E660" eyu_reserved="0" hfsfile=""
host="host.ibm.com" hostcodepage="" hosttype="NOTAPPLIC"
installagent="CREATESPI" installtime="2009-10-14T11:26:21"
installusrid="CICSRS1" ipfamily="UNKNOWN" ipresolved="0.0.0.0"
location="" maprefcount="0" matchdisabld="0" matchredirec="0"
mediatype="" name="MYURI" numciphers="0" path="/path" pipeline="MYPIPE"
port="-1" program="" redirecttype="NONE" scheme="HTTP" tcpipservice=""
templatename="" transaction="CPIH" usage="PIPELINE" userid=""
webservice="" />
   </records>
B
</response>
```

2.2.6 Modifying an installed CICS resource using CMCI

To request CMCI to modify an installed resource, we use the PUT method on our HTTP request. The URI identifies the resource to be updated as an installed URIMAP (CICSURIMap) in region EPRED4 in CICSplex EPRED. See Example 2-15.

Example 2-15 HTTP PUT request for a CMCI modification

```
PUT
/CICSSystemManagement/CICSURIMap/EPRED/EPRED4?CRITERIA=((NAME='MYURI'))
HTTP/1.1
Authorization: Basic Q0lDU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 51

<request><action name="DISABLE"></action></request>
```

The body of the HTTP request contains the XML message with the details of the update we want to perform, in this case DISABLE the URIMAP. Example 2-16 shows the HTTP response returned by the CMCI. Note that the enablestatus attribute of the cicsurimap element is set to DISABLED.

Example 2-16 HTTP response to CMCI disable request

```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:36 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive
13D
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
91
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
displayed_recordcount="1" />

C
   <records>
386
      <cicsurimap _keydata="D4E8E4D9C9404040" analyzerstat="NOANALYZER"
atomservice="" authenticate="NOAUTHENTIC" basdefinever="1"
certificate="" changeagent="DREPAPI" changeagrel="0660"
changetime="2009-10-14T11:26:20" changeusrid="CICSRS1" characterset=""
ciphers="" converter="" definesource="CPSMV01"
definetime="2009-10-14T11:26:19" enablestatus="DISABLED"
eyu_cicsname="EPRED4" eyu_cicsrel="E660" eyu_reserved="0" hfsfile=""
host="host.ibm.com" hostcodepage="" hosttype="NOTAPPLIC"
installagent="CREATESPI" installtime="2009-10-14T11:26:21"
installusrid="CICSRS1" ipfamily="UNKNOWN" ipresolved="0.0.0.0"
```

```
location="Hello World" maprefcount="0" matchdisabld="0"
matchredirec="0" mediatype="" name="MYURI" numciphers="0" path="/path"
pipeline="MYPIPE" port="-1" program="" redirecttype="NONE"
scheme="HTTP" tcpipservice="" templatename="" transaction="CPIH"
usage="PIPELINE" userid="" webservice="" />
D
   </records>
B
</response>
```

2.2.7 Discarding an installed resource using CMCI

In Example 2-17 we see the HTTP request sent to the CMCI by CICS Explorer when it wants to discard an installed CICS resource. Note that now we are using the `DELETE` method. The URI identifies the type of resource to be discarded, and which CICS region in which CICSPlex, whilst the query string gives the name of the resource to be discarded MYURI.

Example 2-17 HTTP request to CMCI discard an installed resource

```
DELETE
/CICSSystemManagement/CICSURIMap/EPRED/EPRED4/?CRITERIA=((NAME='MYURI')
) HTTP/1.1
Authorization: Basic Q01DU1JTMTpibHUzQjBvaw==
User-Agent: IBM_CICS_CICS Explorer/1.0.2.200909240951 (Windows XP)
Host: localhost:16001
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

When CICS has successfully discarded MYURI, it returns the HTTP response shown in Example 2-18, with `api_response1` set to 1024.

Example 2-18 HTTP Response from CMCI to a discard of an installed resource

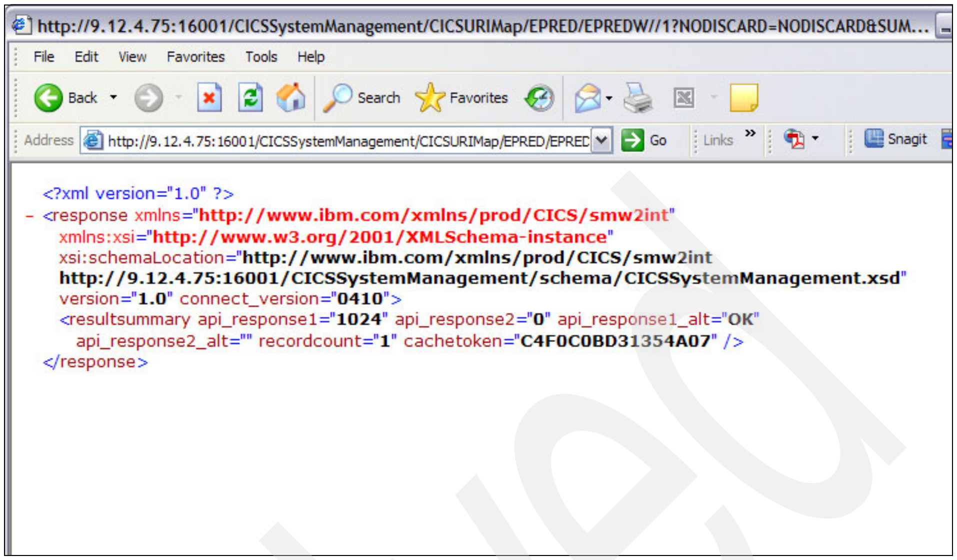
```
HTTP/1.1 200 OK
Cache-Control: no-store
Date: Wed, 14 Oct 2009 15:26:37 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive
13D
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
http://localhost:16001/CICSSystemManagement/schema/CICSSystemManagement
.xsd" version="1.0" connect_version="0410">
88
   <resultsummary api_response1="1024" api_response2="0"
api_response1_alt="OK" api_response2_alt="" recordcount="1"
successcount="1" />
B
</response>
```

2.2.8 A fun way to explore the CMCI Interface

If you want to learn about the CMCI and better understand the messages that flow between the client and CMCI, you can see some of it from a Web browser. Enter the following URL on a Web browser:

```
http://9.12.4.75:16001/CICSSystemManagement/CICSURIMap/EPRED/EPREDW//1?
NODISCARD=NODISCARD&SUMMONLY
```

You see the XML response returned by the CMCI shown in Figure 2-1.


http://9.12.4.75:16001/CICSSystemManagement/CICSURIMap/EPRED/EPREDW//1?NODISCARD=NODISCARD&SUM...
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://9.12.4.75:16001/CICSSystemManagement/CICSURIMap/EPRED/EPRED
Go
Links
Snagit

```
  <?xml version="1.0" ?>
- <response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/xmlns/prod/CICS/smw2int
    http://9.12.4.75:16001/CICSSystemManagement/schema/CICSSystemManagement.xsd"
    version="1.0" connect_version="0410">
    <resultsummary api_response1="1024" api_response2="0" api_response1_alt="OK"
      api_response2_alt="" recordcount="1" cachetoken="C4F0C0BD31354A07" />
  </response>
```

Figure 2-1 Using a Web browser to drive the CMCI

2.2.9 Comparing the Web user interface and CICS Explorer

For a CICS person, probably the simplest way to appreciate the advantages of a RESTful style application when compared with a more traditional Web application is to compare the CICSPlex SM WUI with the CICS Explorer. They do much the same thing, but most users agree that the CICS Explorer is a far superior implementation.

WUI Plumbing

Figure 2-2 is an overview of how information about the CICSPlex is exchanged between the WUI and its HTTP client.

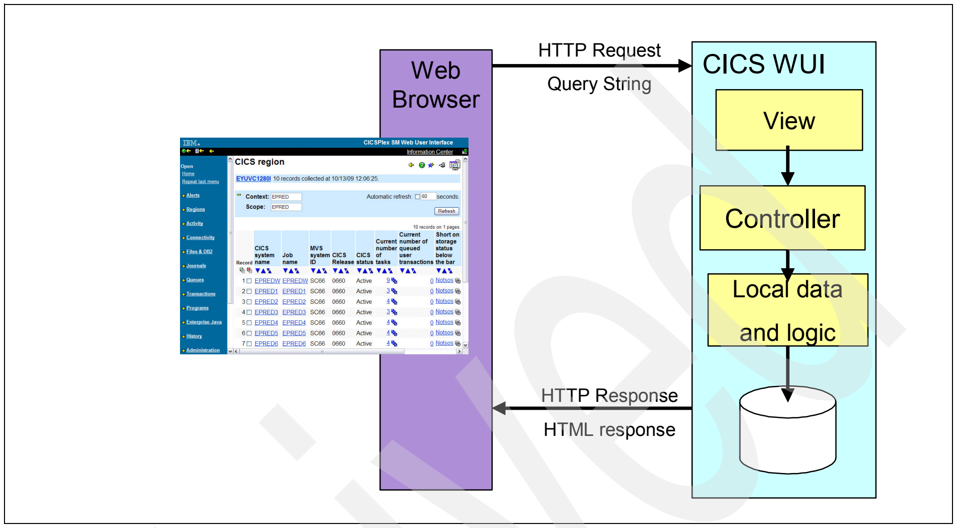


Figure 2-2 Browser-based component architecture

The workstation element of the WUI is a Web browser (an HTTP client), that is, Internet CICS Explorer, or Mozilla Firefox, or something similar. It is a thin client, with almost no client-side programming logic other than Javascript to help with formatting of the HTML data being passed from the CICS WUI region to the browser. All the information about the CICS resources and the logic to manage that information being passed to the user is executing in CICSPlex SM.

The data sent from CICS to the Web browser is in HTML format, and is often large, resulting in large amounts of data flowing across the network, and slower response times. This is because all the data relevant to a request from the user is returned in a single response. If you repeat the request, all the information is resent from CICS to the Web browser.

When you use the WUI, all communication between the Web browser is synchronous. The user has to wait for the HTTP response to the last input be returned from CICS before he can continue with his work.

CICS Explorer Plumbing

In Figure 2-3 we see a high-level view of the CICS Explorer architecture. There is a lot more happening on the client side than in the WUI model. The client (Eclipse RCP) incorporates logic and data to handle the interface with the user, the state of the user session, and to hold information about the CICS configuration being used. This information extends to a detailed model of the CICSPlex being managed (based on the CICSPlex SM Resource Tables Reference model).

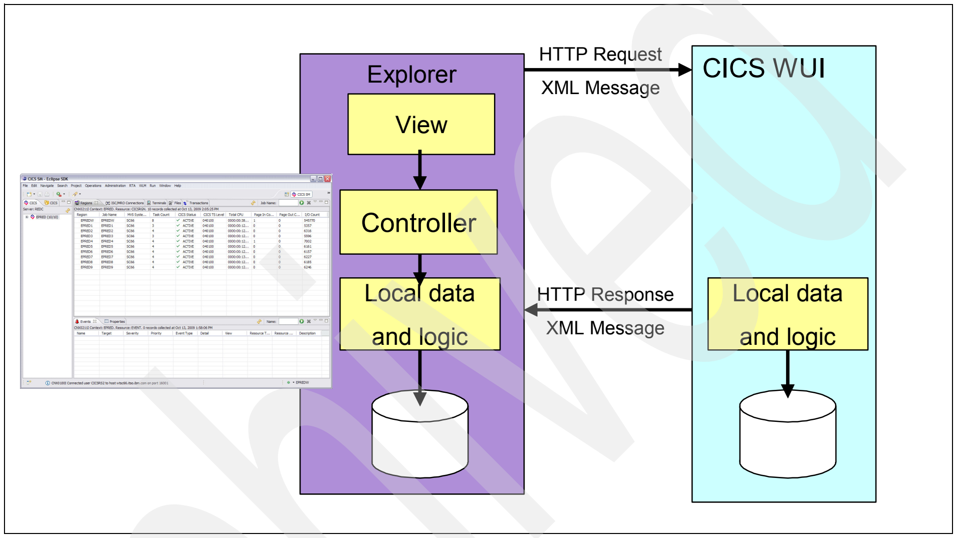


Figure 2-3 CICS Explorer component architecture

There is logic in CICS Explorer to cache data and manage the session, so CICS Explorer is not obliged to send an HTTP request to the WUI CICS region for each user interaction. Because CICS Explorer is managing the user interface, the data that flows from the WUI CICS region need not be HTML. In fact, CICS Explorer exchanges XML messages with the target WUI CICS region.

Responsibility for holding information about CICS resources is shared between CICS Explorer and CICSPlex SM. In several cases, when the user makes a request for what might be a large amount of data, the WUI CICS region caches the requested data, and passes subsets of the cached data to the CICS Explorer only when they are requested. So, for example, as the user scrolls through a list of CICS regions, CICS Explorer sends requests for the next subset of data at the moment the user needs it.

3

CICS Explorer SDK

In this chapter we take an in depth look at the CICS Explorer SDK. The SDK provides a collection of Java objects that give a programmatic interface for creating, updating, and performing actions on CICS resources and definitions.

3.1 CICS Explorer SDK Java library

As discussed in Chapter 2, “CICS Explorer and the CICS Client Management Interface” on page 23, the CICS Explorer interacts with CICS using a RESTful interface. To facilitate a more productive Java programming experience, a collection of Java objects have been created that wrapper the generation of RESTful requests and the parsing of XML response data. Collectively, these Java objects are known as the CICS Explorer SDK.

You can download the latest version of the CICS Explorer SDK by following these steps:

1. Go to the following Web site:

   `http://www-01.ibm.com/software/htp/cics/explorer/`

2. Select **Downloads** → **CICS Explorer** → **CICS Explorer for CICS TS V4 licensees** → **Link to zip file**.
3. Input your IBM ID or register if you do not have one.
4. Answer the questionnaire and agree to the license statement.
5. Select **CICS Explorer for CICS Transaction Server for z/OS V4.1 (SDK)** and then click **Download now**. Your CICS Explorer SDK download then begins.
6. After the download has finished follow the instructions detailed in “Creating your CICS Explorer plug-in development environment” on page 62 for installing into your Eclipse development environment.

CICS Explorer SDK grows in parallel with the CICS Explorer. With the frequently planned code drops of the CICS Explorer, it is always worth investigating whether you have the latest version of the CICS Explorer SDK.

The version discussed in this chapter is `cicstsv41_explorer_sdk_v1002.zip`.

In the following sections we highlight the important objects in the CICS Explorer SDK, and explain how they are used.

3.2 The CICSPlex System Manager object

To get information about your CICSplex, or to perform actions on CICS systems, you need to have an instance of the CICSPlex System Manager (ICPSM) object. This is the main object you use to gain access to your CICS Explorer. Example 3-1 on page 47 shows how you obtain a reference to the ICPSM object.

Example 3-1 Obtaining an instance of the ICPSM object

```
ICPSM cpsm = UIPlugin.getDefault().getCPSM();
```

With a reference to the **cpsm** object, you are able to retrieve basic information about the CICS regions to which you are connected. Example 3-2 shows how to obtain a selection of this information from the **cpsm** object.

Example 3-2 Getting information from the ICPSM object

```
System.out.println("Name: " + cpsm.getName());
System.out.println("Host: " + cpsm.getHost());
System.out.println("Port: " + cpsm.getPort());
System.out.println("UserID: " + cpsm.getUserID());
System.out.println("Connected: " + cpsm.isConnected());
```

The output generated from this code is shown in Example 3-2.

Example 3-3 Output from Example 3-2

```
Name: REDC     // CMAS SYSIDNT
Host: wtsc66.itso.ibm.com// WUI hostname
Port: 16001    // WUI port no.
UserID: CICSRS1// Connected userid
Connected: true// CICS Explorer connection to CICSplex
```

The cpsm object also provides a mechanism for traversing the collection of CICSplexes and their managed CICS regions. Example 3-4 shows how to do this.

Example 3-4 Getting CICSplex information from the ICPSM object

```
// Iterate through the available CICSplexes and their regions
for (ICICSplex cicsPlex : cpsm.getCICSplexes())
{
   System.out.println("CICSplex = " + cicsPlex.getName());

   for (IManagedRegion region :
      cpsm.getManagedRegions(cicsPlex))
   {
      System.out.println("\tRegion = " + region.getName());
   }
}
```

The output generated from this code is shown in Example 3-5.

Example 3-5 Our CICSplex and its CICS regions

```
CICSplex: EPRED
   Region: EPREDW
   Region: EPRED1
   Region: EPRED2
   Region: EPRED3
   Region: EPRED4
   Region: EPRED5
   Region: EPRED6
   Region: EPRED7
   Region: EPRED8
   Region: EPRED9
```

The cpsm object is also used for interacting with CICS resources and definitions. To do this, familiarize yourself with using contexts in the SDK. Contexts are discussed in 3.3, “Working with contexts” on page 48.

3.3 Working with contexts

In Example 3-4 on page 47 you retrieved the names of the CICSplexes known to this instance of CPSM. Next, we define what subset of those CICS resources we want to work with. Iit one particular CICSplex? Or one particular CICS region?

Contexts and scopes are mechanisms used in CICSPlex System Manager to provide scoping on the collection of CICS regions with which you want to work. In the simplest case you can set the context to the name of your CICSplex and the scope to an individual CICS region. In the CICS Explorer SDK utility classes have been created to represent contexts and scopes. There is also a class to enable advanced filtering. Example 3-6 shows an instance of context that is set to the name of a CICSplex.

Example 3-6 CICSplex context

```
Context plexContext = new Context("EPRED");
```

Example 3-7 on page 49 shows an instance of ScopedContext that is set to an individual region in the preceding CICSplex. In addition to passing the preceding object to the constructor of ScopedContext, you can also directly pass a string containing the CICSplex name, as shown in Example 3-7 on page 49.

Example 3-7 CICS region within CICSplex context

```
ScopedContext scopedContext
   = new ScopedContext(plexContext, "EPRED4");

// An alternative way of instantiating a ScopedContext object
ScopedContext scopedContext
   = new ScopedContext("EPRED", "EPRED4");
```

For advanced filtering you can use the AbstractFilteredContext Java class. This class provides the ability to filter on any attribute of the resource on which you are filtering. Because AbstractFilteredContext is an abstract class it requires you to create a new Java class that extends it and implements the methods `getAttributeValue` and `getAttributeNames`.

Example 3-8 shows an inner class that extends AbstractFilteredContext and returns all URIMaps in a CICS region that have the name MYURI. To scope it to a single CICS region, the scopedContext object is passed to the constructor of AbstractFilteredContext.

Example 3-8 Filter on a URIMap resource in single CICS region within a CICSplex

```
IFilteredContext filteredScopedContext
   = new AbstractFilteredContext(scopedContext)
{
   public String getAttributeValue(String arg0)
   {
      return “MYURI”;
   }

   public Set<String> getAttributeNames()
   {
return Collections.singleton(URIMapType.NAME.getCicsName());
   }
};
```

The call to URIMapType.NAME.getCICSName() returns the internal key for that attribute. This is used by CICSPlex System Manager to filter on that type. The class name “URIMapType” can be changed for any other CICS type. The value following it (which in this case is NAME) can be changed to any supported attribute on that type. The call to getCicsName() returns the string representation of that attribute.

You are not limited to passing a ScopedContext object to the constructor of AbstractFilteredContext. You can also pass in any other context object. Example 3-9 shows the use of AbstractFilterContext to filter on all URIMap definitions in a CICSplex. Note that a CICSplex wide context is passed to its constructor.

Example 3-9 Filter on a URIMapDefinition resource in a CICSplex

```
IFilteredContext filteredPlexContext
   = new AbstractFilteredContext(plexContext)
{
   public String getAttributeValue(String arg0)
   {
      return “MYURI”;
   }

   public Set<String> getAttributeNames()
   {
return Collections.singleton(URIMapDefinitionType.NAME.getCicsName());
   }
};
```

The ability to filter on attributes doesn’t stop there. You can also filter on multiple attributes at the same time. Example 3-10 shows a filter that looks for all TSQueues in a CICSplex that have a name beginning with the letter “S” and a queue length of 128.

Example 3-10 Filtering on multiple attributes

```
final HashMap<String, String> attributes
   = new HashMap<String, String>();

attributes.put(TSQueueType.NAME.getCicsName(), "S*");
attributes.put(TSQueueType.QUEUE_LENGTH.getCicsName(), "128");

IFilteredContext multipleFilteredPlexContext
   = new AbstractFilteredContext(plexContext)
{
   public String getAttributeValue(String arg0)
   {
      return attributes.get(arg0);
   }

   public Set<String> getAttributeNames()
   {
      return attributes.keySet();
   }
};
```

3.4 CICS objects

When you retrieve resources using the CICS Explorer SDK, they are given as an instance of ICICSObject. If you want to work with a specific type, whether it is a resource or a definition, cast the ICICSObject to that type. Because all things are treated as ICICSObjects, there are generic utility methods supplied for retrieving, performing actions on, and updating them.

3.4.1 Getting a CICS object

To obtain a reference to a ICICSObject, provide a reference to the CICS type that you are after, and a context instance. The `getModel()` method of the cpsm object provides a mechanism which you can use to browse the results.

Call model.activate() for the result set size to be calculated. Next, you have the ability to fetch results in chunks using the `model.maybeFetch()` method. The parameters to the method are the start and end index values respectively. Using this method enables you to minimize the results that are requested across the network at any one point. Finally, to get hold of each ICICSObject in, call model.get() and pass it an index value. Example 3-11 illustrates this.

Example 3-11 Getting all the URIMapDefinitions in a CICSplex

```
IResourcesModel model
= cpsm.getModel(CICSTypes.URIMapDefinition, plexContext);

model.activate(); // Initiate the model

if (model.size() > 0) // Check there are objects to get
{
   ICICSObject[] cicsObjects = new ICICSObject[model.size()];

   model.maybeFetch(0, model.size()); // Fetch the results

   for (int i = 0; i < model.size(); i++)
      cicsObjects[i] = model.get(i);
}
```

Note: For a full list of CICSTypes see Appendix A, “Reference list of CICS SDK elements” on page 303.

It is good practice only to retrieve the results that you need to show on the display, and to do this in a just-in-time basis. This is how the CICS Explorer works. You can use the `model.size()` method to get an idea of the result set size by calling `model.size()`, and fetch the results in incremental chunks using multiple calls to `model.maybeFetch()`.

3.4.2 Performing actions on CICS objects

When you are able to get hold of CICS objects as shown in Example 3-11 on page 51, performing actions on them is easy. Using the cpsm object, pass the `perform()` method an array of CICSObjects and a reference to the action you want to perform. The `perform()` method declares that it can throw a CICSSystemManagerException if something goes wrong, so this needs to be dealt with by wrapping the call in a try/catch block. Example 3-12 shows the command for performing a disable action on CICS objects.

Example 3-12 Disabling multiple CICS objects

```
try
{
   cpsm.perform(cicsObjects, SystemManagerActions.Disable);
}
catch (CICSSystemManagerException e) { e.printStackTrace(); }
```

Note: For a full list of SystemManagerActions see Appendix A, "Reference list of CICS SDK elements" on page 303.

To perform a separate action such as a pipeline scan, you need to swap SystemManagerActions.Disable for SystemManagerActions.Scan. If you attempt to perform an action on a resource that does not support it (such as a pipeline scan on a program resource), a CICSSystemManagerException is thrown, as shown in Example 3-13.

Example 3-13 Exception trace from performing an invalid command

```
com.ibm.cics.core.model.CICSSystemManagerException:
com.ibm.cics.sm.comm.sm.SMConnectionException: NOTFOUND,
ACTION
```

3.4.3 Updating a CICS object

To update a CICS object you first need to get a mutable version of it. This is done by casting your ICICSObject it to an ICoreObject and calling the `getAdapter()` method on the ICoreObject. The call to this method returns an instance of IMutableCoreObject that you cast to a mutable version of the object with which you are working. After updating various values on the object, feed the original reference to the IMutableCoreObject to the cpsm.update() method to commit the changes.

Example 3-14 shows how to do this with an IMutableURIMapDefinition. The final call to the cpsm object needs to be wrapped in a try/catch block to deal with any CICSSystemManagerExceptions that might occur.

Example 3-14 Updating a CICS object

```
// Cast ICICSObject instance to an ICoreObject
ICoreObject coreObject = (ICoreObject) cicsObjects[0];

// Get an instance of IMutableCoreObject
IMutableCoreObject mutableCoreObject = (IMutableCoreObject)
   coreObject.getAdapter(IMutableCoreObject.class);

// Convert it to an IMutableURIMapDefinition
IMutableURIMapDefinition mutableURIMapDefinition
   = (IMutableURIMapDefinition) mutableCoreObject;

// Update the description field
mutableURIMapDefinition.setDescription("New description");

try
{
   // Commit the update
   IMutableCoreObject updatedMutableCoreObject
      = cpsm.update(mutableCoreObject);
}
catch (CICSSystemManagerException e) { e.printStackTrace(); }
```

Note: For a full list of Mutable objects see Appendix A, "Reference list of CICS SDK elements" on page 303.

The call to cpsm.update() returns you a reference to the updated object so that you can continue working with it if required.

3.5 Definitions

When you want to create new resource instances using the CICS Explorer SDK, you need to create definitions for those resources. These definitions map to definitions in the CICS System Manager Business Application Services (BAS) repository or in the CICS CSD, depending on how the CICS Explorer is connected to your CICS systems. After creating a definition you are able to install or delete it.

3.5.1 Creating a definition

To create a new definition you need to create a IDefinitionBuilder for the resource type. The parameters you pass to the constructor of your definition builder are the minimum ones that are needed to create and install that resource. The remaining parameters inherit default values that can be updated by following the steps in Example 3-14 on page 53 on the returned IMutableCoreObject.

Example 3-15 shows the code for creating a pipeline specific URIMap definition.

Example 3-15 Creating a definition resource for a URIMap

```
IDefinitionBuilder definitionBuilder
   = new PipelineURIMapDefinitionBuilder("MYURIMap",
                                          1L,
                                          "www.ibm.com"
                                          "/path",
                                          "MYPIPE");

try
{
   IMutableCoreObject mutableCoreObject
      = cpsm.create(plexContext, definitionBuilder);
}
catch (CICSSystemManagerException e) { e.printStackTrace(); }
```

Note: For a full list of DefinitionBuilders see Appendix A, "Reference list of CICS SDK elements" on page 303.

Be aware: At the time of publishing there is an inconsistency in the error handling of the create command. If you attempt to create a definition that already exists then you get a com.ibm.cics.sm.comm.CreateException instead of a CICSSystemManagerException. The CreateException is a RuntimeException, so you are not forced to code a try/catch block for it. This means that you might forget to code for this case. If this happens, exception trace gets dumped to the stderr output stream if the error occurs.

This inconsistency is resolved in a future version of the CICS Explorer SDK.

3.5.2 Installing a definition

Having created a definition the next stage is to install it. To do this, call the `cpsm.install()` method passing it the definition instance and a context in which to install it. Example 3-16 shows code for installing a definition.

Example 3-16 Installing a definition

```
try
{
   IDefinition installedDefinition
      = cpsm.install(uriMapDefinition, scopedContext);
}
catch (CICSSystemManagerException e) { e.printStackTrace(); }
```

If you try to get hold of the resource immediately after the install command, you might get no results back initially. This is because the install process causes an asynchronous task to install the resource from the definition. When the install has completed, you are able to retrieve a reference to it.

Example 3-17 shows code that loops around until the resource has been installed. If you use this code you might want the loop to time out after a given number of iterations. Additionally, you must ensure that any thread waits happen on a background thread and not the main GUI thread, as discussed in 5.5, "Background process implementation" on page 138.

Example 3-17 Looping while you wait for a resource to be installed

```
while (model.size() == 0)
{
   try { Thread.sleep(1000); } // Sleep for 1 second
   catch (InterruptedException e) {}

   model = cpsm.getModel(CICSTypes.URIMap, plexContext);
   model.activate();
}
```

3.5.3 Deleting a definition

If you want to delete a definition, call cpsm.delete() and pass a reference to it. Example 3-18 shows how this is done.

Example 3-18 Deleting a definition

```
try
{
   cpsm.delete(uriMapDefinition);
}
catch (CICSSystemManagerException e) { e.printStackTrace(); }
```

3.6 Resources

Previous sections have shown how to get, perform actions on, and update resources in CICS. The final stage of the life cycle is to be able to discard them.

3.6.1 Discard a resource

To discard a resource, you need to add them to an ICICSResource array, which is then passed to cpsm.discardResources(). Example 3-19 on page 56 shows a quick way of copying an array of ICICSObjects to an ICICSResource array, the contents of which are then discarded.

Example 3-19 Discarding resources

```
ICICSResource[] cicsResources
   = new ICICSResource[cicsObjects.length];

System.arraycopy(cicsObjects, 0,
                     cicsResources, 0,
                     cicsObjects.length);

try
{
   cpsm.discardResource(cicsResources);
}
catch (CICSSystemManagerException e)
```

```
{
   SMConnectionException connectionException
      = (SMConnectionException) e.getCause();

   System.out.println("ReasonName: "
         + connectionException.getReasonName());
   System.out.println("ResponseName: "
         + connectionException.getResponseName());
   System.out.println("Resp1Name: "
         + connectionException.getResp1Name());
   System.out.println("Resp1: "
         + connectionException.getResp1());
   System.out.println("Resp2: "
         + connectionException.getResp2());
}
```

Additional error processing to visualize the difference between two separate error situations is included in Example 3-19. With this knowledge in mind, you are able to code for the separate situations. To get the response data, first obtain an instance of SMConnectionException. This is done by calling the `getCause()` method on the CICSSystemManagerException.

If you attempt to discard a resource that does not exist, you see the values shown in Example 3-20.

Example 3-20 Response values when attempting to delete a non-existent resource

```
connectionException.getReasonName(): OK
connectionException.getResponseName(): NODATA
connectionException.getResp1Name(): null
connectionException.getResp1(): 0
connectionException.getResp2(): 0
```

If you attempt to discard a resource before it has been disabled, you see the values shown in Example 3-21.

Example 3-21 Response values when attempting to discard an enabled resource

```
connectionException.getReasonName(): DATAERROR
connectionException.getResponseName(): TABLEERROR
connectionException.getResp1Name(): INVREQ
connectionException.getResp1(): 16
connectionException.getResp2(): 4
```

Part 3

Extending CICS Explorer

In this part of this IBM Redbooks publication we discuss how to install and configure the development environment you use to develop your CICS Explorer plug-ins. We also provide more examples of plug-ins that can be coded to use IBM CICS Explorer in useful ways.

4

Writing a plug-in for CICS Explorer

In this chapter we discuss how to install and configure the development environment you use to develop your CICS Explorer plug-ins. After the development environment is set up, we work through a “Hello World” plug-in example to get us started. After we have got our “Hello World” example, we use it as a vehicle to discover the various ways in which we can interact with the user and with CICS Explorer.

4.1 Creating your CICS Explorer plug-in development environment

Before you can develop your own CICS Explorer plug-ins, you need to install and configure the development environment. There are three components that you need to consider when setting this up:

- Java Virtual Machine
- Eclipse Classic SDK
- CICS Explorer SDK

Important: It is critical that you have the right software levels of all three of these components, and that you are using the correct JVM when installing the other components. Check the required levels for the version of the CICS Explorer SDK that you are using.

We used the following software levels for this Redbooks publication:

- Java Version 1.6.0 (build pwi3260sr5-20090529_04(SR5))
- IBM CICS Explorer   Version 1.0.0.2 [build id: 200909240951]
- Eclipse SDK Version 3.5.1 (Build id: M20090917-0800)
- IBM CICS Explorer   Version 1.0.0.2 [build id: 200909240951]

4.1.1 Setting up the Java environment

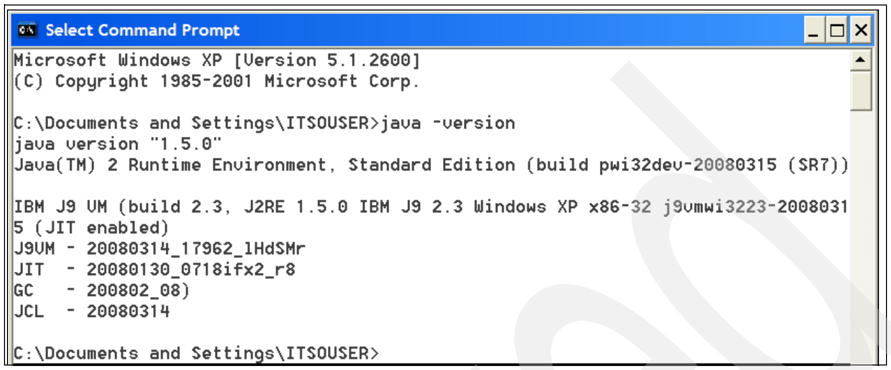
Ensure that you have the correct JVM installed and configured before you install the other components on your workstation. If you are using the latest version of CICS Explorer, you need Java Version 1.6. To determine the default version of Java running on your Windows® machine, open an MS-DOS window, and enter the command **`java -version`**. This command tells you whether there is a version of Java currently installed, and the version number. If it is not 1.6.0, download and install the later JVM.

If you do not have the correct version of Java, you can download it from the following Web page:

`http://www.ibm.com/developerworks/java/jdk/`

In this example, we downloaded the `ibm-java-sdk-60-win-i386.zip` file. After you download the necessary `.zip` file, extract it. We extracted the file to the `C:\software\ibm-java-sdk-60-win-i386` directory.

To make sure that the Java installation is successful, in an MS-DOS window, go to the `c:\ibm-java-sdk-60-win-i386\sdk\bin` directory and enter the **`java -version`** command. You see the messages shown in Figure 4-1.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ITSOUSER>java -version
java version "1.5.0"
Java(TM) 2 Runtime Environment, Standard Edition (build pwi32dev-20080315 (SR7))

IBM J9 VM (build 2.3, J2RE 1.5.0 IBM J9 2.3 Windows XP x86-32 j9vmwi3223-2008031
5 (JIT enabled)
J9VM - 20080314_17962_lHdSMr
JIT  - 20080130_0718ifx2_r8
GC   - 200802_08)
JCL  - 20080314

C:\Documents and Settings\ITSOUSER>
```

Figure 4-1 Determining the installed Java version

Now that you have the correct version of Java installed in your machine, you can install the Eclipse SDK.

4.1.2 Installing the Eclipse SDK onto your workstation

After you have verified which version of the Eclipse SDK you are working with (in our case it is Eclipse SDK Version 3.5.1), you can download it from the following Web page:

`http://www.eclipse.org/downloads`

We downloaded file `eclipse-SDK-3.5.1-win32.zip`, and extracted the contents into directory eclipse-SDK-3.5.1-win32.

Before you run Eclipse for the first time, verify that it runs with the Version 1.6 JVM you installed earlier. We did this by creating an executable called `startEclipse.bat` in the `C:\software` directory. This executable passes the address of the correct JVM when starting Eclipse. The source code for this .bat file is shown in Example 4-1.

Example 4-1 Source code to launch Eclipse pointing to correct JVM

```
C:\software\eclipse-SDK-3.5.1-win32\eclipse\eclipse -vm
C:\software\ibm-java-sdk-60-win-i386\sdk\bin
```

Using Windows CICS Explorer, open `C:\software\` and double-click `startEclipse.bat` to execute it. Your Eclipse workspace folder contains all the Eclipse artifacts that you create in the course of your plug-in development. For now, use the default workspace name and click **OK**. You see the Eclipse welcome panel displayed in Figure 4-2.

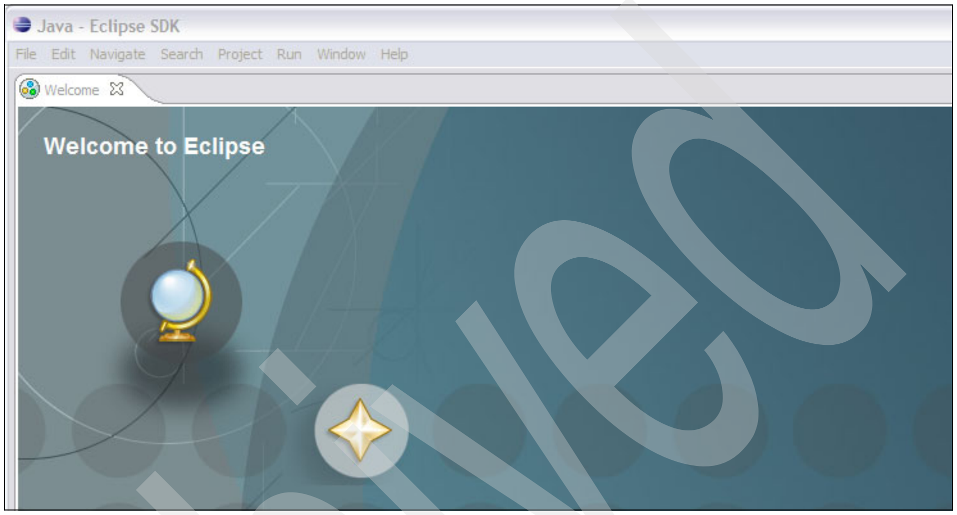
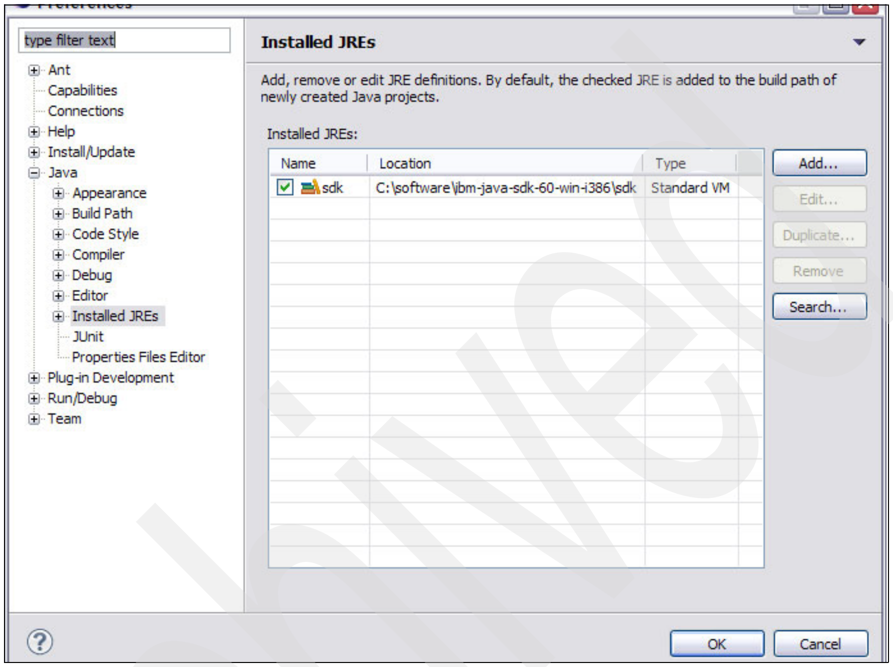


Figure 4-2 Eclipse Welcome panel

After you have Eclipse up and running, it is worth updating it to point to the Java 1.6 environment as its default runtime. Do this by clicking **Window** → **Preferences** → **Java** → **Installed JREs**. The panel shown in Figure 4-3 is displayed.



Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

Name	Location	Type
✓ sdk	C:\software\ibm-java-sdk-60-win-i386\sdk	Standard VM

Figure 4-3 Set default JRE for Eclipse (EclipseDefaultJVM.gif)

Click **Add** → **Standard JVM** → **Next**. Click **Directory** and select the home directory of your Java 1.6 JRE (in our case this directory is `C:\software\ibm-java-sdk-60-win-i386\sdk`), and click **Finish**. Select this JRE as the default for Eclipse to use and click **OK**.

You have now successfully created your Eclipse SDK environment. You are ready to develop your first Eclipse plug-in.

Note: So far, we have not installed any CICS code into Eclipse. We are just working with the environment provided by the Eclipse platform.

4.2 An Eclipse RCP "Hello World" plug-in

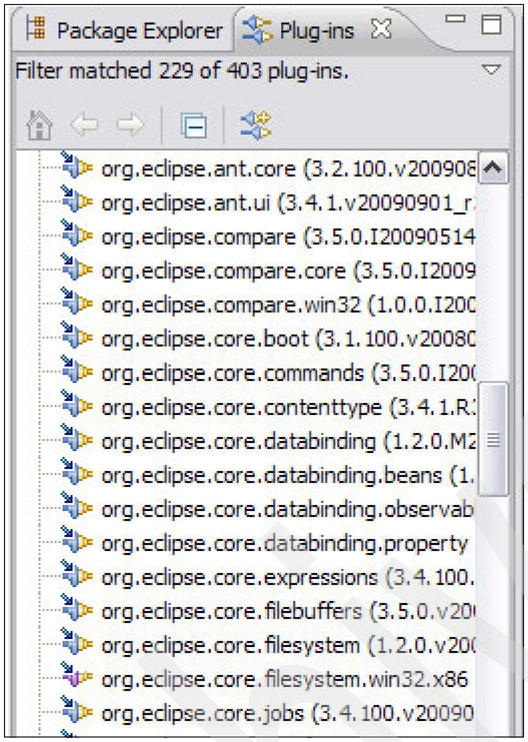
When you create your Eclipse SDK environment, it includes a set of templates and wizards to generate code skeletons that you can modify to add your own code. The Eclipse plug-in environment is Java-based, so you need to do Java coding to handle communications with the user and invoke business functionality behind (which might be Java). Working with the Eclipse "Hello World" samples is a good way to learn about coding for the Eclipse user interface.

4.2.1 Creating your plug-in project

Whenever you do something in Eclipse, you do it in the context of a project. You can think of a project as a folder in which Eclipse stores all the information and resources associated with this activity. There are many projects, depending on the task which you are executing. Because you want to create a new Eclipse plug-in, you are going to create a plug-in project.

You begin by creating a new Workspace. Start your Eclipse workbench, and create a new workspace. We called ours HelloWorldWorkspace. Because it is a new workspace, you are presented with the Eclipse Welcome panel.

Close the Welcome panel. You are presented with the Java Developers perspective. Click **Window** → **Open Perspective** → **Other** and select the Plug-in Developer perspective. You can see the layout is similar to the default. The difference is that the tab hidden behind the Package CICS Explorer view, in the navigation pane tab group on the left side of the panel, is a view called Plug-in. If you click the tab, you see the panel shown in Figure 4-4 on page 67.



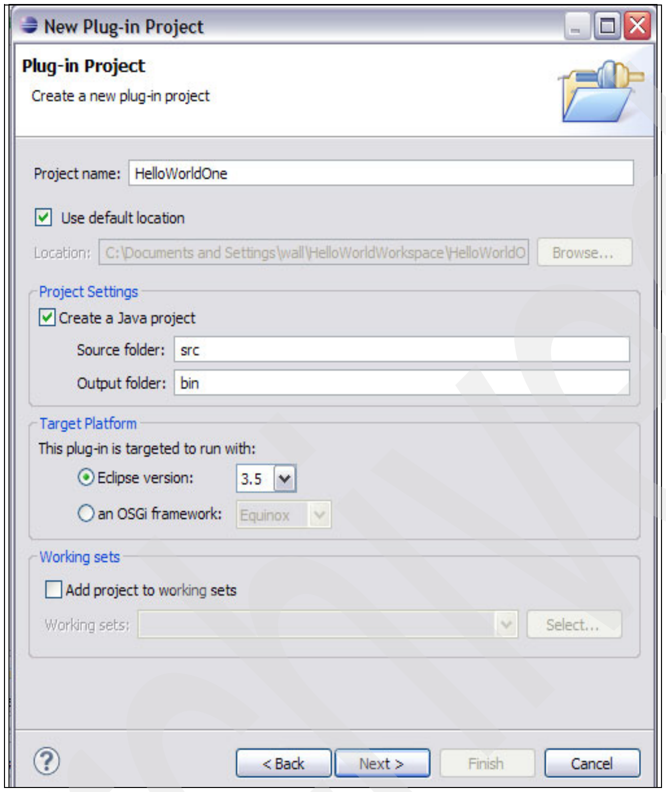
Filter matched 229 of 403 plug-ins.

org.eclipse.ant.core (3.2.100.v200908
org.eclipse.ant.ui (3.4.1.v20090901_r
org.eclipse.compare (3.5.0.I20090514
org.eclipse.compare.core (3.5.0.I2009
org.eclipse.compare.win32 (1.0.0.I200
org.eclipse.core.boot (3.1.100.v20080
org.eclipse.core.commands (3.5.0.I200
org.eclipse.core.contenttype (3.4.1.R
org.eclipse.core.databinding (1.2.0.M2
org.eclipse.core.databinding.beans (1.
org.eclipse.core.databinding.observab
org.eclipse.core.databinding.property
org.eclipse.core.expressions (3.4.100.
org.eclipse.core.filebuffers (3.5.0.v20
org.eclipse.core.filesystem (1.2.0.v20
org.eclipse.core.filesystem.win32.x86
org.eclipse.core.jobs (3.4.100.v20090

Figure 4-4 Plug-in view

These are all the plug-ins that make up the Eclipse SDK. They are available for re-use by your new plug-in. After you have coded you own plug-ins, they can be available for re-use in the same way. Click the Package CICS Explorer tab to return to the Package CICS Explorer view. It is currently empty.

To create your new plug-in package, click **File** → **New** → **Project** to start the New Project wizard. Select **Plug-in Project** (not Plug-in-Development), and click **Next**. You are presented with the panel shown in Figure 4-5.


New Plug-in Project

Plug-in Project
Create a new plug-in project

Project name: HelloWorldOne
☑ Use default location
Location: C:\Documents and Settings\wall\HelloWorldWorkspace\HelloWorldO

Project Settings
☑ Create a Java project
Source folder: src
Output folder: bin

Target Platform
This plug-in is targeted to run with:
◉ Eclipse version: 3.5
○ an OSGi framework: Equinox

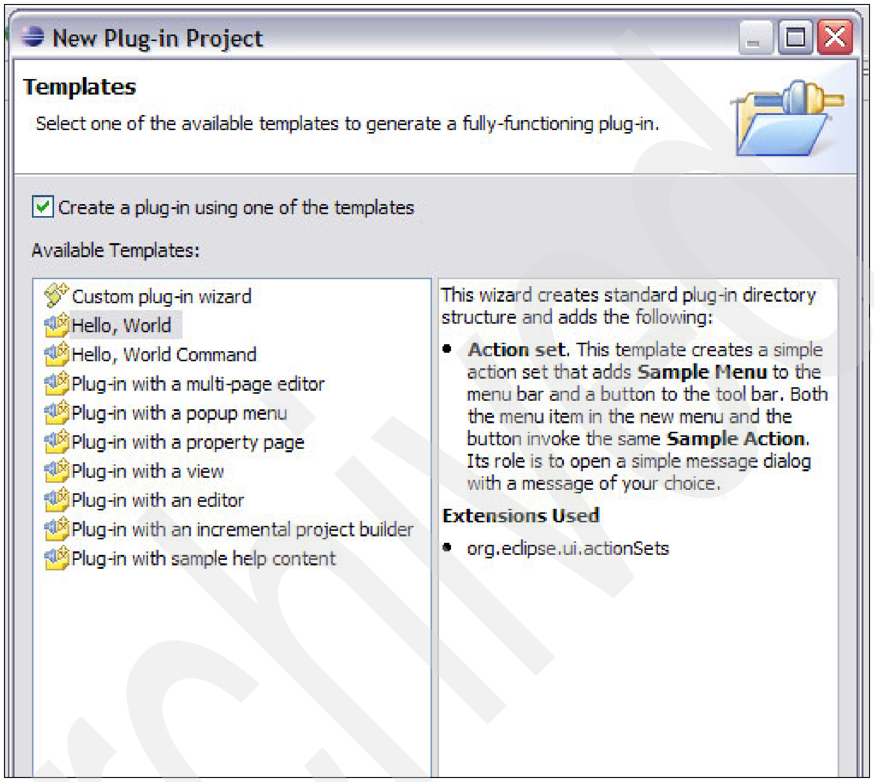
Working sets
☐ Add project to working sets
Working sets:

Figure 4-5 New Plug-in project wizard

Give the new project a name. We called ours “HelloWorldOne.” Leave the other values set to their default values. Click **Next.**

The next panel in the wizard asks you to give more details about the plug-in content. You can keep all the default values. We need an activator class (more about this later). We want to generate a plug-in that interfaces with the user. You are not creating an entire new Eclipse RCP application. Click **Next**.

Fortunately, Eclipse comes with a rich set of examples that can be used to familiarize ourselves with programming in this environment. At this point in the New Plug-in Project wizard, Eclipse gives a list of possible examples, or templates, on which you can base your new plug-in. This is shown in Figure 4-6.


New Plug-in Project

Templates

Select one of the available templates to generate a fully-functioning plug-in.

[✓] Create a plug-in using one of the templates

Available Templates:

- Custom plug-in wizard
- Hello, World
- Hello, World Command
- Plug-in with a multi-page editor
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with an editor
- Plug-in with an incremental project builder
- Plug-in with sample help content

This wizard creates standard plug-in directory structure and adds the following:

- **Action set.** This template creates a simple action set that adds **Sample Menu** to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same **Sample Action**. Its role is to open a simple message dialog with a message of your choice.

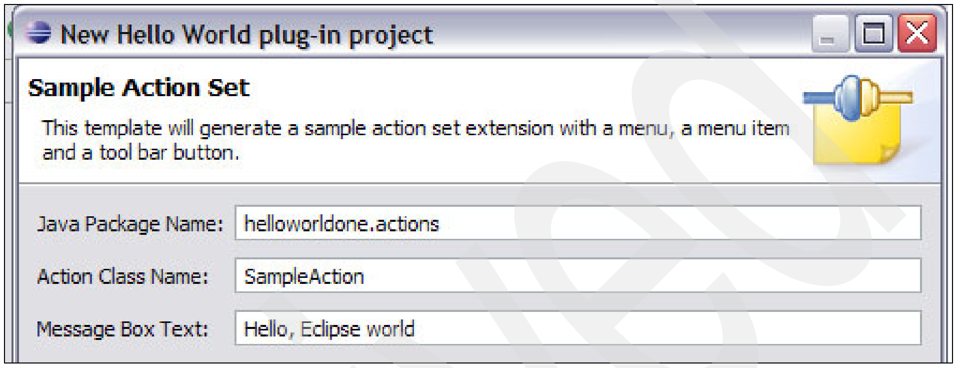
Extensions Used

- org.eclipse.ui.actionSets

Figure 4-6 List of plug-in templates provided by Eclipse

Try selecting each of the templates in turn. A description of the template appears in the pane on the right. When you have finished browsing the list, select the Hello, World template.

The wizard proceeds with the creation of your plug-in, and presents you with a panel summarizing the plug-in that you are creating (see Figure 4-7). Click **Finish** to create the plug-in.



Sample Action Set

This template will generate a sample action set extension with a menu, a menu item and a tool bar button.

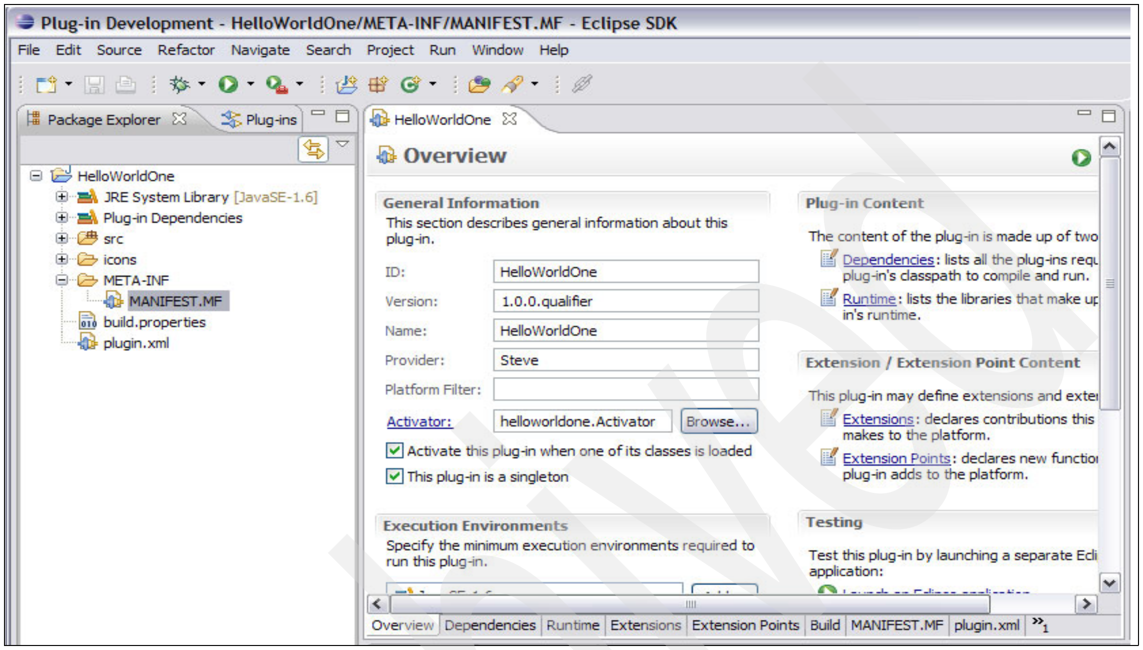
Java Package Name: helloworldone.actions

Action Class Name: SampleAction

Message Box Text: Hello, Eclipse world

Figure 4-7 First Hello World plug-in details

After you have clicked **Finish** in the New Plug-in Project wizard, the plug-in is created. In the Package CICS Explorer view on the left side, you can see the various objects that have been created. See Figure 4-8.


Plug-in Development - HelloWorldOne/META-INF/MANIFEST.MF - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Plug-ins
HelloWorldOne
JRE System Library [JavaSE-1.6]
Plug-in Dependencies
src
icons
META-INF
MANIFEST.MF
build.properties
plugin.xml
HelloWorldOne
Overview
Overview Dependencies Runtime Extensions Extension Points Build MANIFEST.MF plugin.xml

General Information
This section describes general information about this plug-in.

ID:	HelloWorldOne
Version:	1.0.0.qualifier
Name:	HelloWorldOne
Provider:	Steve
Platform Filter:	
Activator:	helloworldone.Activator

☑ Activate this plug-in when one of its classes is loaded
☑ This plug-in is a singleton

Execution Environments
Specify the minimum execution environments required to run this plug-in.

Plug-in Content
The content of the plug-in is made up of two
Dependencies: lists all the plug-ins requ
plug-in's classpath to compile and run.
Runtime: lists the libraries that make up
in's runtime.

Extension / Extension Point Content
This plug-in may define extensions and exte
Extensions: declares contributions this
makes to the platform.
Extension Points: declares new functio
plug-in adds to the platform.

Testing
Test this plug-in by launching a separate Ecli
application:

Figure 4-8 Plug-in Overview view

Run the plug-in so that you can see exactly what it does. One of the great things about Eclipse is that after you have written your plug-in, it is easy to test it.

To run the plug-in, in the Package CICS Explorer view, right-click the HelloWorldOne project, and select **Run as** → **Eclipse Application**. You see that a completely new instance of Eclipse is started, with a new workspace. Close the Welcome window, and look closely at your new workspace. Can you see what has changed?

Hint: Look closely at the workspace menu bar and toolbar.

Figure 4-9 shows the workspace generated when you test the HelloWorldOne sample. This Hello world example adds a new item “Sample Menu” to the workspace menu bar, and a new icon to the workspace toolbar (because no specific icon is referenced in the sample, it uses the default Eclipse icon).

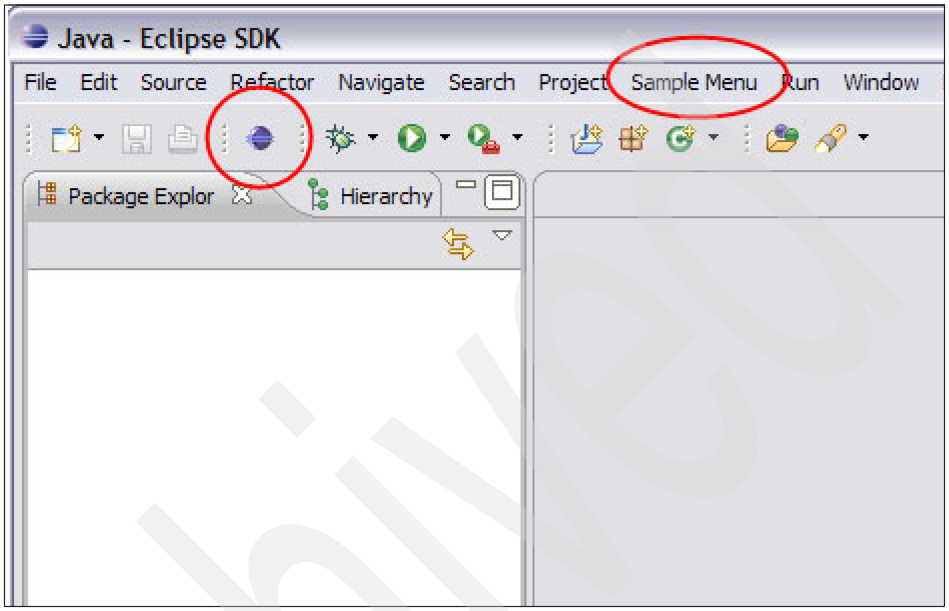
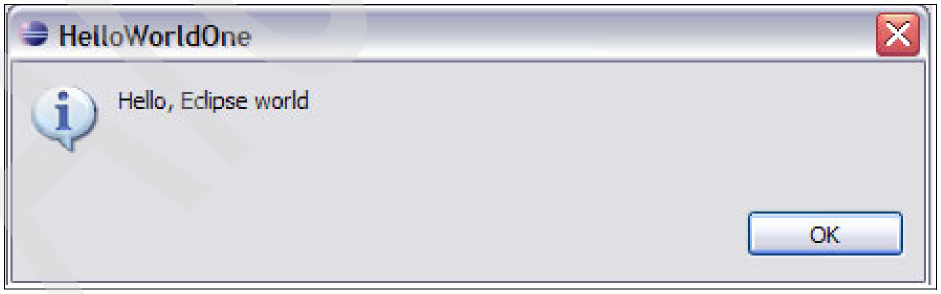


Figure 4-9 HelloWorldOne test

If you now left-click the icon, or left-click **Sample Menu** and select **Sample Action**, a window appears in the workspace with our “Hello World” message (see Figure 4-10).



Hello, Eclipse world

Figure 4-10 Hello World One Sample Action

If you create and run projects using the other Hello World templates provided by Eclipse, you see other examples of user interaction.

4.2.2 What is in our HelloWorld plug-in project?

The manifest editor displays the Overview view of the plug-in in the upper central panel. This gives a user-friendly version of the contents of the `manifest.mf` and `plugin.xml` files. Double-click the Overview tab to get a full-panel display. Click the MANIFEST.MF tab following this view (Example 4-2), and you see the raw file (which is less verbose than the overview).

Example 4-2 HelloWorldOne manifest.mf

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorldOne
Bundle-SymbolicName: HelloWorldOne; singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: helloworldone.Activator
Bundle-Vendor: Steve
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-ActivationPolicy: lazy
```

The manifest.mf contains information about the origin, version, and run-time environment of the plug-in.

The details about the dependencies of our plug-in, and its relationship to other plug-ins is provided in the `plug-in.xml` file. Click the plugin.xml tab, and you see the `HelloWorldOneplug-in.xml` file (Example 4-3).

Example 4-3 HelloWorldOneplug-in.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
 <extension point="org.eclipse.ui.actionSets">
      <actionSet
            label="Sample Action Set"
            visible="true"
            id="HelloWorldOne.actionSet">
         <menu label="Sample &amp;Menu" id="sampleMenu">
            <separator name="sampleGroup">
            </separator>
         </menu>
         <action label="&amp;Sample Action"
               icon="icons/sample.gif"
               class="helloworldone.actions.SampleAction"
```

```
                  tooltip="Hello, Eclipse world"
                  menubarPath="sampleMenu/sampleGroup"
                  toolbarPath="sampleGroup"
                  id="helloworldone.actions.SampleAction">
            </action>
         </actionSet>
      </extension>
</plugin>
```

The extension tag tells Eclipse the Java class which this plug-in is extending (in this case, the org.eclipse.ui.actionSets class). This is the class that manages the Eclipse menu bar and toolbar. Our HelloWorldOne class is going to interact with these two parts of the Eclipse workspace (you can also see this information by clicking the Extensions tab). Within the extension tag are more details about the additions to the menu bar and toolbar (the icon to appear on the toolbar, the name of the new tab in the menu bar, and associated information). Most important of all, the class parameter tells Eclipse which Java class to invoke to perform our new action (helloworldone.actions.SampleAction).

Click the Dependencies tab underneath this view and you see that your new plug-in relies on two packages that are provided by the Eclipse base platform:

- classes org.eclipse.ui
- org.eclipse.core.runtime

These two packages correspond to those specified on the Require-Bundle parameter in the `manifest.mf` file.

If you click the Extension Points tab, you see that there are none. This is because our plug-in does not expose any interfaces that are to be used by other plug-ins.

Double-click the Extension Points tab to return to the Plug-in perspective default view showing the Package CICS Explorer view.

HelloWorldOne Java implementation

You can see that as well as creating the `manifest.mf` and `plugin.xml` code, the wizard has generated other objects, including the Java packages that implement the plug-in. In fact, two Java packages have been created; the helloworldone package contains the Activator.java class, and the helloworldone.actions package contains the SampleAction.java class, as shown in Figure 4-11 on page 75.

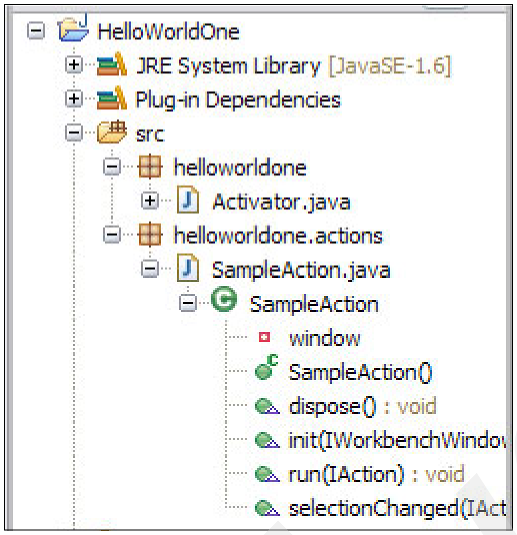

HelloWorldOne
JRE System Library [JavaSE-1.6]
Plug-in Dependencies
src
helloworldone
Activator.java
helloworldone.actions
SampleAction.java
SampleAction
window
SampleAction()
dispose() : void
init(IWorkbenchWindo
run(IAction) : void
selectionChanged(IAct

Figure 4-11 HelloWorldOne Java Packages

The Activator.java object has been created because during the creation of our Plug-in project, we left the “Generate an activator” check box selected. The Activator.java object is used to allow the programmer to execute any actions that might need to be performed during the life cycle of our SampleAction object (for example, there might be a need to execute code at the moment this object is started, and at the moment it is stopped). It is also invoked to allow a caller to inquire on the icon associated with our SampleAction object. The generated object is effectively a no-op for the methods, which you modify to add the start and stop functionality you want to implement.

Because our HelloWorldOne sample does not require any work to be done when it is started and stopped, we could have chosen to clear the “Generate an Activator” check box. In this case, the Activator.java class would not have been generated for our plug-in. It still executes without error.

Click the SampleAction.java file, and you are presented with the Java source code for our HelloWorldone plug-in. Double-click the SampleAction.java tag to get a full panel view of the code.

The generated code can be broken down into the parts shown in Example 4-4.

Example 4-4 HelloWorldOne package and import statements

```
package helloworldone.actions;

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

Because you are using a function provided by Eclipse, you need to import the necessary packages. These packages are all related to interacting with the Eclipse user interface. When you start using the function provided by CICS Explorer, you need to add import statements for packages supplied by CICS Explorer, rather than by Eclipse.

IAction, ISelection, IworkbencWindow, and IworkbenchWindowActionDelegate are interface classes. This means they define the interfaces used to communicate between the Eclipse user interface component, and your helloworldone component. As the implementer of helloworldone, it is your responsibility to provide the code that implements the business function to be executed when your action is invoked, and return the expected output.

MessageDialog is the class used to interact with the user by sending “hello World” to the panel. Our code invokes this plug-in, which is provided by Eclipse.

In Example 4-5 we see the class definition and constructor for our class. SampleAction implements the IWorkbenchWindowActionDelegate interface. This is how the Eclipse plug-in architecture works. To create a new Eclipse entity such as a perspective, a view, or (as in this case) an action, Eclipse creates a proxy object and delegates responsibility for executing the requested method to the associated object (as defined in the `plugin.xml` file). Make sure that all the necessary code is in place.

Example 4-5 Class definition and constructor for SampleAction

```
public class SampleAction implements IWorkbenchWindowActionDelegate {
   private IWorkbenchWindow window;
   /**
    * The constructor.
    */
   public SampleAction() {
   }
```

Methods associated with our HelloWorldOne class

The following methods (as defined in the IActionSet interface class) have been generated for our SampleAction class:

- `init`

  This method is invoked when your ActionSet becomes active. It gains access to the Workspace window, and saves it for later use. See Example 4-6 for example.

Example 4-6 Init method of HelloWorldOne

```
   /**
    * We cache window object to be able to
    * provide parent shell for the message dialog.
    * @see IWorkbenchWindowActionDelegate#init
    */
   public void init(IWorkbenchWindow window) {
      this.window = window;
   }
```

- `run`

  The user has clicked on the icon or chosen this action from the menu bar. The window object was initialized by the `init` method. We invoke its `getShell` method to give ourselves the ability to write to the panel.

  We use the MessageDialog class supplied by Eclipse to open an information window and put out our Hello World text message. See Example 4-7.

Example 4-7 Run method of HelloWorldOne

```
/**
    * The action has been activated. The argument of the
    * method represents the 'real' action sitting
    * in the workbench UI.
    * @see IWorkbenchWindowActionDelegate#run
    */
   public void run(IAction action) {
      MessageDialog.openInformation(
         window.getShell(),
         "HelloWorldOne",
         "Hello, Eclipse world");
```

- `selectionChanged`

  The `selectionChanged` method is a no-op in this example. The selectionChanged method is invoked when the cursor is clicked on our action. Prior to running the `init` method.u rpr. having selected this action, the user changes his mind, allowing you to undo anything done in the `Run` method.

- `dispose`

  The `dispose` method is a no-op in this example. The `dispose` method is invoked to clean up when the object is being destroyed.

Java implementation summary

We have seen that Eclipse can generate useful Java code templates. Now we see how we can take those templates and fill them out with more functionality, and then we can start building messages with CICS content and putting them out to the panel.

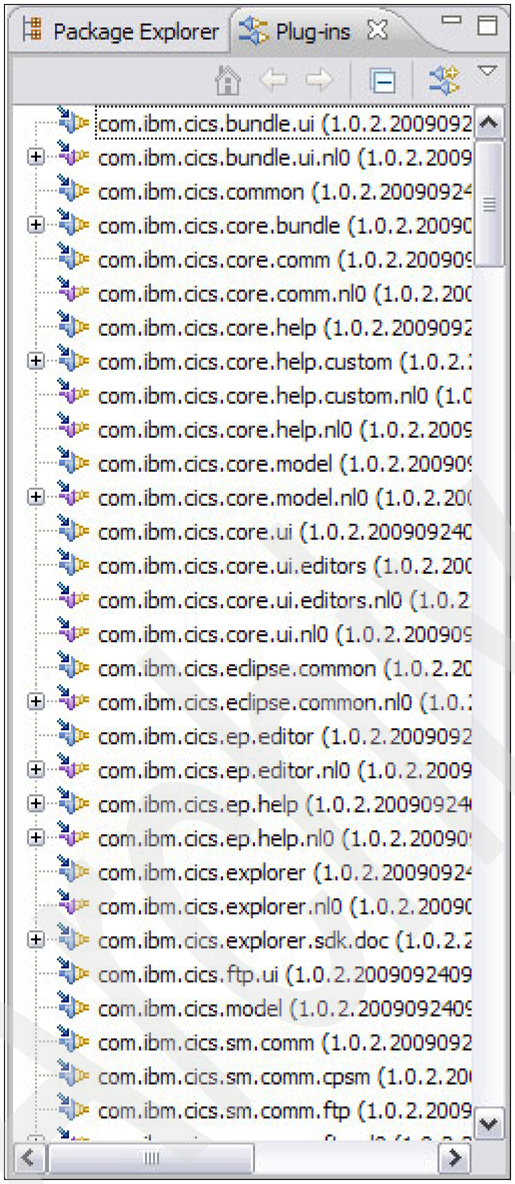
4.3 Creating your first CICS Explorer plug-in

Now that we have seen how the wizards provided by Eclipse can be used to generate template code, we follow the process detailed in 4.2, "An Eclipse RCP "Hello World" plug-in" on page 66, but we add CICS Explorer code to give our HelloWorld a more CICS feel.

4.3.1 Creating your CICS Explorer plug-in project

Start the Eclipse SDK that has the CICS Explorer installed into it. Click **Window** → **Open Perspective** → **Other** and select the Plug-in Developer perspective. This is the perspective with which you are working.

If this is not already shown, click the Plug-in view in the left pane. Scroll through the list. You see plug-ins that are provided in the CICS Explorer SDK, as shown in Figure 4-12 on page 79.

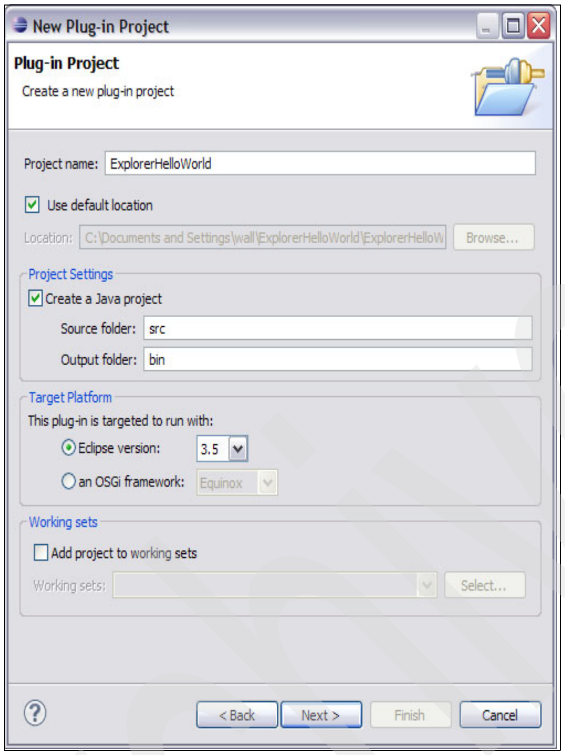

Package Explorer Plug-ins

```
com.ibm.cics.bundle.ui (1.0.2.2009092
com.ibm.cics.bundle.ui.nl0 (1.0.2.2009
com.ibm.cics.common (1.0.2.20090924
com.ibm.cics.core.bundle (1.0.2.20090
com.ibm.cics.core.comm (1.0.2.20090
com.ibm.cics.core.comm.nl0 (1.0.2.200
com.ibm.cics.core.help (1.0.2.2009092
com.ibm.cics.core.help.custom (1.0.2.
com.ibm.cics.core.help.custom.nl0 (1.0
com.ibm.cics.core.help.nl0 (1.0.2.2009
com.ibm.cics.core.model (1.0.2.20090
com.ibm.cics.core.model.nl0 (1.0.2.200
com.ibm.cics.core.ui (1.0.2.200909240
com.ibm.cics.core.ui.editors (1.0.2.200
com.ibm.cics.core.ui.editors.nl0 (1.0.2
com.ibm.cics.core.ui.nl0 (1.0.2.20090
com.ibm.cics.eclipse.common (1.0.2.20
com.ibm.cics.eclipse.common.nl0 (1.0.
com.ibm.cics.ep.editor (1.0.2.2009092
com.ibm.cics.ep.editor.nl0 (1.0.2.2009
com.ibm.cics.ep.help (1.0.2.20090924
com.ibm.cics.ep.help.nl0 (1.0.2.20090
com.ibm.cics.explorer (1.0.2.20090924
com.ibm.cics.explorer.nl0 (1.0.2.20090
com.ibm.cics.explorer.sdk.doc (1.0.2.2
com.ibm.cics.ftp.ui (1.0.2.2009092409
com.ibm.cics.model (1.0.2.200909240
com.ibm.cics.sm.comm (1.0.2.2009092
com.ibm.cics.sm.comm.cpsm (1.0.2.20
com.ibm.cics.sm.comm.ftp (1.0.2.2009
```

Figure 4-12 CICS Explorer content in Plug-in view

Click the Package Explorer tab to return to the Package Explorer view. It is currently empty.

To create your new plug-in package, click **File** → **New** → **Project** to start the New Plug-in Project wizard. Select **Plug-in Project a** (not Plug-in-Development), and click **Next**. You are presented with the window shown in Figure 4-13.


New Plug-in Project

Plug-in Project
Create a new plug-in project

Project name: ExplorerHelloWorld
[✓] Use default location
Location: C:\Documents and Settings\wall\ExplorerHelloWorld\ExplorerHelloW

Project Settings
[✓] Create a Java project
Source folder: src
Output folder: bin

Target Platform
This plug-in is targeted to run with:
(●) Eclipse version: 3.5
( ) an OSGi framework: Equinox

Working sets
[ ] Add project to working sets
Working sets:

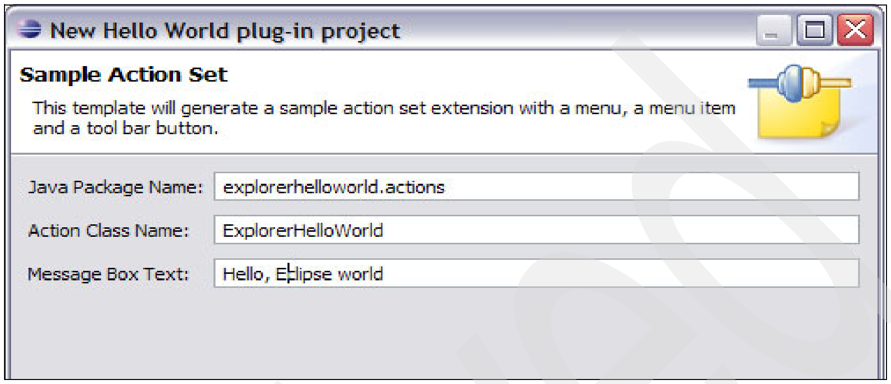
Figure 4-13 New Plug-in Project wizard

You need to give the new project a name. We called ours “CICS Explorer HelloWorld.” The other values can remain with their default values. Click **Next**.

The next window in the wizard asks you to give more details about the plug-in content. As we have already seen, you do not always need an Activator class. Clear the box so that no Activator class is generated for your new plug-in. Allow the other settings to default. You want to generate a plug-in that interfaces with the user, and you do not want to create a new Eclipse RCP application. Click **Next**.

On the next panel, where you are presented with a list of plug-in samples and templates, select the same Hello, World template used in 4.2, “An Eclipse RCP “Hello World” plug-in” on page 66. This time, we are going add CICS Explorer functionality to the generated code.

The wizard proceeds with the creation of the HelloWorld plug-in, and presents you with a panel summarizing the plug-in that you are creating (see Figure 4-14). Change the Action Class Name to `CICS Explorer HelloWorld` and click **Finish** to create the plug-in (we are going to add code to change the Message Box Text).



New Hello World plug-in project

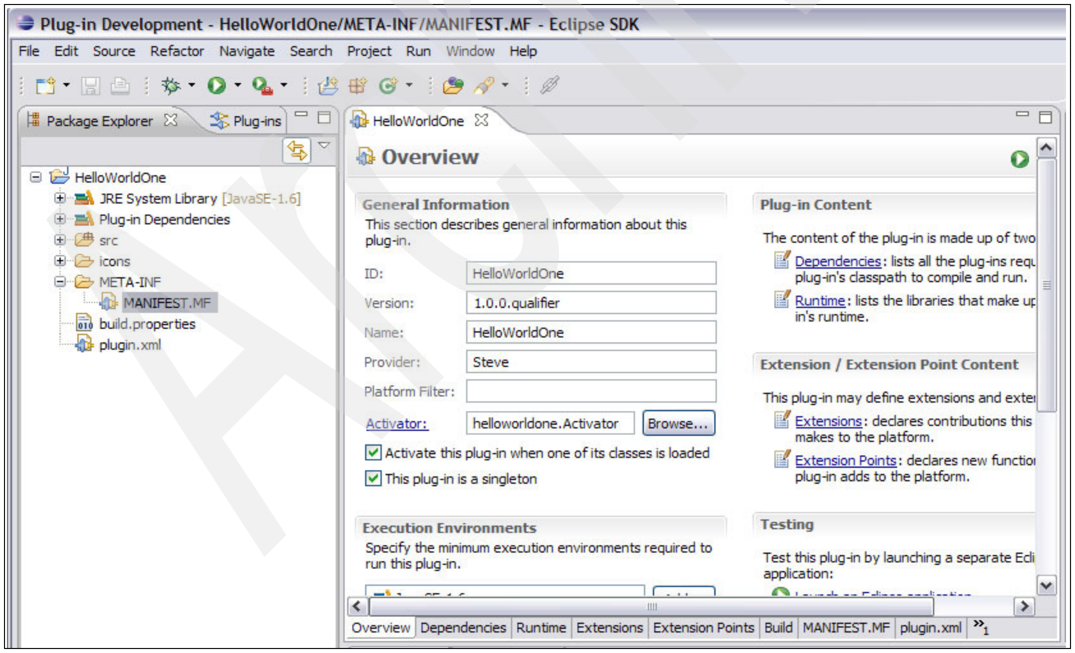
Sample Action Set

This template will generate a sample action set extension with a menu, a menu item and a tool bar button.

Java Package Name:	explorerhelloworld.actions
Action Class Name:	ExplorerHelloWorld
Message Box Text:	Hello, Eclipse world

Figure 4-14 CICS Explorer Hello World plug-in details

After you have clicked **Finish** in the New Plug-in Project wizard, the plug-in is created. In the Package Explorer view on the left side, you can see the various objects that have been created. See Figure 4-15.



Plug-in Development - HelloWorldOne/META-INF/MANIFEST.MF - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | Plug-ins

```
HelloWorldOne
  JRE System Library [JavaSE-1.6]
  Plug-in Dependencies
  src
  icons
  META-INF
    MANIFEST.MF
  build.properties
  plugin.xml
```

HelloWorldOne

Overview

General Information

This section describes general information about this plug-in.

ID:	HelloWorldOne
Version:	1.0.0.qualifier
Name:	HelloWorldOne
Provider:	Steve
Platform Filter:	
Activator:	helloworldone.Activator

☑ Activate this plug-in when one of its classes is loaded

☑ This plug-in is a singleton

Execution Environments

Specify the minimum execution environments required to run this plug-in.

Plug-in Content

The content of the plug-in is made up of two

Dependencies: lists all the plug-ins requ plug-in's classpath to compile and run.

Runtime: lists the libraries that make up in's runtime.

Extension / Extension Point Content

This plug-in may define extensions and exte

Extensions: declares contributions this makes to the platform.

Extension Points: declares new functio plug-in adds to the platform.

Testing

Test this plug-in by launching a separate Ecl application:

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml

Figure 4-15 CICS Explorer HelloWorld Plug-in Overview view

At this stage, the only difference between our CICS Explorer HelloWorld plug-in and the one we created in4.2, “An Eclipse RCP “Hello World” plug-in” on page 66 is the name of the generated package file and Java class class file.

4.3.2 Adding CICS Explorer to plug-in dependencies

You are going to use Java code provided by the CICS Explorer SDK, so you need to tell Eclipse that your ExplorerHelloWorld project has dependencies on plug-ins supplied by the CICS Explorer SDK. Do this by clicking the Dependencies tab in the central pane at the bottom of the Overview Editor .

Click **Add** in the Imported Packages pane on the right side of the Dependencies view, and add the following packages:

- com.ibm.cics.core.comm
- com.ibm.cics.core.model
- com.ibm.cics.core.ui
- com.ibm.cics.model

4.3.3 Adding CICS Explorer code to the Eclipse Template code

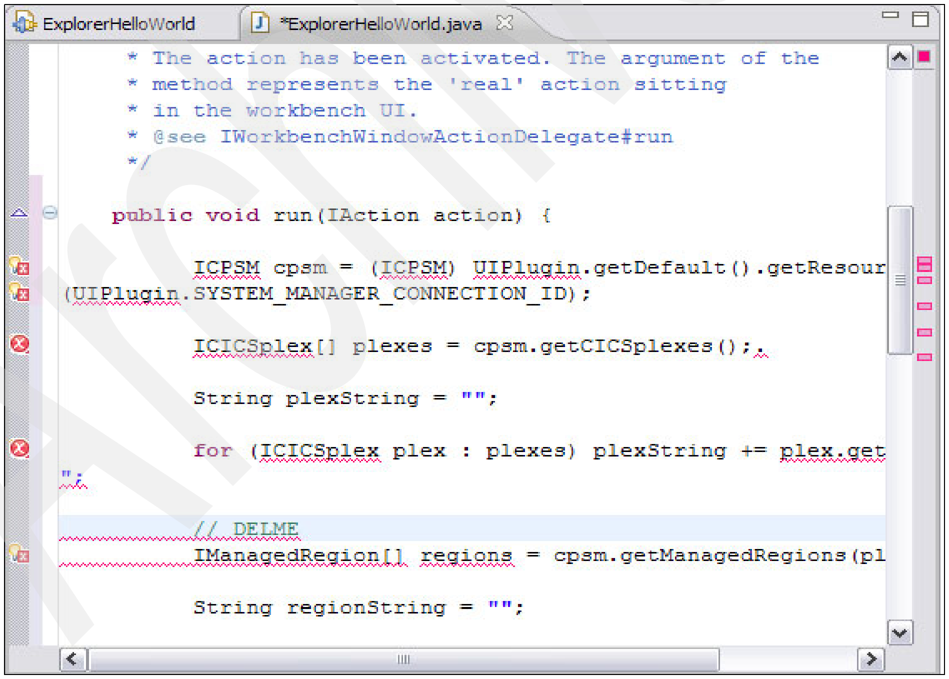
Now you are going to modify the code in the generated Java code to use functionality provided by the CICS Explorer SDK. Replace the supplied `run` method of your ExplorerHelloWorld class with the one shown in Example 4-8.

Example 4-8   ExplorerHelloWorld run method

```
public void run(IAction action) {

            ICPSM cpsm = (ICPSM)
UIPlugin.getDefault().getResourceManager
(UIPlugin.SYSTEM_MANAGER_CONNECTION_ID);

            ICICSplex[] plexes = cpsm.getCICSplexes();.

            String plexString = "";

            for (ICICSplex plex : plexes) plexString += plex.getName()
+ "
";

            // DELME
            IManagedRegion[] regions =
cpsm.getManagedRegions(plexes[0]);
```

```
            String regionString = "";

            for (IManagedRegion region : regions) regionString += 
region.getName() + " ";


            MessageDialog.openInformation(
                  window.getShell(),
                  "Hello_World",
                  "Hello" +
                  "\nYou are logged in to CPSM as " + cpsm.getUserID() 
+ 
                  "\nYour server name is " + cpsm.getName() +
                "\nYour CICSplex name is " + plexString +
              "\nYour regions are is " + regionString);
```

After you have copied the code in Example 4-8 on page 82 into the editor in Eclipse, you see that there are now a number of errors signalled by Eclipse, as shown in Figure 4-16.



```
         * The action has been activated. The argument of the
         * method represents the 'real' action sitting
         * in the workbench UI.
         * @see IWorkbenchWindowActionDelegate#run
         */

        public void run(IAction action) {

            ICPSM cpsm = (ICPSM) UIPlugin.getDefault().getResour
(UIPlugin.SYSTEM_MANAGER_CONNECTION_ID);

            ICICSplex[] plexes = cpsm.getCICSplexes();.

            String plexString = "";

            for (ICICSplex plex : plexes) plexString += plex.get
";

            // DELME
            IManagedRegion[] regions = cpsm.getManagedRegions(pl

            String regionString = "";
```

Figure 4-16   ExplorerHelloWorld dependencies

Your ExplorerHelloWorld plug-in now has dependencies on other plug-ins. You must tell Eclipse which plug-ins or packages are to be used to resolve those dependencies.

Tip: Follow this suggestion to avoid spending time trying to map CICS Explorer objects to their appropriate packages.

You are going to be working with CICS Explorer objects that are scattered across a large number of CICS Explorer-supplied plug-ins. Eclipse provides the Navigate facility to find the Java package containing an object that you want to use. However, you have to tell Eclipse that you want the CICS Explorer plug-ins to be included in its searches. Click the Plug-in tab to show the Plug-in view on the left pane, then click the last icon on the toolbar of the Plug-in view in the left pane (As you hold the cursor over this icon you see the text “Add All Plug-ins to Java Search.” See Figure 4-17). Eclipse now includes the CICS Explorer plug-ins in its searches for Java artifacts.)

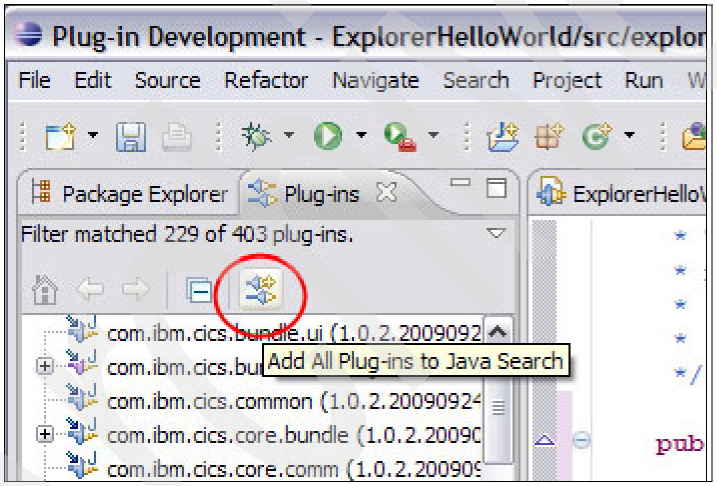
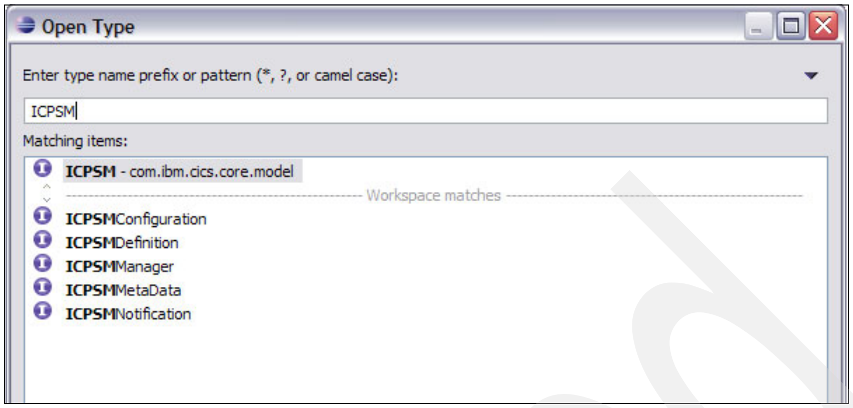

Plug-in Development - ExplorerHelloWorld/src/explor
File
Edit
Source
Refactor
Navigate
Search
Project
Run
W
Package Explorer
Plug-ins
ExplorerHello
Filter matched 229 of 403 plug-ins.
com.ibm.cics.bundle.ui (1.0.2.2009092
com.ibm.cics.bu
Add All Plug-ins to Java Search
com.ibm.cics.common (1.0.2.20090924
com.ibm.cics.core.bundle (1.0.2.20090
com.ibm.cics.core.comm (1.0.2.20090
pub

Figure 4-17 Add All Plug-ins to Java Search

You only have to add all plug-ins to a Java search in this manner when you have added more plug-ins.

Return to the editor view containing your ExplorerHelloWorld.java file, and identify the first Java statement that is in error. It appears to be the ICPSM object that is causing the problem.

Click the Navigate tab on the workspace action bar, and select **Open Type**. The panel shown Figure 4-18 on page 85 is displayed, showing that the ICPSM object is to be found in package com.ibm.core.model.


Open Type

Enter type name prefix or pattern (*, ?, or camel case):

ICPSM

Matching items:

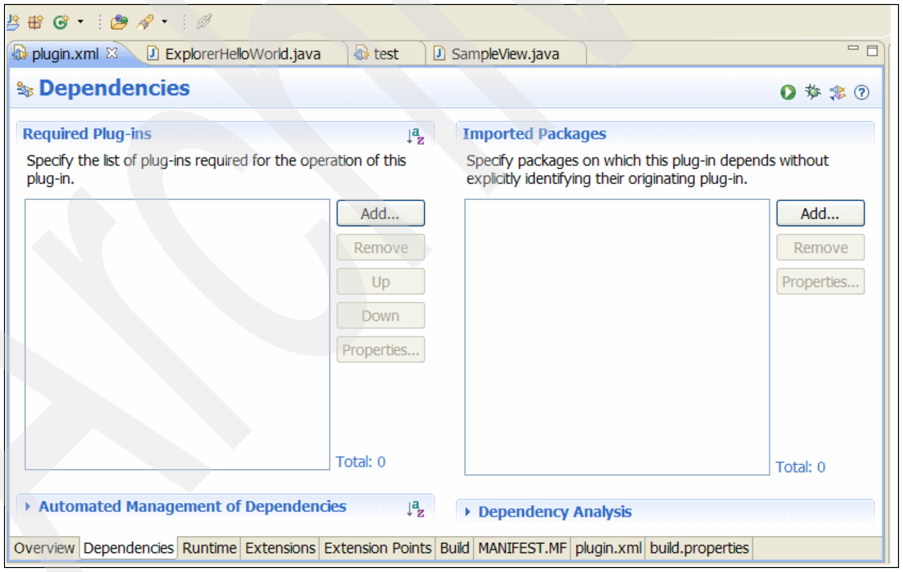
ICPSM - com.ibm.cics.core.model

Workspace matches

ICPSMConfiguration
ICPSMDefinition
ICPSMManager
ICPSMMetaData
ICPSMNotification

Figure 4-18 ICPSM object is to be found in package com.ibm.core.model.

You need to declare package com.ibm.core.model as an external dependency. Do this by returning to the Manifest Editor view, which was opened when we created our ExplorerHelloWorld project. Double-click the Dependencies tab at the bottom of the central pane, and then double-click the tab to get the panel view shown in Figure 4-19.


plugin.xml | ExplorerHelloWorld.java | test | SampleView.java

Dependencies

Required Plug-ins

Specify the list of plug-ins required for the operation of this plug-in.

Total: 0

Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

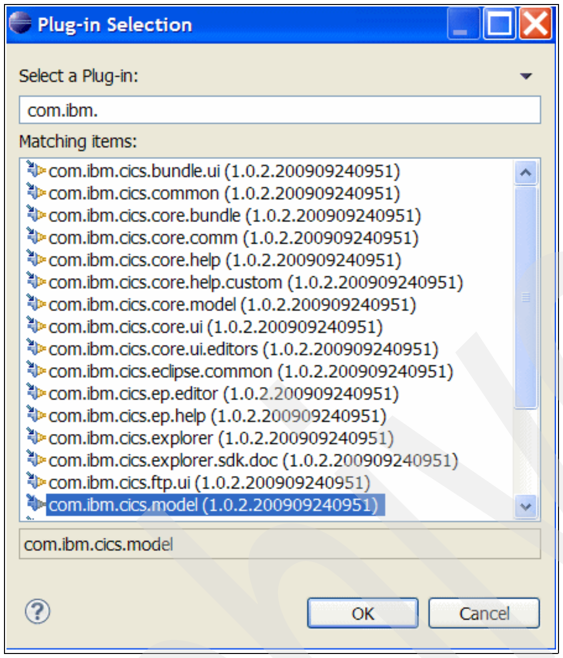
Total: 0

Automated Management of Dependencies

Dependency Analysis

Figure 4-19 Declare package com.ibm.core as dependency

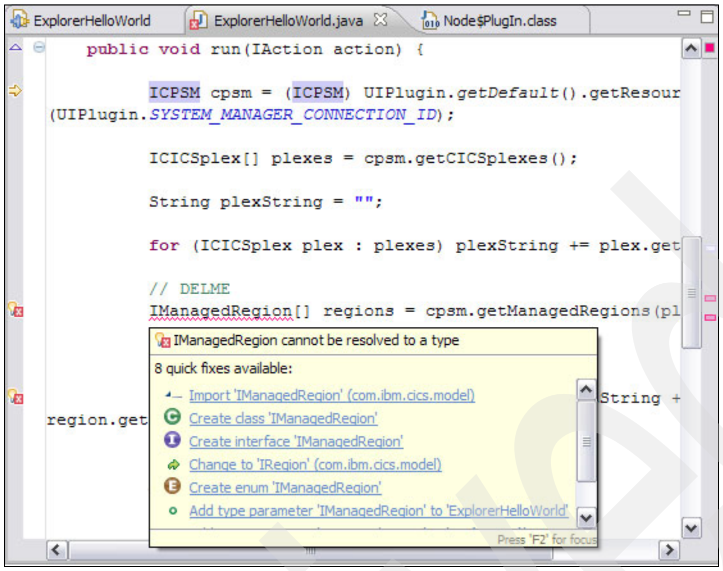
Click **Add** in the Required Plug-ins pane. The Plug-in selection window displays. Enter the name of the package you found with navigator. (The window prompts you with a list of suitable matches as you type in the characters) Click **OK** to add this plug-in to the list of dependencies. See Figure 4-20.



```
com.ibm.
com.ibm.cics.bundle.ui (1.0.2.200909240951)
com.ibm.cics.common (1.0.2.200909240951)
com.ibm.cics.core.bundle (1.0.2.200909240951)
com.ibm.cics.core.comm (1.0.2.200909240951)
com.ibm.cics.core.help (1.0.2.200909240951)
com.ibm.cics.core.help.custom (1.0.2.200909240951)
com.ibm.cics.core.model (1.0.2.200909240951)
com.ibm.cics.core.ui (1.0.2.200909240951)
com.ibm.cics.core.ui.editors (1.0.2.200909240951)
com.ibm.cics.eclipse.common (1.0.2.200909240951)
com.ibm.cics.ep.editor (1.0.2.200909240951)
com.ibm.cics.ep.help (1.0.2.200909240951)
com.ibm.cics.explorer (1.0.2.200909240951)
com.ibm.cics.explorer.sdk.doc (1.0.2.200909240951)
com.ibm.cics.ftp.ui (1.0.2.200909240951)
com.ibm.cics.model (1.0.2.200909240951)
com.ibm.cics.model
```

Figure 4-20 Plug-in added

You need to add import statements for the CICS Explorer packages you are using to the ExplorerHelloWorld.java file. When you have the correct dependencies in place, you can resolve missing imports by placing the cursor over unresolved objects. Eclipse offers to find the missing import for you. Put the cursor on the Import option in the list, and Eclipse adds the required import statement. See Figure 4-21 on page 87.



```
    public void run(IAction action) {

            ICPSM cpsm = (ICPSM) UIPlugin.getDefault().getResour
(UIPlugin.SYSTEM_MANAGER_CONNECTION_ID);

            ICICSplex[] plexes = cpsm.getCICSplexes();

            String plexString = "";

            for (ICICSplex plex : plexes) plexString += plex.get

            // DELME
            IManagedRegion[] regions = cpsm.getManagedRegions(pl
                                                         String +
region.get
```

```
IManagedRegion cannot be resolved to a type
8 quick fixes available:
  Import 'IManagedRegion' (com.ibm.cics.model)
  Create class 'IManagedRegion'
  Create interface 'IManagedRegion'
  Change to 'IRegion' (com.ibm.cics.model)
  Create enum 'IManagedRegion'
  Add type parameter 'IManagedRegion' to 'ExplorerHelloWorld'
                                              Press 'F2' for focus
```

Figure 4-21 Using Eclipse to resolve missing objects

If you put the cursor over an object, and Eclipse cannot find it, then you need to add another dependency as described previously.

After you have successfully resolved all the missing objects, press Ctrl+S to save the changes to the workspace.

When you are ready to run the plug-in, in the Package CICS Explorer view, right-click the ExplorerHelloWorld project, and select **Run as** → **Eclipse. Application**. You see that a completely new instance of Eclipse is started, with a new workspace.

Figure 4-22 shows the workspace generated when you test the ExplorerHelloWorld sample. You can see that it looks much the same as our original HelloWorldOne example. The “Sample Menu” item is in the workspace menu bar, and the new icon is in the workspace toolbar.

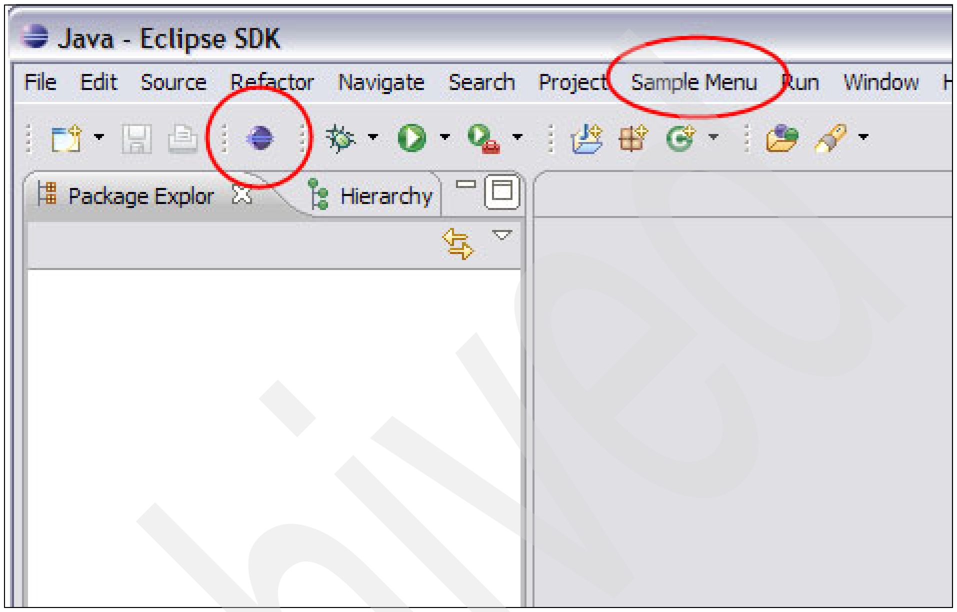
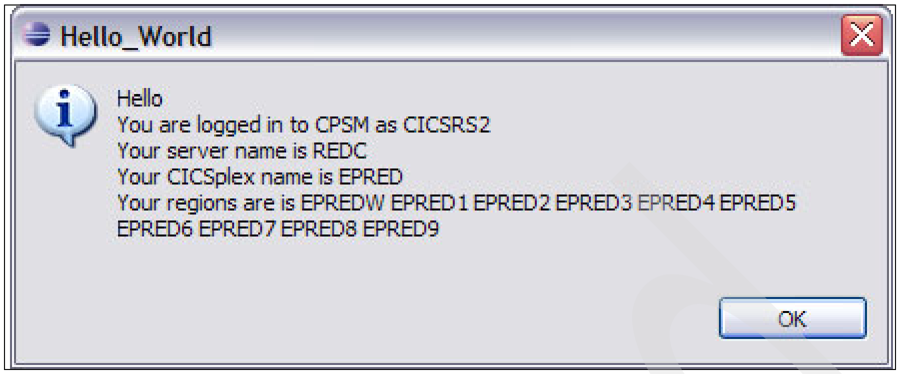


Figure 4-22   ExplorerHelloWorld Test

Because your ExplorerHelloWorld plug-in accesses CICSPlex SM, you need to make sure we are connected to CICSPlex SM before we try to run it. Click **Window** → **Open Perspective** → **Other** and select the CICS SM perspective. Configure the connection to your CICS WUI Server using the CMCI interface. When you have successfully connected to your CICS WUI Server you are ready to try out your new plug-in.

If you left-click the icon, or left-click **Sample Menu**, and select **Sample Action**, a window appears in the workspace (Figure 4-23 on page 89).

.



Hello_World

Hello
You are logged in to CPSM as CICSRS2
Your server name is REDC
Your CICSplex name is EPRED
Your regions are is EPREDW EPRED1 EPRED2 EPRED3 EPRED4 EPRED5
EPRED6 EPRED7 EPRED8 EPRED9

Figure 4-23 ExplorerHelloWorld Sample Action

Our plug-in has requested CICSPlex SM configuration information from the CICS WUI region. The server name is the SYSID of the CMAS to which the CICS WUI Server is connected, the rest is self-explanatory.

You can now close down the Eclipse in which your ExplorerHelloWorld is running.

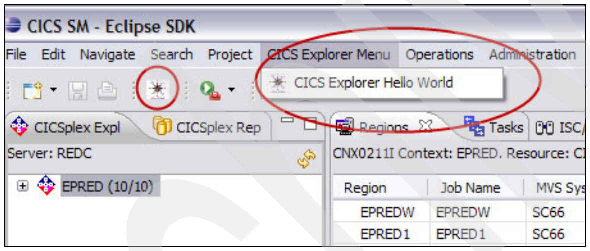
You can make the plug-in look a lot more CICS-centric by changing some of the information in the `plugin.xml` file. Return to the Package Explorer view of your development Eclipse, and double-click the `plugin.xml` file. to open the `plugin.xml` editor. Make the changes highlighted in red in Example 4-9, presse Ctrl+S to save the changes, and re-run the test.

Note: You need to copy an appropriate icon into the icons directory of your ExplorerHelloWorld project.

Example 4-9 Plug-in XML changes in red

```
<plugin>
<extension
         point="org.eclipse.ui.actionSets">
      <actionSet
            label="CICS Explorer Hello World"
            visible="true"
            id="ExplorerHelloWorld.actionSet">
         <menu
               label="CICS Explorer &amp;Menu"
               id="sampleMenu">
            <separator
                  name="sampleGroup">
            </separator>
         </menu>
         <action
               label="&amp;CICS Explorer Hello World"
               icon="icons/CICSExplorerIcon.GIF"
               class="explorerhelloworld.actions.ExplorerHelloWorld"
               tooltip="Hello, Eclipse world"
               menubarPath="sampleMenu/sampleGroup"
               toolbarPath="sampleGroup"
               id="explorerhelloworld.actions.ExplorerHelloWorld">
         </action>
      </actionSet>
   </extension>

</plugin>
```

When you re-run the test, you get a more CICS-centric action tab and icon (Figure 4-24), although the function has not changed.



CNX0211I Context: EPRED. Resource: C

Region	Job Name	MVS Sys
EPREDW	EPREDW	SC66
EPRED1	EPRED1	SC66

Figure 4-24 Customized ExplorerHelloWorld

4.3.4 CICS Explorer plug-in design consideration

When your ExplorerHelloWorld plug-in requested information from CICSPlex SM itself, there was a delay between requesting the action, and CICSPlex SM Hello World message appearing on the window. During this period you were not able to do anything in your Eclipse environment.

When you want to use the CICS Explorer classes to connect to the CICS WUI region, create a new thread specifically for executing the call to the WUI region. Then your workspace GUI thread can execute while you wait for the call to CICSPlex SM to complete. When the operation completes, the communication thread passes the results back to the workspace thread for display.

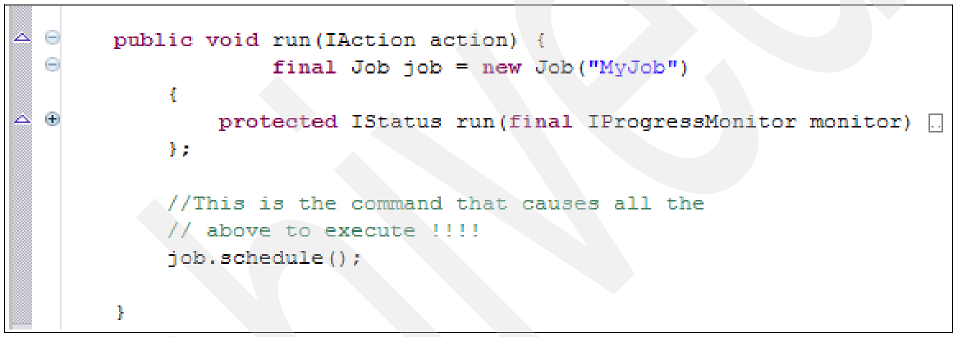
You are now going to modify ExplorerHelloWorld to work in this way. To achieve this you must insert inline class methods into the existing code, and modify the existing code to use them.

Move the variables accessed by the CPSM to be global across all methods of the class, as shown in Example 4-10.

Example 4-10 Move the variables accessed by the CPSM

```
private String plexString = "";
private String regionString = "";
```

Change the `run()` method of the ExplorerHelloWorld object as shown in Figure 4-25.



```
    public void run(IAction action) {
                final Job job = new Job("MyJob")
        {
            protected IStatus run(final IProgressMonitor monitor) 
        };

        //This is the command that causes all the
        // above to execute !!!!
        job.schedule();

    }
```

Figure 4-25 Change the run() method of our ExplorerHelloWorld object

The statement `final Job job = new Job(“MYJob”)` tells Eclipse that you are going to be running a piece of work `job` under a separate execution thread. This piece of work is executed when you invoke the `schedule` method of your job object `job.schedule`. You can see that there is more code between these two statements. The code between the two contains the inline `run()` method that contains the code to be executed on the separate thread. If you click the cross on the right side of the display, Eclipse expands the inline class and you see the code shown in Example 4-11 on page 91. We have highlighted the interesting parts of the text.

Example 4-11 Progress bar run method

```
      protected IStatus run(final IProgressMonitor monitor)
      {
         // Start the progress bar running
         monitor.beginTask("MyJob", IProgressMonitor.UNKNOWN);

         // Do the CPSM stuff here
```

```
        // This is the big CPSM call that gives us the CPSM
        // context we can use to invoke CPSM !!!!
        final ICPSM cpsm = UIPlugin.getDefault().getCPSM();

        // Get the names of the CICSPlexes
        // that this WUI knows about
        ICICSplex[] plexes = cpsm.getCICSplexes();


        // Write out the CICSPlexes returned
        for (ICICSplex plex : plexes) plexString += plex.getName() + "
";

                // Ask CPSM for the CICS regions it knows about
                IManagedRegion[] regions =
cpsm.getManagedRegions(plexes[0]);

                // Write out the CICS region names returned
                for (IManagedRegion region : regions) regionString
+=
        region.getName() + "  ";

        //Tell the GUI thread that we want it to do some
        //stuff that we have implemented by an inline run()
        //method.
        Display.getDefault().asyncExec(new Runnable()
        {
             @Override
             public void run() {
                // TODO Auto-generated method stub
                // Moved the CPSM screen formatting to
                // be executed when we have grabbed back
                // the GUI thread.
                MessageDialog.openInformation(
                        window.getShell(),
                        "Hello_World",
                        "Hello" +
                        "\nYou are logged in to CPSM as " +
cpsm.getUserID() +
                        "\nYour server name is " + cpsm.getName() +
                      "\nYour CICSplex name is " + plexString +
                    "\nYour regions are " + regionString);
             }
        });
        monitor.done();

        return Status.OK_STATUS;
      }
    };
```

5

Extending CICS Explorer plug-ins

In this chapter we provide meaningful examples of plug-ins that can be coded to use IBM CICS Explorer in ways specific to a business or enterprise, or which allow the CICS data managed by CICS Explorer to be accessed in new and interesting ways (which are not possible with the standard CICS Explorer implementation). We look at the separate user interfaces which Eclipse provides to do this.

We take a step by step approach:

- Extending new CICS Explorer views
- Using a pop-up menu to access URIMap information
- Extending a toolbar to access URIMap information
- Extending a toolbar to add a filter based on user input
- Background process implementation
- Summary of extending functions
- Package all extending functions into a plug-in

5.1 Extending new CICS Explorer views

In this section we look at how we can extend new CICS Exploerer views.

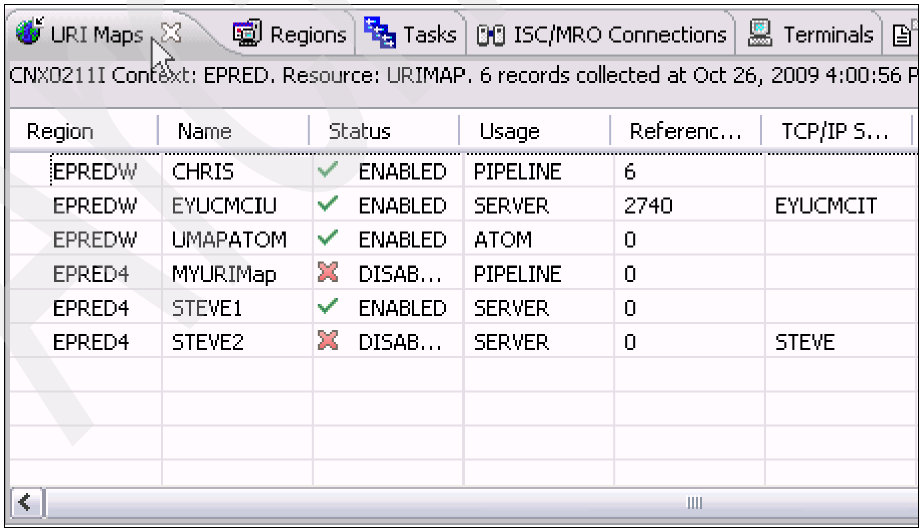
5.1.1 Extending the URIMap information provided by CICS Explorer

We provided details on developing a sampleView"in Chapter 4, "Writing a plug-in for CICS Explorer" on page 61. In this section, we describe how to modify a sampleView to be a more practical one. As a practical example, we apply URIMap resource information to the sampleView. A URIMap is CICS resource definition that maps the URI of an incoming HTTP or Web service request to how CICS processes the request. URIMap attributes have a hierarchical structure.

We use URIMap related information as an example of how to create a new view showing CICS resources. Our sample code in this chapter implements separate processes depending on the URIMap attribute: SERVER, CLIENT, PIPELINE, or ATOM as USAGE parameter. For example, if USAGE=SERVER, you can get and display PROGRAM information.

5.1.2 Specification of new view

The new CICS Explorer view provides detailed information about a URIMAP, which can be selected by clicking on it in the URIMap view. The URIMap view is provided by CICS Explorer. See Figure 5-1.

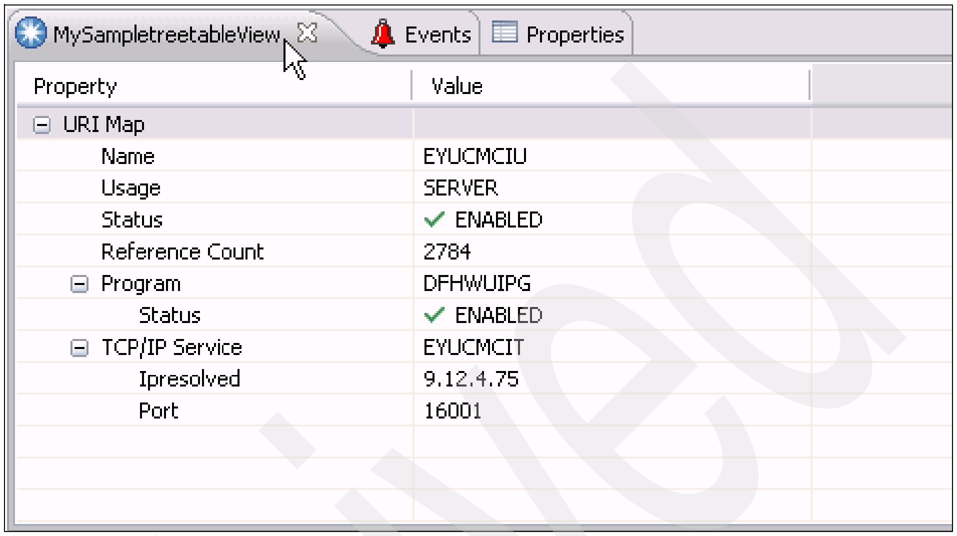


CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected at Oct 26, 2009 4:00:56 P

Region	Name	Status	Usage	Referenc...	TCP/IP S...
EPREDW	CHRIS	✓ ENABLED	PIPELINE	6	
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	2740	EYUCMCIT
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0	
EPRED4	MYURIMap	✗ DISAB...	PIPELINE	0	
EPRED4	STEVE1	✓ ENABLED	SERVER	0	
EPRED4	STEVE2	✗ DISAB...	SERVER	0	STEVE

Figure 5-1 URIMap view

To show the structured URIMAP information in our sampleView, we implement a tree view displaying hierarchical URIMap information that is not available in the CICS Explorer provided view. See Figure 5-2.



Property	Value
⊟ URI Map	
Name	EYUCMCIU
Usage	SERVER
Status	✓ ENABLED
Reference Count	2784
⊟ Program	DFHWUIPG
Status	✓ ENABLED
⊟ TCP/IP Service	EYUCMCIT
Ipresolved	9.12.4.75
Port	16001

Figure 5-2 Tree view for URIMap detailed information

It is also possible to implement automatic Get and Display functions in the view for the following additional related information depending on the selected URIMAP USAGE value:

- If URIMAP USAGE is SERVER|PIPELINE|ATOM, the view gets and displays TCPIPService instance information.
- If URIMAP USAGE is SERVER, the view gets and displays Program instance information.
- If URIMAP USAGE is PIPELINE, the view gets and displays Pipeline instance information.

You can show a sampleView from the menu bar by navigating to **Window** → **Show View** → **Other**. Then, click record in the URIMap view. The sampleView shows the URIMap detailed information.

5.1.3 Using our sample code

All code to get and display the URIMap detailed information is integrated into the `populateInformation` method for better extensibility. This view has the following process steps:

1. `createPartControl` method
   - Create the Tree View to display URIMap detailed information, as an initial process.
2. `populateInformation` method
   - Integrate a process to get and display target URIMap detailed information. This process is done only when the following two conditions are both satisfied, because both conditions are required to allow the target to get detailed URIMap information.
     - URIMap view is opened
     - One line in the URIMap is selected
3. `selectionListener` method
   - The left-click operation in the URIMap view causes this selectionListener event. You can invoke populateInformation here.

Example 5-1 Sample code: MySampletreetableView.java

```
package my_treetableview.views;

import helper.FilteredContext;

import java.net.URL;

import org.eclipse.core.runtime.FileLocator;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
import org.eclipse.core.runtime.Status;
import org.eclipse.core.runtime.jobs.Job;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.StructuredSelection;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.swt.widgets.TreeColumn;
import org.eclipse.swt.widgets.TreeItem;
import org.eclipse.ui.ISelectionListener;
import org.eclipse.ui.IViewSite;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.part.ViewPart;
import org.osgi.framework.Bundle;

import com.ibm.cics.core.model.CICSTypes;
import com.ibm.cics.core.model.Context;
import com.ibm.cics.core.model.ICPSM;
import com.ibm.cics.core.model.IResourcesModel;
import com.ibm.cics.core.ui.UIPlugin;
import com.ibm.cics.model.IAtomService;
import com.ibm.cics.model.ICICSObject;
import com.ibm.cics.model.ICICSType;
import com.ibm.cics.model.ICICSplex;
import com.ibm.cics.model.IPipeline;
import com.ibm.cics.model.IProgram;
import com.ibm.cics.model.ITCPIPService;
import com.ibm.cics.model.IURIMap;
import com.ibm.cics.model.IWebService;

public class MySampletreetableView extends ViewPart {

   private Tree tree;
   private TreeItem itemURIMapUsageKey, itemURIMapTCPIPServiceKey;
   private String uriMapString;
   private ICPSM cpsm;
   private IURIMap iurimap;
   private ITCPIPService itcpipservice;
   private IPipeline ipipeline;
   private IWebService iwebservice;
   private IProgram iprogram;
   private IAtomService iatomservice;
   private Context context;
   private IResourcesModel model;

   // Load images from icons directory
   private Bundle bundle = Platform.getBundle(my_treetableview.Activator.PLUGIN_ID);

   private URL imageEnableURL = FileLocator.find(bundle, new Path(
         "icons/ENABLED.gif"), null);
   private URL imageDisableURL = FileLocator.find(bundle, new Path(
         "icons/DISABLED.gif"), null);
   private URL imageErrorURL = FileLocator.find(bundle, new Path(
         "icons/Error.gif"), null);

   private Image imageEnable = ImageDescriptor.createFromURL(imageEnableURL)
         .createImage();
   private Image imageDisable = ImageDescriptor.createFromURL(imageDisableURL)
         .createImage();
   private Image imageError = ImageDescriptor.createFromURL(imageErrorURL)
         .createImage();

   private FilteredContext filteredContext;

   @Override
   public void createPartControl(Composite arg0) {
      // tree(table) definition
      tree = new Tree(arg0, SWT.BORDER | SWT.FULL_SELECTION);
      // add attribution
      tree.setHeaderVisible(true); // display header
      tree.setLinesVisible(true); // display lines
      // set headers
      String[] cols = { "Property", "Value" };
      for (int i = 0; i < cols.length; i++) {
         TreeColumn col = new TreeColumn(tree, SWT.LEFT);
         col.setText(cols[i]);
         col.setWidth(200);
      }
   }

   /** populateInformation
    * @param uriMap **/
   public void populateInformation(final String urimapstring) {
      // reserve argument(urimap)
      uriMapString = urimapstring;
      if (IO() == true) {
         GUI();
      } else {
         ngGUI();
      }
   }

   public void connectCPSM() {
      // get CPSM information
      cpsm = (ICPSM) UIPlugin.getDefault().getResourceManager(
            UIPlugin.SYSTEM_MANAGER_CONNECTION_ID);
```

```
      // get CICSplexes
      ICICSplex[] cicsplexes = cpsm.getCICSplexes();
      // set scopedContext(CPSM,CPSM)
      context = new Context(cicsplexes[0].getName());
      filteredContext = new FilteredContext(context);
   }

   private ICICSObject getCICSObject(ICICSType cicsType, String resourceName) {
      filteredContext.setFilterValues(cicsType.getNameAttribute(), resourceName);
      model = cpsm.getModel(cicsType, filteredContext);
      model.activate();
      model.maybeFetch(0, model.size());
      return model.size() > 0 ? model.get(0) : null;
   }

   public void getURIMapResource() {
      iurimap = (IURIMap) getCICSObject(CICSTypes.URIMap, uriMapString);
   }

   public void getTCPIPServiceResource() {
      itcpipservice = (ITCPIPService) getCICSObject(CICSTypes.TCPIPService,
            iurimap.getTCPIPService());
   }

   public void getProgramResource() {
      iprogram = (IProgram) getCICSObject(CICSTypes.Program, iurimap
            .getProgram());
   }

   public void getAtomServiceResource() {
      iatomservice = (IAtomService) getCICSObject(CICSTypes.AtomService,
            iurimap.getAtomservice());
   }

   public void getPipelineResource() {
      ipipeline = (IPipeline) getCICSObject(CICSTypes.Pipeline, iurimap
            .getPipeline());
   }

   public void getWebServiceResource() {
      iwebservice = (IWebService) getCICSObject(CICSTypes.WebService, iurimap
            .getWebService());
   }

   private boolean IO() {
      connectCPSM();
      getURIMapResource();
      if (model.size() == 0) { return false; }
      getTCPIPServiceResource();
      if (iurimap.getUsage().toString() == "SERVER") {
         getProgramResource();
      } else if (iurimap.getUsage().toString() == "CLIENT") {
         // ** no related resources **
      } else if (iurimap.getUsage().toString() == "ATOM") {
         getAtomServiceResource();
      } else if (iurimap.getUsage().toString() == "PIPELINE") {
         getPipelineResource();
         getWebServiceResource();
      }
      return true;
   }

   public void GUI() {
      // delete all items
      tree.removeAll();
      tree.setHeaderVisible(true); // display header
      tree.setLinesVisible(true); // display lines
      // URI Maps
      TreeItem itemURIMap = new TreeItem(tree, SWT.NULL);
      itemURIMap.setText(0, "URI Map");
      itemURIMap.setText(1, "");

      TreeItem itemURIMapName = new TreeItem(itemURIMap, SWT.NULL);
      itemURIMapName.setText(0, "Name");
      itemURIMapName.setText(1, iurimap.getName());

      TreeItem itemURIMapUsage = new TreeItem(itemURIMap, SWT.NULL);
      itemURIMapUsage.setText(0, "Usage");
      itemURIMapUsage.setText(1, iurimap.getUsage().toString());

      TreeItem itemURIMapStatus = new TreeItem(itemURIMap, SWT.NULL);
      itemURIMapStatus.setText(0, "Status");

      if (iurimap.getStatus().toString() == "ENABLED") {
         itemURIMapStatus.setImage(1, imageEnable);
      } else if (iurimap.getStatus().toString() == "DISABLED") {
         itemURIMapStatus.setImage(1, imageDisable);
      }
      itemURIMapStatus.setText(1, iurimap.getStatus().toString());
```

```
TreeItem itemURIMapReferenceCount = new TreeItem(itemURIMap, SWT.NULL);
itemURIMapReferenceCount.setText(0, "Reference Count");
itemURIMapReferenceCount.setText(1, iurimap.getReferenceCount().toString());

itemURIMapUsageKey = new TreeItem(itemURIMap, SWT.NULL);

if (iurimap.getUsage().toString() == "SERVER") {
   itemURIMapUsageKey.setText(0, "Program");
   itemURIMapUsageKey.setText(1, iurimap.getProgram());
} else if (iurimap.getUsage().toString() == "CLIENT") {
   itemURIMapUsageKey.setText(0, "Port");
   itemURIMapUsageKey.setText(1, iurimap.getPort().toString());
} else if (iurimap.getUsage().toString() == "ATOM") {
   itemURIMapUsageKey.setText(0, "Atomservice");
   itemURIMapUsageKey.setText(1, iurimap.getAtomservice());
} else if (iurimap.getUsage().toString() == "PIPELINE") {
   itemURIMapUsageKey.setText(0, "Pipeline");
   itemURIMapUsageKey.setText(1, iurimap.getPipeline());
}

if (iurimap.getUsage().toString() != "CLIENT") {
   itemURIMapTCPIPServiceKey = new TreeItem(itemURIMap, SWT.NULL);
   itemURIMapTCPIPServiceKey.setText(0, "TCP/IP Service");
   itemURIMapTCPIPServiceKey.setText(1, iurimap.getTCPIPService());
}

// if (tcpipserviceModel.size() > 0){
if (itcpipservice != null) {
   TreeItem itemTCPIPServiceIPResolved = new TreeItem(
         itemURIMapTCPIPServiceKey, SWT.NULL);
   itemTCPIPServiceIPResolved.setText(0, "Ipresolved");
   itemTCPIPServiceIPResolved.setText(1, itcpipservice.getIpresolved());

   TreeItem itemTCPIPServicePort = new TreeItem(
         itemURIMapTCPIPServiceKey, SWT.NULL);
   itemTCPIPServicePort.setText(0, "Port");
   itemTCPIPServicePort.setText(1, itcpipservice.getPort().toString());
}

// items related with URIMap
if (iurimap.getUsage().toString() == "SERVER") {
   if (iprogram != null) {
      // Program resource
      TreeItem itemProgramStatus = new TreeItem(itemURIMapUsageKey, SWT.NULL);
      itemProgramStatus.setText(0, "Status");
      if (iprogram.getStatus().toString() == "ENABLED") {
         itemProgramStatus.setImage(1, imageEnable);
      } else if (iprogram.getStatus().toString() == "DISABLED") {
         itemProgramStatus.setImage(1, imageDisable);
      }
      itemProgramStatus.setText(1, iprogram.getStatus().toString());
   }
} else if (iurimap.getUsage().toString() == "CLIENT") {
} else if (iurimap.getUsage().toString() == "ATOM") {
   if (iatomservice != null) {
      // Atomservice resource
      TreeItem itemAtomservice = new TreeItem(itemURIMapUsageKey, SWT.NULL);
      itemAtomservice.setText(0, "Atomservice");
      itemAtomservice.setText(1, "");

      TreeItem itemAtomserviceEnablestatus = new TreeItem(
            itemAtomservice, SWT.NULL);
      itemAtomserviceEnablestatus.setText(0, "Status");
      if (iatomservice.getEnablestatus().toString() == "ENABLED") {
         itemAtomserviceEnablestatus.setImage(1, imageEnable);
      } else if (iatomservice.getEnablestatus().toString() == "DISABLED") {
         itemAtomserviceEnablestatus.setImage(1, imageDisable);
      }
      itemAtomserviceEnablestatus.setText(1, iatomservice
            .getEnablestatus().toString());
   }
} else if (iurimap.getUsage().toString() == "PIPELINE") {
   if (ipipeline != null) {
      // Pipeline resource
      TreeItem itemPipelineStatus = new TreeItem(itemURIMapUsageKey, SWT.NULL);
      itemPipelineStatus.setText(0, "Status");
      if (ipipeline.getStatus().toString() == "ENABLED") {
         itemPipelineStatus.setImage(1, imageEnable);
      } else if (ipipeline.getStatus().toString() == "DISABLED") {
         itemPipelineStatus.setImage(1, imageDisable);
      }
      itemPipelineStatus.setText(1, ipipeline.getStatus().toString());

      TreeItem itemURIMapWebService = new TreeItem(itemURIMap, SWT.NULL);
      itemURIMapWebService.setText(0, "WebService");
      itemURIMapWebService.setText(1, iurimap.getWebService());

      if (iwebservice != null) {
         // Pipeline resource
         TreeItem itemWebServiceState = new TreeItem(
               itemURIMapWebService, SWT.NULL);
         itemWebServiceState.setText(0, "State");
```

```
               if (iwebservice.getState().toString() == "ENABLED") {
                  itemWebServiceState.setImage(1, imageEnable);
               } else if (iwebservice.getState().toString() == "DISABLED") {
                  itemWebServiceState.setImage(1, imageDisable);
               }
               itemWebServiceState.setText(1, ipipeline.getStatus().toString());
            }
         }
      }
   }

   public void ngGUI() {
      // delete all items
      tree.removeAll();
      tree.setHeaderVisible(false); // display header
      tree.setLinesVisible(false); // display lines
      // URI Maps
      TreeItem itemURIMap = new TreeItem(tree, SWT.NULL);
      itemURIMap.setImage(0, imageError);
      itemURIMap.setText(0, "No URIMap found.");
   }

   @Override
   public void init(IViewSite site) throws PartInitException {
      super.init(site);
      site.getPage().addPostSelectionListener(new ISelectionListener() {
         @Override
         public void selectionChanged(IWorkbenchPart arg0,
               ISelection selection) {
            if (isVisible() && !selection.isEmpty()) {
               Object firstElement = ((StructuredSelection) selection)
                     .getFirstElement();
               if (firstElement instanceof IURIMap) {
                  final IURIMap urimap2 = (IURIMap) firstElement;
                  populateInformation(urimap2.getName());
               }
            }
         } // the end of selectionChanged
      }); // the end of site.getPage().addSelectionListener
   } // the end of Override init method

   private boolean isVisible() { return getSite().getPage().isPartVisible(this); }

   public void setFocus() {} // the end of setFocus method
} // the end of this class
```

Note: The `ngGUI()` method is only used for extending the function for new textbox and “GO” button in the toolbar. See 5.4, “Extending a toolbar to search URIMap information based on user input” on page 125 for more detail

5.1.4 Running the sample

We now look at running our sample.

1. Run the plug-in as an Eclipse application.
2. Open the new view (Figure 5-3).

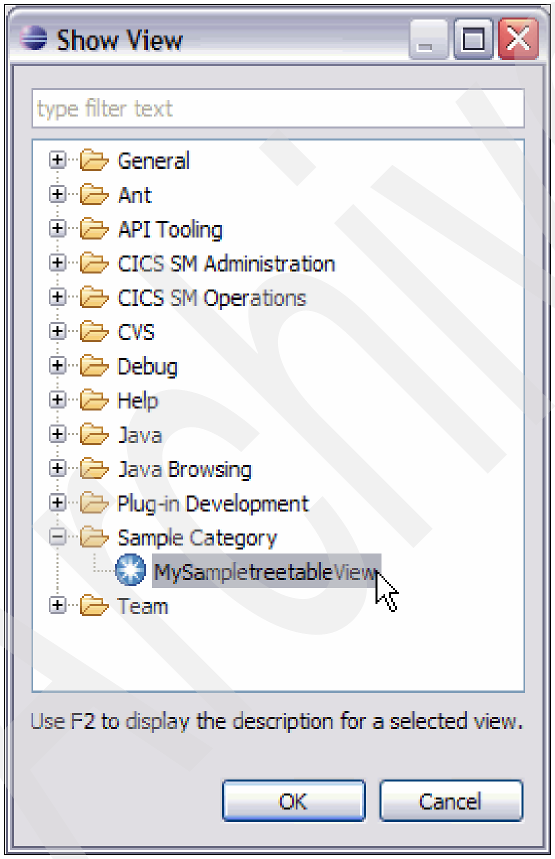


Figure 5-3 Select sampleView as Eclipse standard operation

3. Open the URIMap view (Figure 5-4).

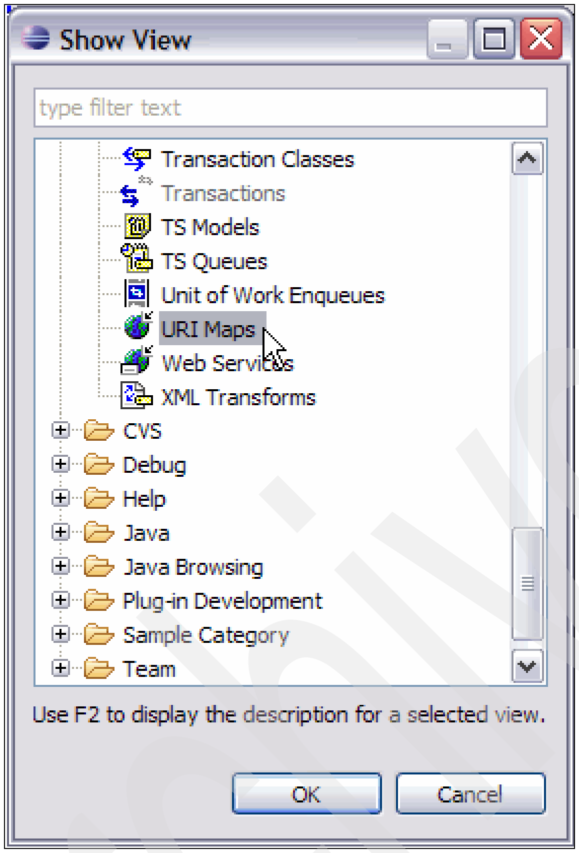
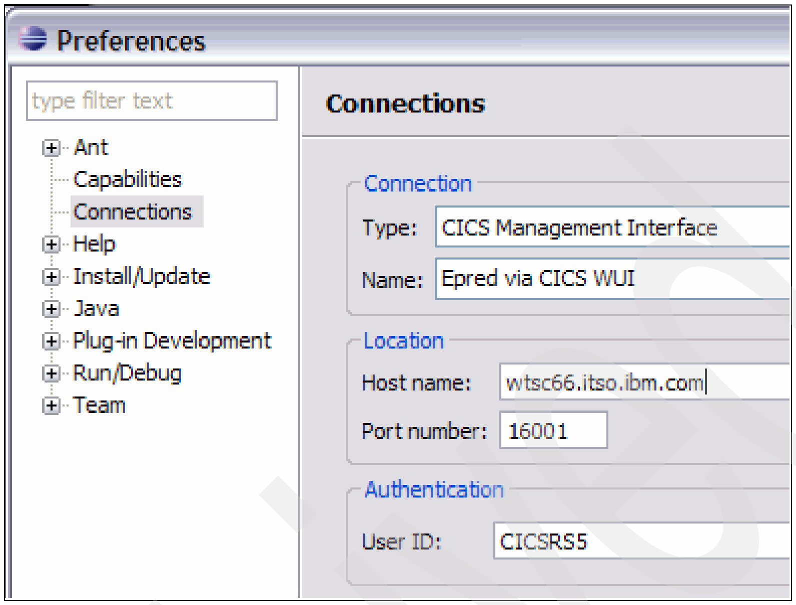


Figure 5-4 Open URIMap view

4. Connect your CICS system using preferences (Figure 5-5).


Preferences

type filter text

- Ant
- Capabilities
- Connections
- Help
- Install/Update
- Java
- Plug-in Development
- Run/Debug
- Team

Connections

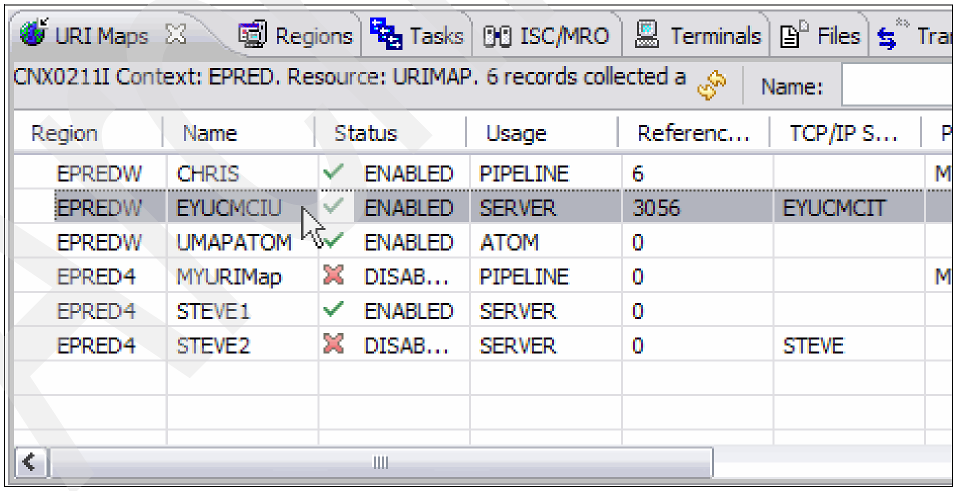
Connection
Type: CICS Management Interface
Name: Epred via CICS WUI

Location
Host name: wtsc66.itso.ibm.com
Port number: 16001

Authentication
User ID: CICSRS5

Figure 5-5 Connect to CICS System by Preference window

5. Click this in the URIMap (Figure 5-6).



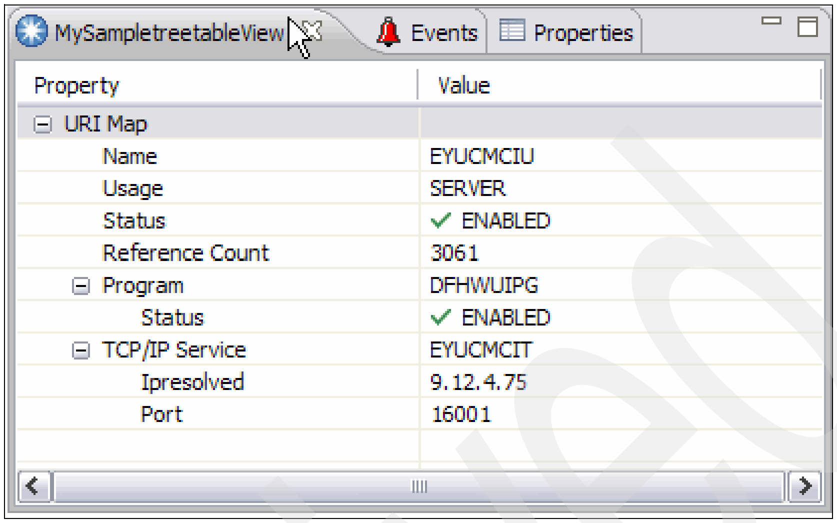
CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected a

Name:

Region	Name	Status	Usage	Referenc...	TCP/IP S...	P
EPREDW	CHRIS	✓ ENABLED	PIPELINE	6		M
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	3056	EYUCMCIT	
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0		
EPRED4	MYURIMap	✗ DISAB...	PIPELINE	0		M
EPRED4	STEVE1	✓ ENABLED	SERVER	0		
EPRED4	STEVE2	✗ DISAB...	SERVER	0	STEVE	

Figure 5-6 URIMap click in the URIMap view

6. You can get URIMap detailed information in the new view (Figure 5-7).



Property	Value
⊟ URI Map	
Name	EYUCMCIU
Usage	SERVER
Status	✓ ENABLED
Reference Count	3061
⊟ Program	DFHWUIPG
Status	✓ ENABLED
⊟ TCP/IP Service	EYUCMCIT
Ipresolved	9.12.4.75
Port	16001

Figure 5-7 URIMap detailed information display results

In 5.2, “Using a pop-up menu to access URIMap information” on page 110, we describe how to further extend and modify based on the new view shown in Figure 5-7.

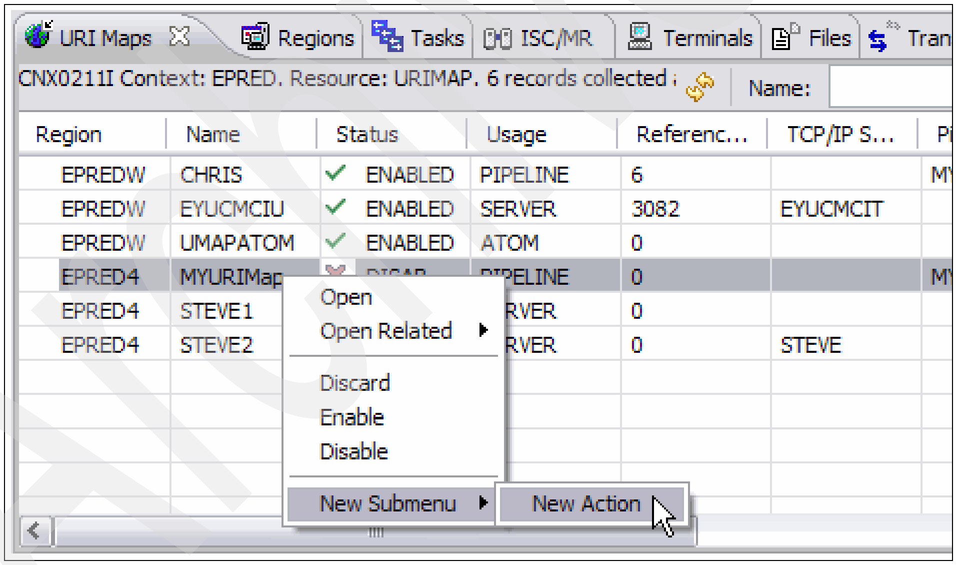
5.2 Using a pop-up menu to access URIMap information

To reach the target URIMap detailed information easier and faster, you can modify your plug-ins. The following sections discuss how to implement a pop-up menu function in the URIMap view.

5.2.1 Specification of a new pop-up menu

The URIMap view is provided by CICS Explorer. You can add a new function to show the selected URIMap detailed information more quickly using a new pop-up menu.

1. Right-click to display pop-up menu list.
2. Select **New Submenu**.
3. Select **New Action**.
4. If hidden, open **sampleView**.
5. If a URIMap is selected in the URI Maps view, its information is retrieved.
6. The URIMap information is then displayed in the sampleView.



CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected

Region	Name	Status	Usage	Referenc...	TCP/IP S...	Pi
EPREDW	CHRIS	✓ ENABLED	PIPELINE	6		MY
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	3082	EYUCMCIT	
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0		
EPRED4	MYURIMap	DISAB	PIPELINE	0		MY
EPRED4	STEVE1		RVER	0		
EPRED4	STEVE2		RVER	0	STEVE	

Figure 5-8 New action pop-up menu

5.2.2 Sample code

Add an extension point, erg.eclipse.ui.popupMenus, to allow for easier implementation using the eclipse template.

1. Click **Add** (Figure 5-9) in the Extensions tab to open the Extensions dialog box.

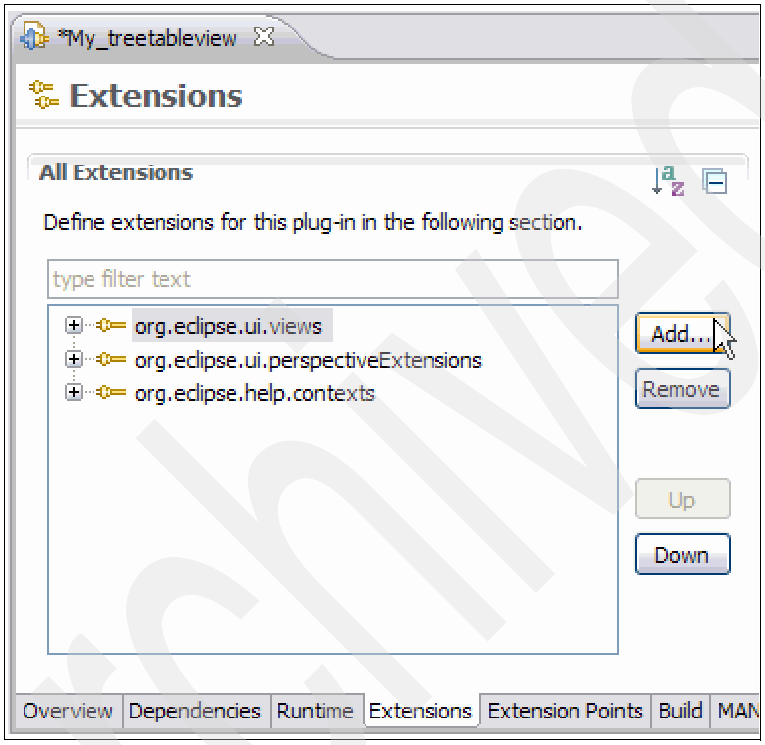
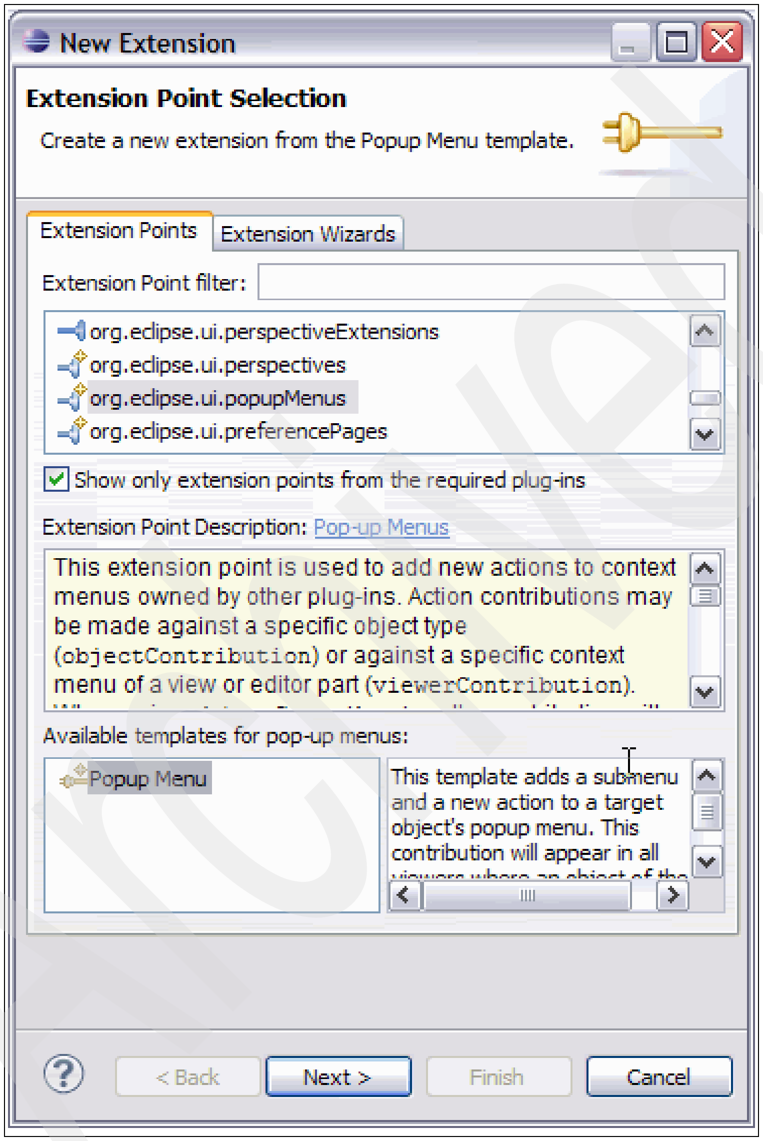


Figure 5-9 Start adding Extension from Extensions tab

2. Choose **org.eclipse.ui.popupMenus** as the extension point. In the “Available templates for pop-up menus” window, and click “Popup Menu” to generate the template source code (Figure 5-10). Click **Next**.


New Extension

Extension Point Selection

Create a new extension from the Popup Menu template.

Extension Points | Extension Wizards

Extension Point filter:

org.eclipse.ui.perspectiveExtensions
org.eclipse.ui.perspectives
org.eclipse.ui.popupMenus
org.eclipse.ui.preferencePages

[✓] Show only extension points from the required plug-ins

Extension Point Description: Pop-up Menus

This extension point is used to add new actions to context menus owned by other plug-ins. Action contributions may be made against a specific object type (objectContribution) or against a specific context menu of a view or editor part (viewerContribution).

Available templates for pop-up menus:

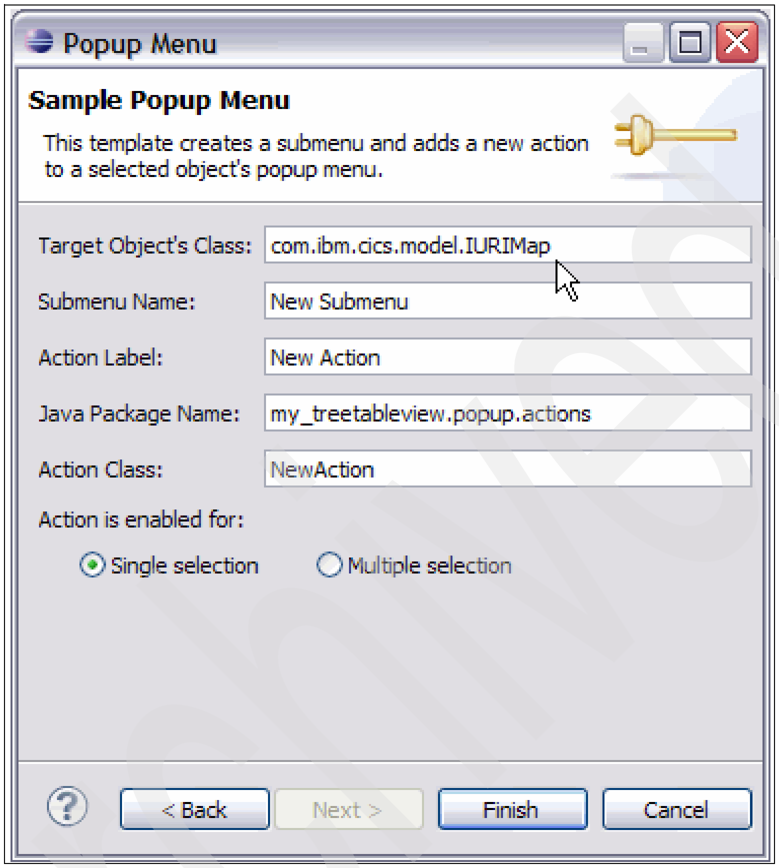
Popup Menu

This template adds a submenu and a new action to a target object's popup menu. This contribution will appear in all

< Back | Next > | Finish | Cancel

Figure 5-10 Extension Point Selection dialog window

3. Modify the Target Object's Class into "com.ibm.cics.model.IURIMap" to relate this new pop-up menu to the URIMap view (Figure 5-11). Click **Next**.



Sample Popup Menu

This template creates a submenu and adds a new action to a selected object's popup menu.

Target Object's Class: com.ibm.cics.model.IURIMap
Submenu Name: New Submenu
Action Label: New Action
Java Package Name: my_treetableview.popup.actions
Action Class: NewAction
Action is enabled for:
[✓ Single selection] Multiple selection

Figure 5-11   Sample pop-up menu dialog window

4. After the template source code and required XML is generated (Figure 5-12), modify the source code to meet your needs.

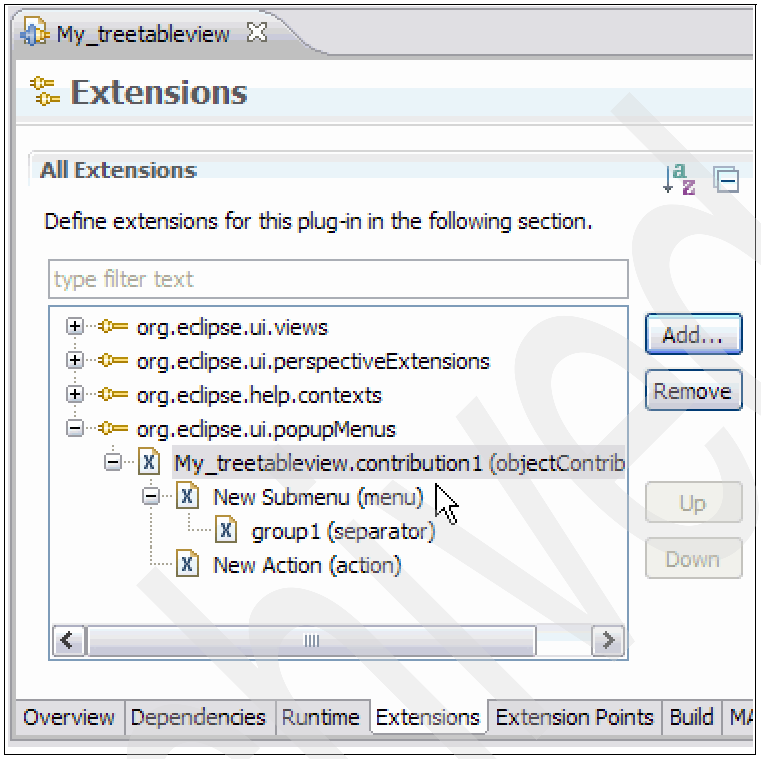


Figure 5-12 Sample pop-up menu dialog window

To get and display the detailed URIMap information, modify the run and `selectionChanged` methods in object PopupAction.java.

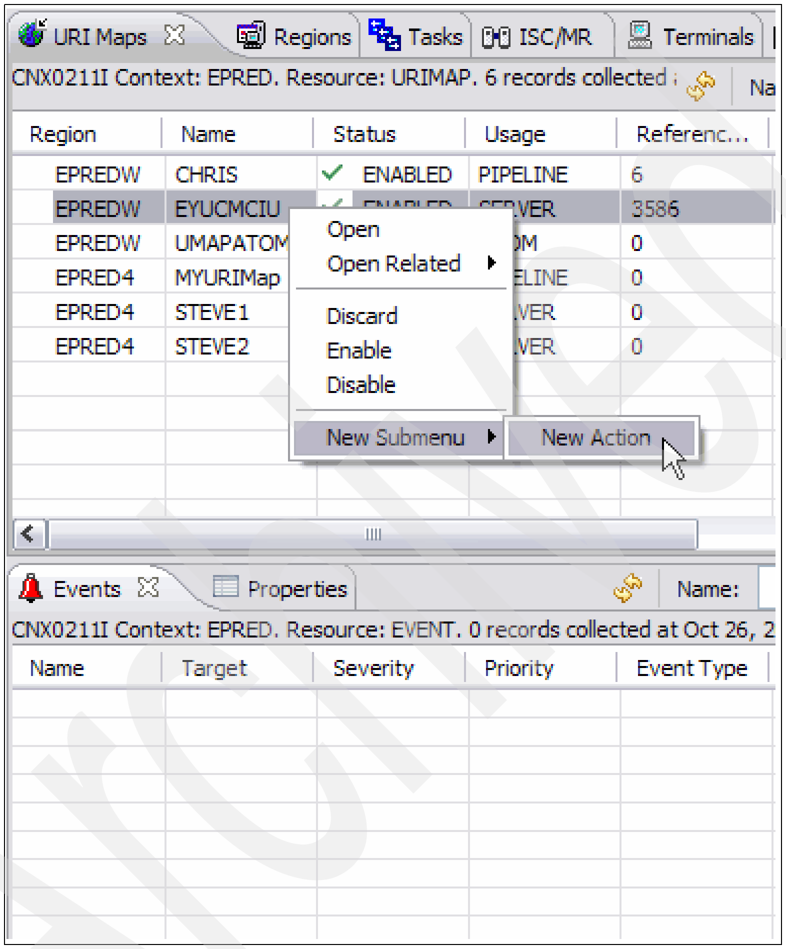
- `run` method
  a. If the view is closed, open the view by double-clicking the PopupAction.Java object.
  b. Invoke the `populateInformation` method in the sampleView with the target URIMap name.
- `selectionChanged` method
  – You can get URIMap resource information from URIMap view to invoke `populateInformation` in the sampleView. Example 5-2 on page 115 shows the PopupAction class after these changes have been applied to the generated template class.

Example 5-2 Sample code: PopupAction.java

```
package my_treetableview.popup.actions;

import my_treetableview.views.MySampletreetableView;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.StructuredSelection;
import org.eclipse.ui.IActionDelegate;
import org.eclipse.ui.IObjectActionDelegate;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;

import com.ibm.cics.model.IURIMap;

public class PopupAction implements IObjectActionDelegate {

   private IURIMap uriMap;

   /** Constructor for Action1. **/
   public PopupAction() { super(); }

   /** @see IObjectActionDelegate#setActivePart(IAction, IWorkbenchPart) **/
   public void setActivePart(IAction action, IWorkbenchPart targetPart) {}

   /** @see IActionDelegate#run(IAction) **/
   public void run(IAction action) {
      IWorkbench workbench = PlatformUI.getWorkbench();
      IWorkbenchWindow window = workbench.getActiveWorkbenchWindow();
      IWorkbenchPage page = window.getActivePage();
      try {
         IViewPart sampleView = page
               .findView("my_treetableview.views.MySampletreetableView");
         if (sampleView == null) {
            sampleView = page
                  .showView("my_treetableview.views.MySampletreetableView");
         }
         ((MySampletreetableView) sampleView).populateInformation(uriMap.getName());
      } catch (PartInitException e) {e.printStackTrace(); }
   }
   /** @see IActionDelegate#selectionChanged(IAction, ISelection) **/
   public void selectionChanged(IAction action, ISelection selection) {
      uriMap = (IURIMap) ((StructuredSelection) selection).getFirstElement();
   }
}
```

5.2.3 Operation

1. To get and show detailed information, right click the target URIMap in the URIMap view, and select pop-up menu (Figure 5-13).



CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected

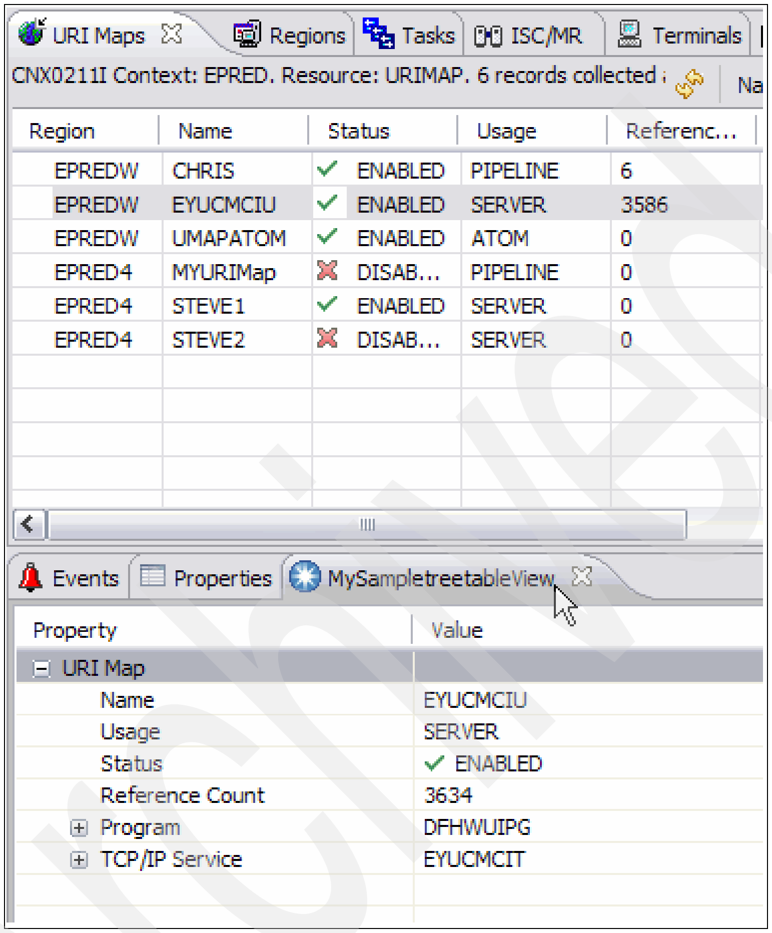
Region	Name	Status	Usage	Referenc...
EPREDW	CHRIS	✓ ENABLED	PIPELINE	6
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	3586
EPREDW	UMAPATOM		OM	0
EPRED4	MYURIMap		ELINE	0
EPRED4	STEVE1		VER	0
EPRED4	STEVE2		VER	0

CNX0211I Context: EPRED. Resource: EVENT. 0 records collected at Oct 26, 2

Name	Target	Severity	Priority	Event Type

Figure 5-13 Pop-up menu on the URIMap

2. If the sampleView is hidden, it appears automatically, and displays the results, as shown in Figure 5-14.



CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected

Region	Name	Status	Usage	Referenc...
EPREDW	CHRIS	✓ ENABLED	PIPELINE	6
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	3586
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0
EPRED4	MYURIMap	✗ DISAB...	PIPELINE	0
EPRED4	STEVE1	✓ ENABLED	SERVER	0
EPRED4	STEVE2	✗ DISAB...	SERVER	0

Property	Value
⊟ URI Map	
Name	EYUCMCIU
Usage	SERVER
Status	✓ ENABLED
Reference Count	3634
⊞ Program	DFHWUIPG
⊞ TCP/IP Service	EYUCMCIT

Figure 5-14 Results of pop-up menu

5.3 Extending actions of the toolbar and menu bar to access URIMap information

To access the detailed URIMap information more quickly, you can implement the view as an actions with an icon button in a toolbar or menu bar.

5.3.1 Specification of new actions

In our generated example, you can view the detailed information of a selected URIMap by performing the following steps in the workbench perspective:

1. Click **Sample Menu** in the menu bar, or press **New** in the toolbar, as shown in Figure 5-15.
2. If hidden, the sampleView appears.
3. The detailed information is fetched for the selected URIMap..
4. The sampleView displays the URIMap information as a hierarchical structure.

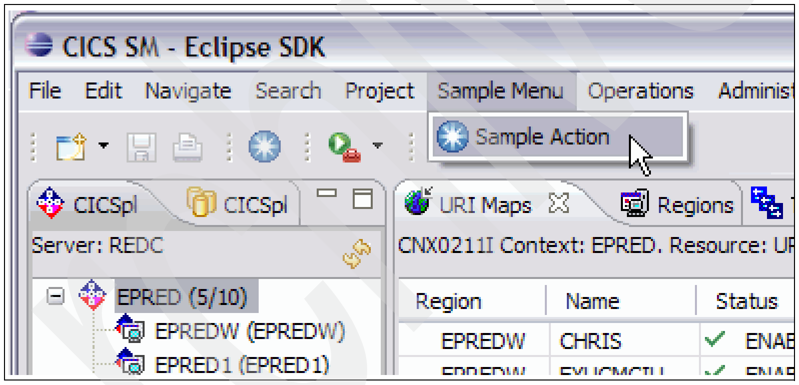


Figure 5-15 New actions by new menu bar, and toolbar(icon button)

Select the target URIMap line in the URIMap view in advance to show the new action. If you do not open the URIMap view or select any URIMap in advance, the view is shown with no information.

5.3.2 Extending actions of the toolbar and menu bar to access URIMap information

Perform the following steps to extend the toolbar and menu bar to access URIMap information:

1. You can add the views Extension point from the Extensions tab of the project as shown in Figure 5-16. Select the views extension and click **Add**.

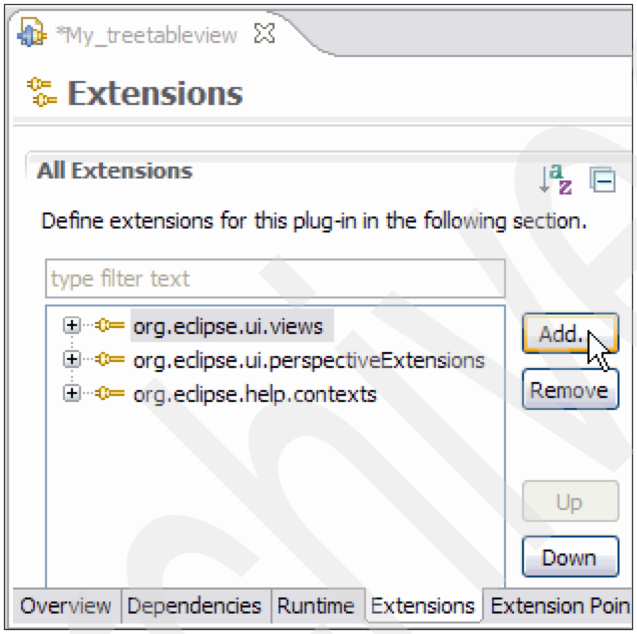
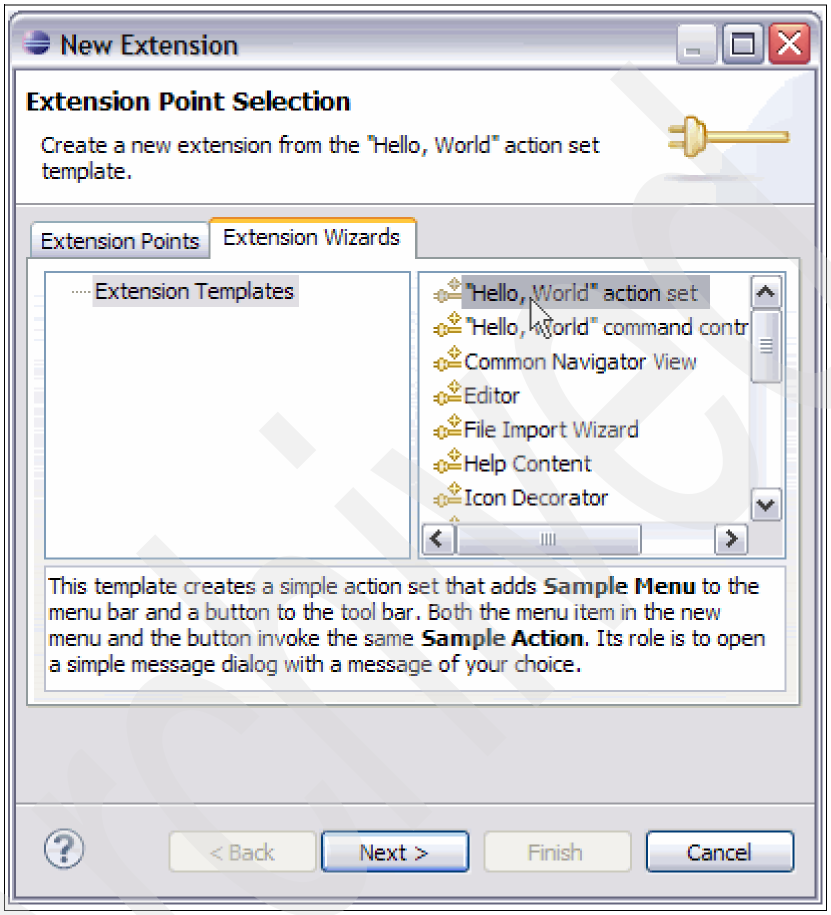


Figure 5-16 Selecting Add in the Extensions tab

2. You can choose the “Hello, World” action set template from the Extension Wizards tab. This template is suitable for the purpose of extending the icon button in the toolbar and menu bar. See Figure 5-17.

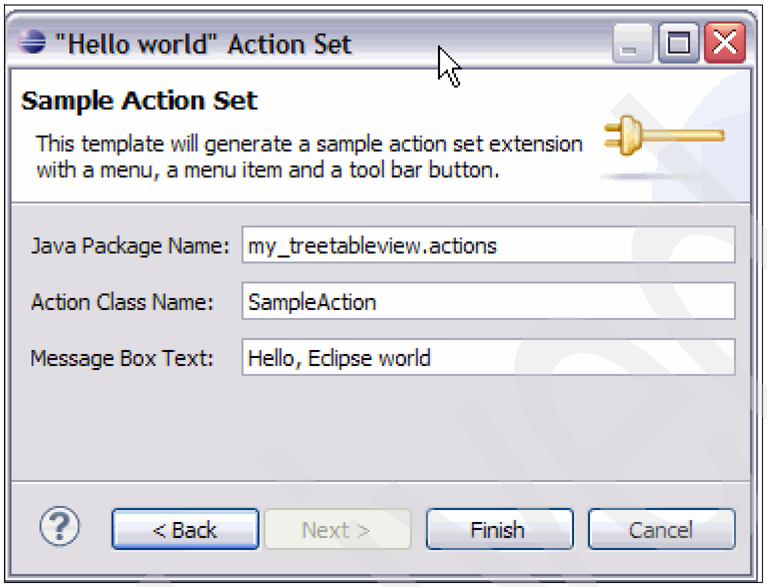


Create a new extension from the "Hello, World" action set template.

This template creates a simple action set that adds **Sample Menu** to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same **Sample Action**. Its role is to open a simple message dialog with a message of your choice.

Figure 5-17 Choose “Hello, World” action set template

3. Enter Java Package name and Action class, as shown in Figure 5-18. The Message Box text is not required so leave it with the default values.



Sample Action Set

This template will generate a sample action set extension with a menu, a menu item and a tool bar button.

Java Package Name: my_treetableview.actions

Action Class Name: SampleAction

Message Box Text: Hello, Eclipse world

Figure 5-18 Sample Action Set dialog window

4. Click **Finish** to generate the sample source code.

This sample uses an extension point called org.eclipse.ui.actionSets. To get and display URIMap detailed information, you modify the `run` method in the SampleAction.

1. Investigate the status of sampleView. If the view is closed, open the view first.
2. Get the name of the URIMap for which you want more information from the URIMap view.
3. Invoke the `populateInformation` method in the sampleView with the target URIMap name.

Example 5-3 shows the sample code.

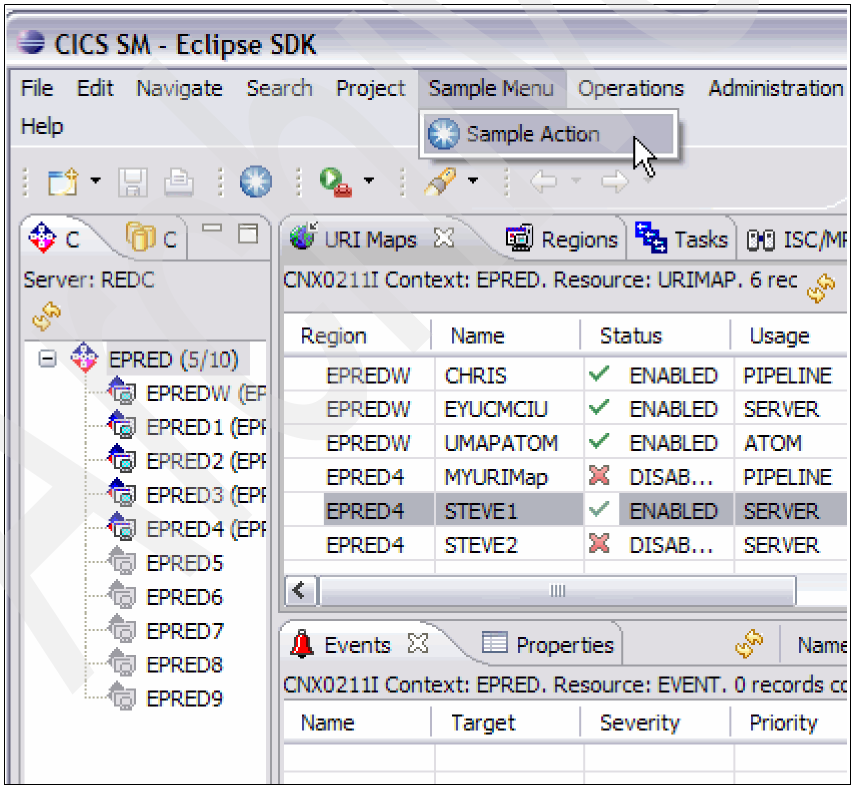
Example 5-3 Sample code: SampleAction.java

```
package my_treetableview.actions;

import my_treetableview.views.MySampletreetableView;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;

import com.ibm.cics.model.IURIMap;

/** @see IWorkbenchWindowActionDelegate **/
public class SampleAction implements IWorkbenchWindowActionDelegate {

   /** The constructor. **/
   public SampleAction() {}

   /** @see IWorkbenchWindowActionDelegate#run **/
   public void run(IAction action) {
      IWorkbench workbench = PlatformUI.getWorkbench();
      IWorkbenchWindow window = workbench.getActiveWorkbenchWindow();
      IWorkbenchPage page = window.getActivePage();
      try {
         IViewPart sampleView = page
               .findView("my_treetableview.views.MySampletreetableView");
         if (sampleView == null) {
            sampleView = page
                  .showView("my_treetableview.views.MySampletreetableView");
         }
         IViewPart uriMapView = page.findView("com.ibm.cics.sm.ui.views.uriMaps");
         if (uriMapView != null) {
            IStructuredSelection structuredSelection = (IStructuredSelection) uriMapView
                  .getSite().getSelectionProvider().getSelection();
            if (!structuredSelection.isEmpty()) {
               IURIMap uriMap = (IURIMap) structuredSelection.getFirstElement();
               ((MySampletreetableView) sampleView)
                     .populateInformation(uriMap.getName());
            }
         }
      } catch (PartInitException e) { e.printStackTrace(); }
   }

   /** @see IWorkbenchWindowActionDelegate#dispose **/
   public void dispose() {}

   /** @see IWorkbenchWindowActionDelegate#init **/
   public void init(IWorkbenchWindow window) {}

   @Override
   public void selectionChanged(IAction action, ISelection selection) {}
}
```

5.3.3 Operation

Now we can test our sample action and look at the deatils for a URIMap.

1. Select the target URIMap in the URIMap view, then click **Sample Action** in the toolbar, or select a new action in the menu bar. See Figure 5-19.



CNX0211I Context: EPRED. Resource: URIMAP. 6 rec

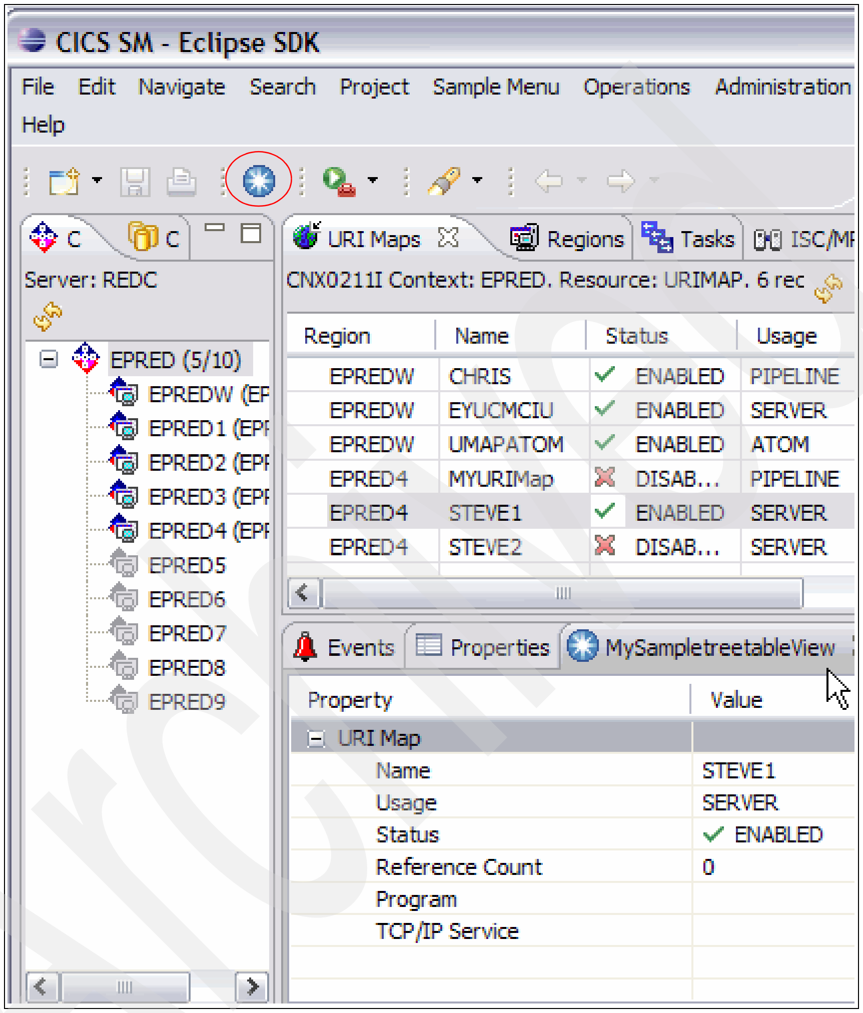
Region	Name	Status	Usage
EPREDW	CHRIS	✓ ENABLED	PIPELINE
EPREDW	EYUCMCIU	✓ ENABLED	SERVER
EPREDW	UMAPATOM	✓ ENABLED	ATOM
EPRED4	MYURIMap	✗ DISAB...	PIPELINE
EPRED4	STEVE1	✓ ENABLED	SERVER
EPRED4	STEVE2	✗ DISAB...	SERVER

CNX0211I Context: EPRED. Resource: EVENT. 0 records co

Name	Target	Severity	Priority

Figure 5-19 Select target URIMap, and select Sample Action menu bar icon

2. If the sampleView is hidden, it appears automatically, and displays the results. If you click the new toolbar button instead of menu bar icon, you get the same results. See Figure 5-20.



CNX0211I Context: EPRED. Resource: URIMAP. 6 rec

Region	Name	Status	Usage
EPREDW	CHRIS	✓ ENABLED	PIPELINE
EPREDW	EYUCMCIU	✓ ENABLED	SERVER
EPREDW	UMAPATOM	✓ ENABLED	ATOM
EPRED4	MYURIMap	✗ DISAB...	PIPELINE
EPRED4	STEVE1	✓ ENABLED	SERVER
EPRED4	STEVE2	✗ DISAB...	SERVER

Property	Value
URI Map	
Name	STEVE1
Usage	SERVER
Status	✓ ENABLED
Reference Count	0
Program	
TCP/IP Service	

Figure 5-20 Results of new action

5.4 Extending a toolbar to search URIMap information based on user input

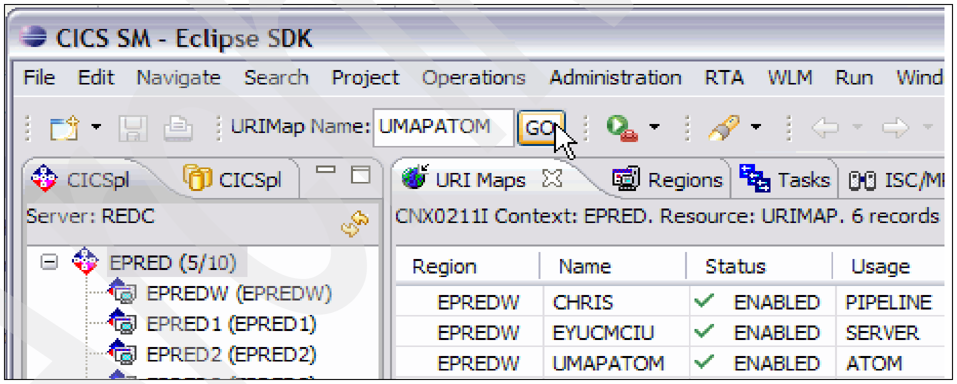
To access URIMap detailed information easier and faster, you can implement a textbox and button in the toolbar, like a URIMap filter search box.

5.4.1 Specification of new textbox and button

You can use a new function to show the target URIMap detailed information more quickly using a textbox and **GO** button in the toolbar workbench perspective:

1. Enter the target URIMap name in the new textbox in the toolbar.
2. Press the enter key, or click the new **GO** button.
3. If hidden, sampleView appears.
4. Get the specified URIMap detailed information.
5. Show this in the sampleView as hierarchy structure.

This function can execute if the URIMap view is hidden, or if no URIMap item has been selected in URIMap view. See Figure 5-21.



CNX0211I Context: EPRED. Resource: URIMAP. 6 records

Region	Name	Status	Usage
EPREDW	CHRIS	✓ ENABLED	PIPELINE
EPREDW	EYUCMCIU	✓ ENABLED	SERVER
EPREDW	UMAPATOM	✓ ENABLED	ATOM

Figure 5-21 New textbox and GO button to show target URIMap information

If the URIMap name does not exist in the URIMap resources, a view is displayed with the message `No URIMap found`. This message handling is implemented by `noGUI` method in the sampleView. For this detailed sample code, see 5.1.3, "Using our sample code" on page 98.

5.4.2 Sample code

To add a text box in the workbench toolbar, perform the following steps:

1. Select the views extension point, and click **Add** in the Extensions tab in the project. See Figure 5-22.

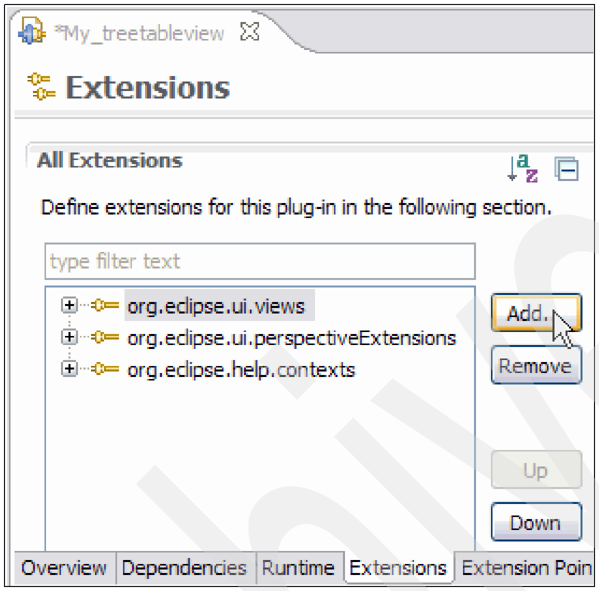


*Figure 5-22 Clicking **Add** in the Extensions tab*

2. Type `org.eclipse.ui.menus` in the Extension Point filter, then click **Finish**.

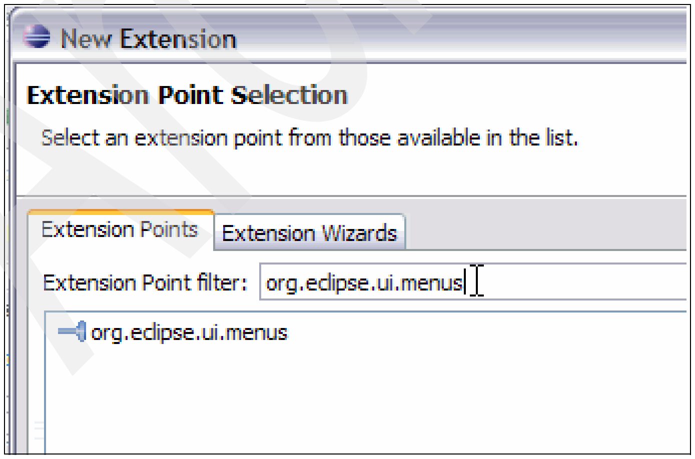


Figure 5-23 Extension Point Selection dialog window

3. Right-click **org.eclipse.ui.menus**, then click **New** → **menuContribution**.

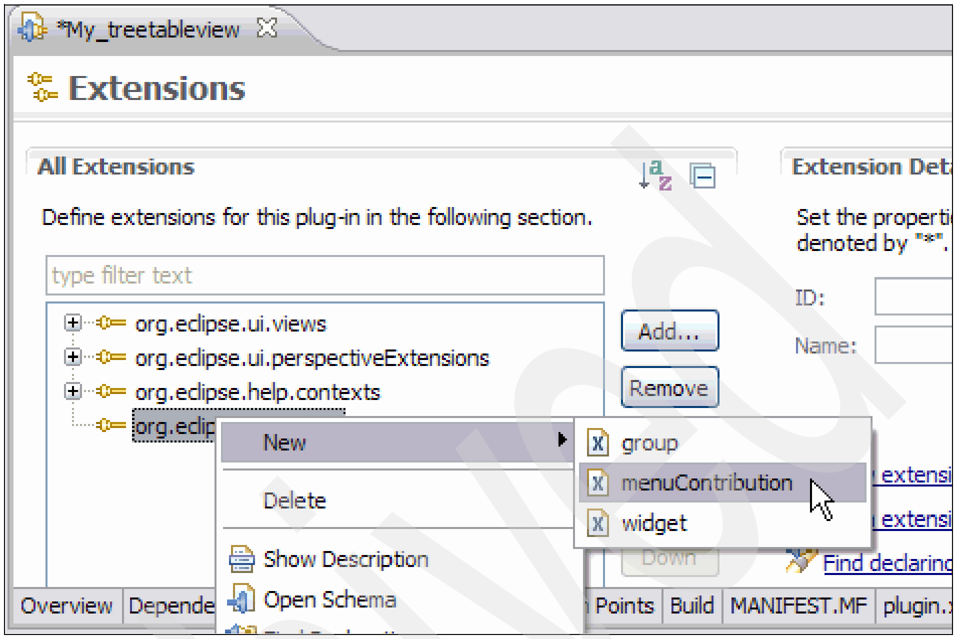


Figure 5-24   Add menuContribution to extension point

4. Enter `toolbar:org.eclipse.ui.main.toolbar` in the locationURI textbox. See Figure 5-25.

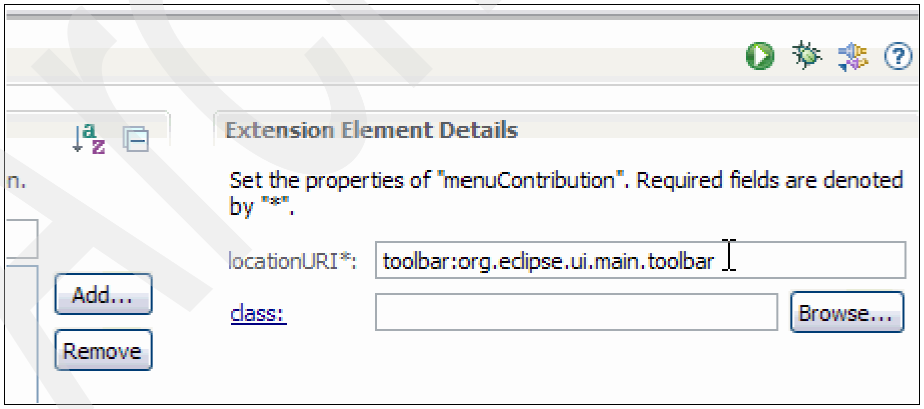

Extension Element Details
Set the properties of "menuContribution". Required fields are denoted by "*".
locationURI*:
toolbar:org.eclipse.ui.main.toolbar
class:
Browse...
Add...
Remove

Figure 5-25   Overwrite locationURI

5. Right-click toolbar:org.eclipse.ui.main.toolbar and select **New** → **Toolbar**. See Figure 5-26.

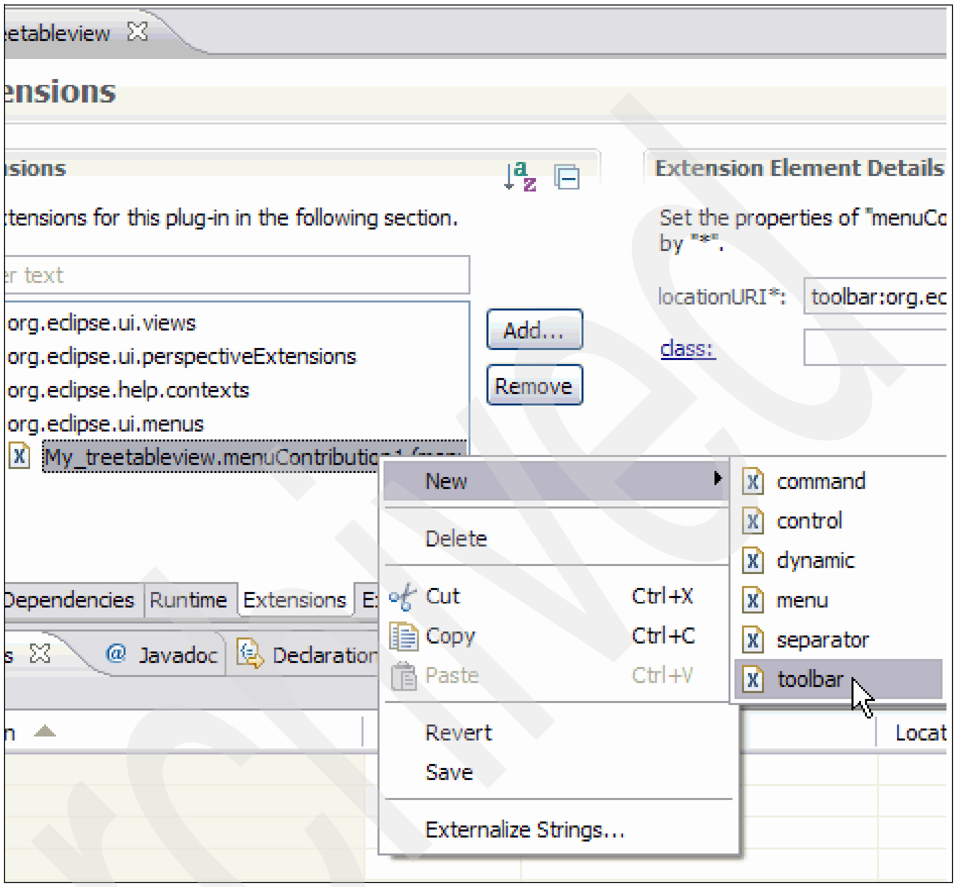


Figure 5-26 Add toolbar to new menuContribution

6. Right-click the new generated toolbar, then click **New** → **Control**. See Figure 5-27.

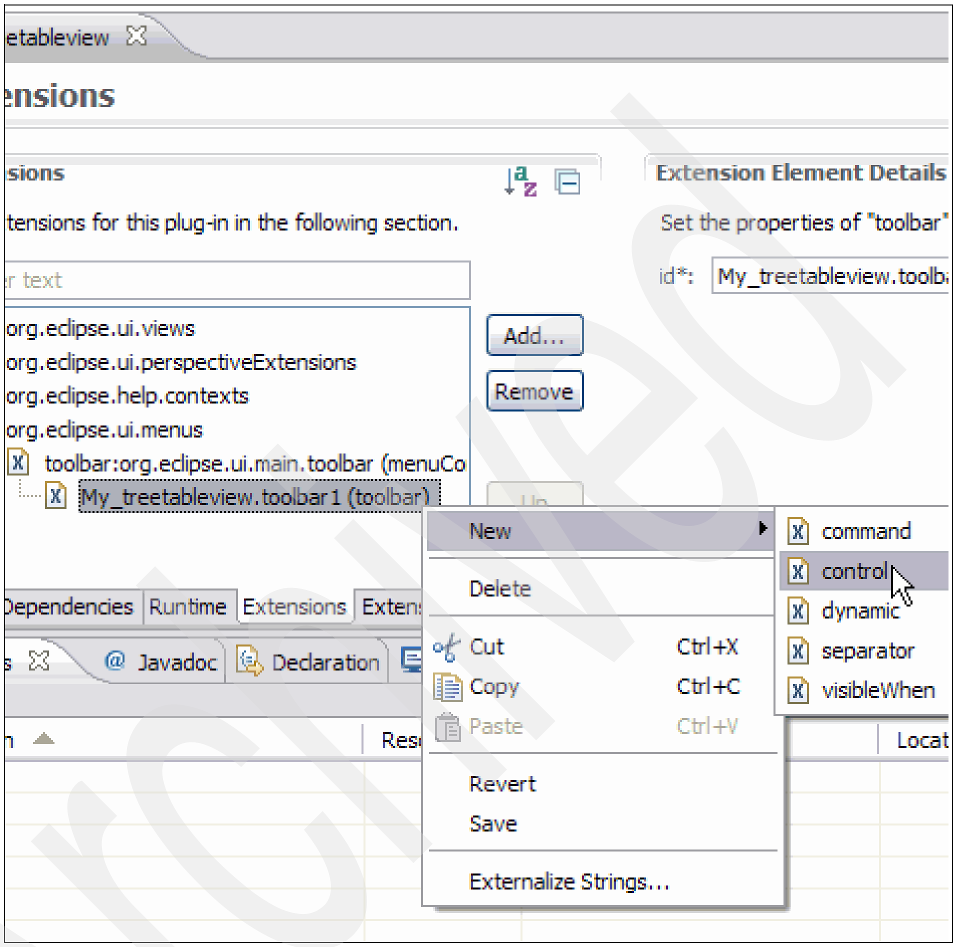
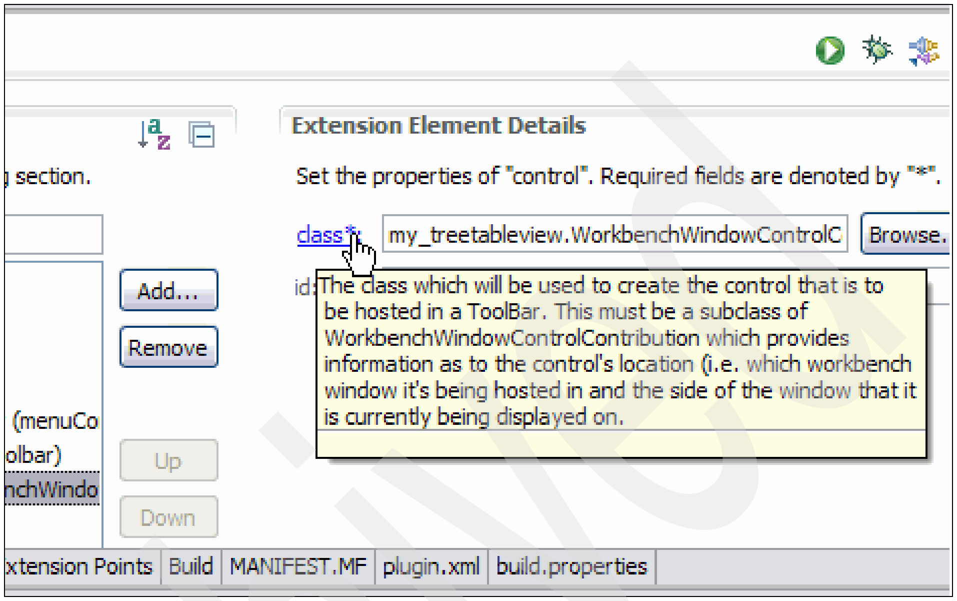

etableview
ensions
sions
tensions for this plug-in in the following section.
r text
org.eclipse.ui.views
org.eclipse.ui.perspectiveExtensions
org.eclipse.help.contexts
org.eclipse.ui.menus
toolbar:org.eclipse.ui.main.toolbar (menuCo
My_treetableview.toolbar1 (toolbar)
Add...
Remove
Extension Element Details
Set the properties of "toolbar"
id*: My_treetableview.toolb
New
Delete
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Revert
Save
Externalize Strings...
command
control
dynamic
separator
visibleWhen
Dependencies
Runtime
Extensions
Exten
Javadoc
Declaration

Figure 5-27 Add new control for textbox

7. This new control references the new class for our textbox. Click **class** to bring up a dialog box that prompts you to specify the name of this class. See Figure 5-28.


Extension Element Details

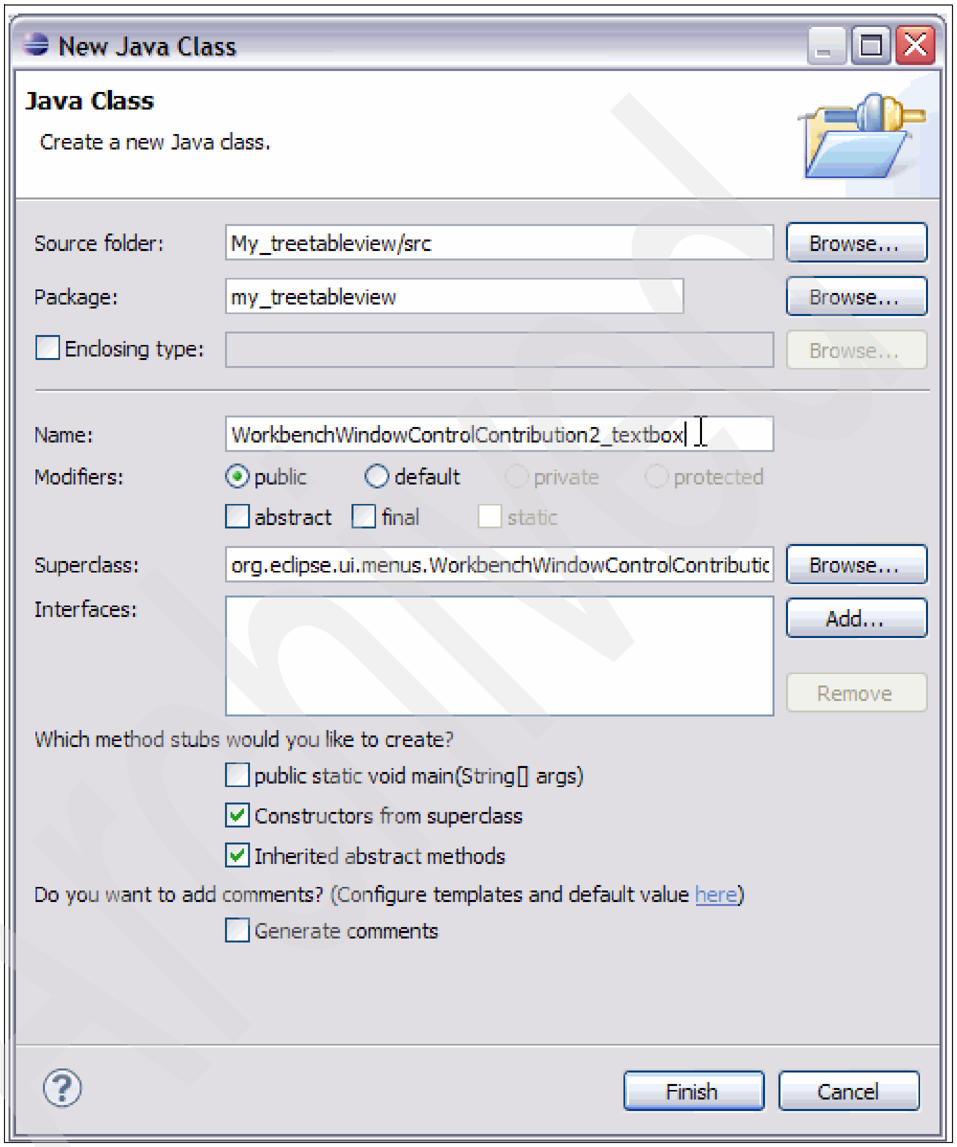
Set the properties of "control". Required fields are denoted by "*".

class*: my_treetableview.WorkbenchWindowControlC

The class which will be used to create the control that is to be hosted in a ToolBar. This must be a subclass of WorkbenchWindowControlContribution which provides information as to the control's location (i.e. which workbench window it's being hosted in and the side of the window that it is currently being displayed on.

Figure 5-28 Click class link to overwrite new class file

8. Specify your new class name in the Name field, then click **Finish**. See Figure 5-29.



New Java Class

Java Class

Create a new Java class.

Source folder: My_treetableview/src

Package: my_treetableview

[ ] Enclosing type:

Name: WorkbenchWindowControlContribution2_textbox

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: org.eclipse.ui.menus.WorkbenchWindowControlContributic

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[✓] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

Figure 5-29 Specify new class information

The sample code is now generated, you are able to modify the source code for your new textbox.

After the Enter key has been pressed, the selectionListener is generated. To get and display the URIMap detailed information based on the specified URIMap name in the textbox, you can now modify selectionListener.

- `createControl` method
  - Create a textbox instance, and specify the detailed attribute.
- `getText` method
  - Get a URIMap name, which is specified in the textbox as String attribute. This method is also invoked by the **GO** button. Integrate this function here.
- `selectionListener` method
  a. Invoke `getText` method to get URIMap name.
  b. Investigate the status of sampleView. If the view is closed, you can open the view at first.
  c. Get URIMap resource information from URIMap view to invoke populateInformation in the sampleView.

The sample code for adding a textbox is shown in Example 5-4.

Example 5-4 Sample code: WorkbenchWindowControlContribution2_textbox.java

```
package my_treetableview;

import my_treetableview.views.MySampletreetableView;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.SelectionListener;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.menus.WorkbenchWindowControlContribution;

public class WorkbenchWindowControlContribution2_textbox extends
      WorkbenchWindowControlContribution {

   static private Text text;
   public WorkbenchWindowControlContribution2_textbox() {}

   public WorkbenchWindowControlContribution2_textbox(String id) {super(id); }

   @Override
   protected Control createControl(Composite parent) {
      text = new Text(parent, SWT.SINGLE | SWT.BORDER);
      // specify maximum number of character
      text.setTextLimit(8);
      // add listener for the case that enter-key is pushed.
      text.addSelectionListener(new myListener());
      return text;
   }

   public String getTextbox() {
      // get the string from the textbox
      String textMessage = text.getText();
      textMessage = textMessage.toUpperCase();
      return textMessage;
   }

   public class myListener implements SelectionListener {
      @Override
      public void widgetDefaultSelected(SelectionEvent e) {
         // get strings from the textbox by getTextbox() method
         String textMessage = getTextbox();
         // get workbench information and judge whether sampleView is open or not.
         IWorkbench workbench = PlatformUI.getWorkbench();
         IWorkbenchWindow window = workbench.getActiveWorkbenchWindow();
         IWorkbenchPage page = window.getActivePage();

         try {
            IViewPart sampleView = page
                  .findView("my_treetableview.views.MySampletreetableView");
            // if sampleView is not opened, invoke showView method.
            if (sampleView == null) {
               sampleView = page
                     .showView("my_treetableview.views.MySampletreetableView");
            }
            ((MySampletreetableView) sampleView).populateInformation(textMessage);
         } catch (PartInitException e1) { e1.printStackTrace(); }
      }
      @Override
      public void widgetSelected(SelectionEvent e2) {}
   }
}
```

A GO button control (again with a text entry box) can also be added, this would be done in the same way as shown in Figure 5-27 on page 129. The generated source code has to be modified to add the **GO** button.

Clicking **GO** generates a SelectionEvent, so we need a SelectionListener that gets and displays the URIMap detailed information based on the URIMap name specified in the textbox. The changes are as follows:

- `createControl` method
  - You can create button instance, and specify detailed attribute.
- `selectionListener` method
  a. Invoke `getText` method in the textbox to get the URIMap name.
  b. Investigate the status of the sampleView. If the view is closed, open the view first.
  c. Get the URIMap resource information from URIMap view to invoke populateInformation in the sampleView.

The sample code for textbox is shown in Example 5-5.

Example 5-5 Sample code: WorkbenchWindowControlContribution3_button.java

```
package my_treetableview;

import my_treetableview.views.MySampletreetableView;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.SelectionListener;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.menus.WorkbenchWindowControlContribution;

public class WorkbenchWindowControlContribution3_bottun extends
      WorkbenchWindowControlContribution {

   public WorkbenchWindowControlContribution3_bottun() {}
   public WorkbenchWindowControlContribution3_bottun(String id) { super(id); }
   @Override
   protected Control createControl(Composite parent) {
      Button button = new Button(parent, SWT.PUSH);
      button.setText("GO");
      // add listener for the case that the bottun is pushed.
      button.addSelectionListener(new myListener());
      return button;
   }

   class myListener implements SelectionListener {
      @Override
      public void widgetDefaultSelected(SelectionEvent e) {}

      @Override
      public void widgetSelected(SelectionEvent e) {
         WorkbenchWindowControlContribution2_textbox sampleclass = new
WorkbenchWindowControlContribution2_textbox();
         String textMessage = sampleclass.getTextbox();
         // get workbench information and judge whether sampleView is open or not.
         IWorkbench workbench = PlatformUI.getWorkbench();
         IWorkbenchWindow window = workbench.getActiveWorkbenchWindow();
         IWorkbenchPage page = window.getActivePage();
         try {
            IViewPart sampleView = page
                  .findView("my_treetableview.views.MySampletreetableView");
            // if sampleView is not opened, invoke showView method.
            if (sampleView == null) {
               sampleView = page
                     .showView("my_treetableview.views.MySampletreetableView");
            }
            ((MySampletreetableView) sampleView).populateInformation(textMessage);
         } catch (PartInitException e2) { e2.printStackTrace(); }
      }
   }
}
```

In Example 5-6 the label is also added for a better user interface. In the `createControl` method, you specify detailed attributes for the label.

Example 5-6 Sample code: WorkbenchWindowControlContribution1_label.java

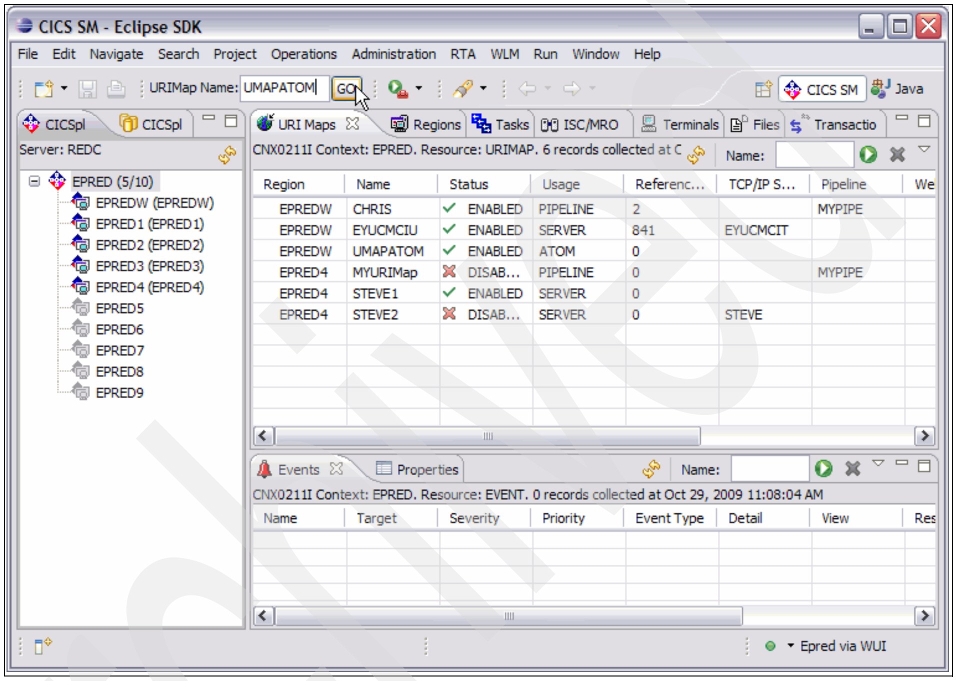
```
package my_treetableview;

import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.ui.menus.WorkbenchWindowControlContribution;

public class WorkbenchWindowControlContribution1_label extends
      WorkbenchWindowControlContribution {

   public WorkbenchWindowControlContribution1_label() {}

   public WorkbenchWindowControlContribution1_label(String id) { super(id); }

   @Override
   protected Control createControl(Composite parent) {
      Composite composite = new Composite(parent, SWT.NONE);
      Label label = new Label(composite, SWT.NONE);
      label.setText("URIMap Name:");
      label.pack();
      label.setLocation(0, 3);
      return composite;
   }
}
```

Note: Simple `setText` method into label makes the text upper location. To avoid this, Example 5-6 uses Composite class, and `pack` method.

5.4.3 Operation

Now we can test our input textbox with **GO** button and look at the details for a URIMap.

1. Input the URIMap name in the text box, and hit the return key or click **GO**. See Figure 5-30.



Server: REDC

- EPRED (5/10)
  - EPREDW (EPREDW)
  - EPRED1 (EPRED1)
  - EPRED2 (EPRED2)
  - EPRED3 (EPRED3)
  - EPRED4 (EPRED4)
  - EPRED5
  - EPRED6
  - EPRED7
  - EPRED8
  - EPRED9

CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected at C

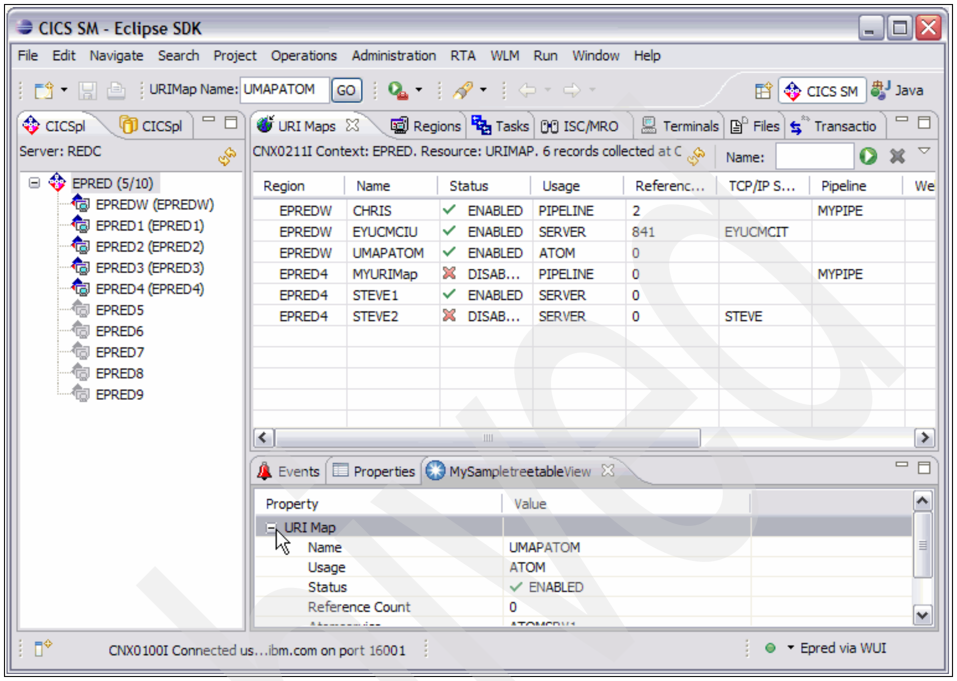
Region	Name	Status	Usage	Referenc...	TCP/IP S...	Pipeline	We
EPREDW	CHRIS	✓ ENABLED	PIPELINE	2		MYPIPE	
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	841	EYUCMCIT		
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0			
EPRED4	MYURIMap	✗ DISAB...	PIPELINE	0		MYPIPE	
EPRED4	STEVE1	✓ ENABLED	SERVER	0			
EPRED4	STEVE2	✗ DISAB...	SERVER	0	STEVE		

CNX0211I Context: EPRED. Resource: EVENT. 0 records collected at Oct 29, 2009 11:08:04 AM

Name	Target	Severity	Priority	Event Type	Detail	View	Res

*Figure 5-30 Input target URIMap name, and click **GO**.*

2. If sampleView is hidden, it appears automatically, and displays results. See Figure 5-31.


CICS SM - Eclipse SDK
File Edit Navigate Search Project Operations Administration RTA WLM Run Window Help
URIMap Name: UMAPATOM GO
CICS SM Java
CICSpl CICSpl
URI Maps Regions Tasks ISC/MRO Terminals Files Transactio
Events Properties MySampletreetableView
Name:
Epred via WUI

Server: REDC

- EPRED (5/10)
  - EPREDW (EPREDW)
  - EPRED1 (EPRED1)
  - EPRED2 (EPRED2)
  - EPRED3 (EPRED3)
  - EPRED4 (EPRED4)
  - EPRED5
  - EPRED6
  - EPRED7
  - EPRED8
  - EPRED9

CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected at C

Region	Name	Status	Usage	Referenc...	TCP/IP S...	Pipeline	We
EPREDW	CHRIS	✓ ENABLED	PIPELINE	2		MYPIPE	
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	841	EYUCMCIT		
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0			
EPRED4	MYURIMap	✗ DISAB...	PIPELINE	0		MYPIPE	
EPRED4	STEVE1	✓ ENABLED	SERVER	0			
EPRED4	STEVE2	✗ DISAB...	SERVER	0	STEVE		

Property	Value
URI Map	
Name	UMAPATOM
Usage	ATOM
Status	✓ ENABLED
Reference Count	0

CNX0100I Connected us...ibm.com on port 16001

Figure 5-31   Results of new textbox and GO button

5.5 Background process implementation

In the above examples where we extend the URIMap view, accessing processes to get CICS resource information is done as a synchronous process in Eclipse using the main thread. This basically stops Eclipse. You cannot do anything with Eclipse in the period, from getting and receiving CICS resource information, to completion of drawing them in the sampleView.

To solve this problem, you can access CICS resource information about a separate thread as a background process. This method is described in the following sections.

5.5.1 Specifications of the background process

Implementing background processing in your plug-in achieves the following:

- Access to actual CICS information using another thread in CICS Explorer.
- Requires drawing a view process as an asynchronous type process, from another thread to main thread. Eclipse can do GUI process only in main thread, so you can NOT draw view from new another thread.
- Implement the progress bar into a workbench window, to allow background process status percentage visually. We divide the progress into 4 steps, and completing each step means 25% progress.
  a. Connect to CICS.
  b. Get URIMap information.
  c. Get TCP/IP service resource information.
  d. Get other information depending on the URIMap USAGE value.
- Invoke "display.getDefault().asyncExec" in another thread to invoke drawing view process to main thread.

These steps are shown in Figure 5-32.

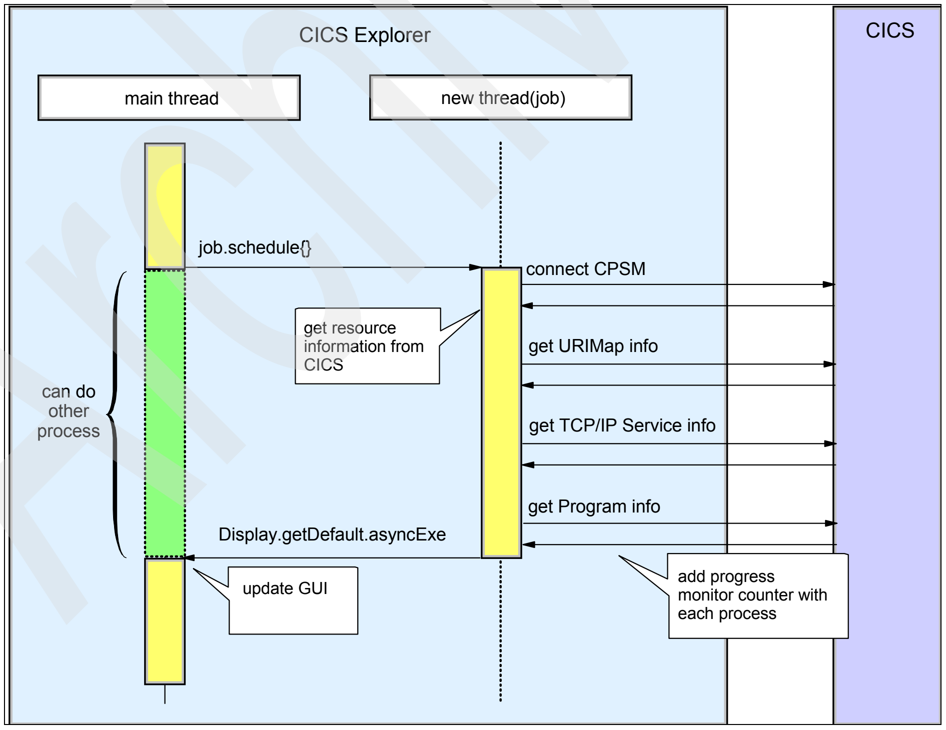


Figure 5-32 Multithread processing

5.5.2 Sample code

Example 5-7 is the sample code, the key points are how to do the process as a job and implement the progress bar.

To implement this, modify only two methods in the MySampletreetableView.java. See Example 5-7 and Example 5-8 for details.

Example 5-7 Background processing sample code_1: populateInformation method

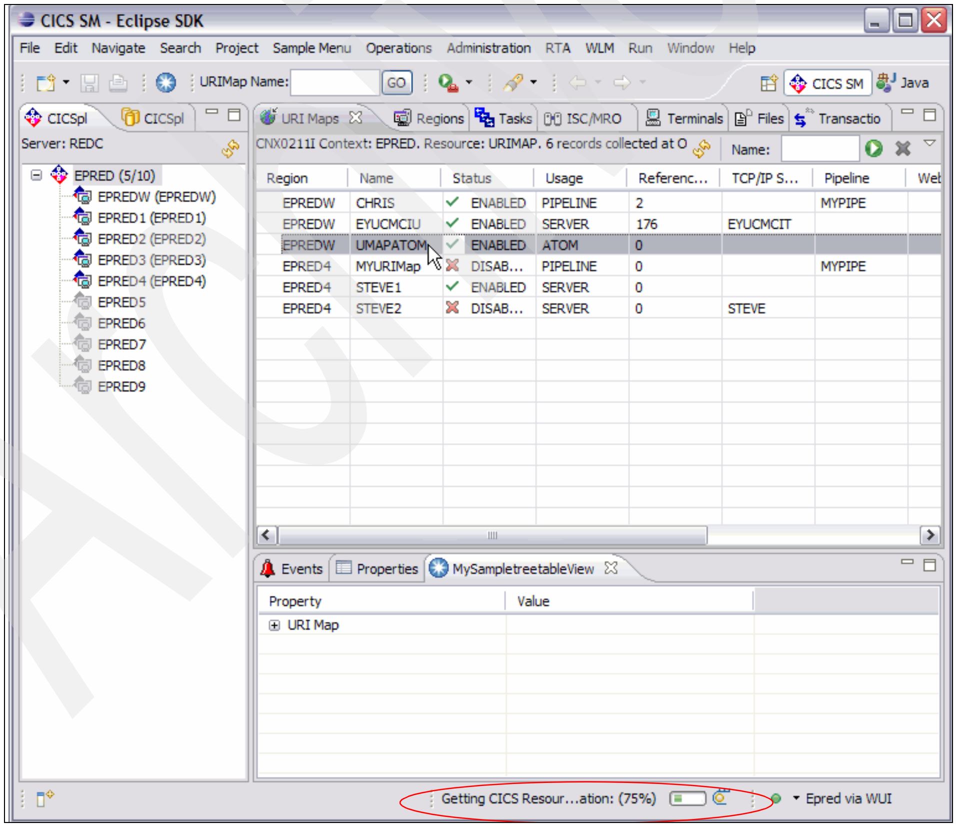
```
public void populateInformation(final String urimapstring) {
   // reserve argument(urimap)
   uriMapString = urimapstring;
   final Job job = new Job("Getting CICS Resource Information") {
      @Override
      protected IStatus run(final IProgressMonitor monitor) {
         // Do your IO stuff here
         if (IO(monitor) == true) {
            Display.getDefault().asyncExec(new Runnable() {
               public void run() { GUI(); } // Do your GUI updates here
            });
         } else {
            Display.getDefault().asyncExec(new Runnable() {
               public void run() {ngGUI(); } // Do your GUI updates here
            });
         }
         monitor.done();
         return Status.OK_STATUS;
      }
   };
   job.schedule();
}
```

Example 5-8 Background processing sample code_2: IO method

```
private boolean IO(IProgressMonitor monitor) {
   monitor.beginTask("Getting CICS Resource Information", 4);
   connectCPSM();
   monitor.worked(1); // progress monitor + 1 -> 1(25%)
   getURIMapResource();
   if (model.size() == 0) { return false; }
   monitor.worked(1); // progress monitor + 1 -> 2(50%)
   getTCPIPServiceResource();
   monitor.worked(1); // progress monitor + 1 -> 3(75%)
   if (iurimap.getUsage().toString() == "SERVER") {
      getProgramResource();
   } else if (iurimap.getUsage().toString() == "CLIENT") {
        // ** no related resources **
    } else if (iurimap.getUsage().toString() == "ATOM") {
        getAtomServiceResource();
    } else if (iurimap.getUsage().toString() == "PIPELINE") {
        getPipelineResource();
        getWebServiceResource();
    }
    monitor.worked(1); // progress monitor + 1 -> 4(100%)
    return true;
}
```

5.5.3 Operation

When you click URIMap view, another thread is generated by the main thread to access the CICS resource information. This new thread is used for this access process, but the main thread is not used and can continue GUI process. You can see the background process progress with the progress bar in the right corner of the window, as shown in Figure 5-33.



CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected at 0

Region	Name	Status	Usage	Referenc...	TCP/IP S...	Pipeline	Wel
EPREDW	CHRIS	✓ ENABLED	PIPELINE	2		MYPIPE	
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	176	EYUCMCIT		
EPREDW	UMAPATOM	✓ ENABLED	ATOM	0			
EPRED4	MYURIMap	✗ DISAB...	PIPELINE	0		MYPIPE	
EPRED4	STEVE1	✓ ENABLED	SERVER	0			
EPRED4	STEVE2	✗ DISAB...	SERVER	0	STEVE		

Property	Value
⊞ URI Map	

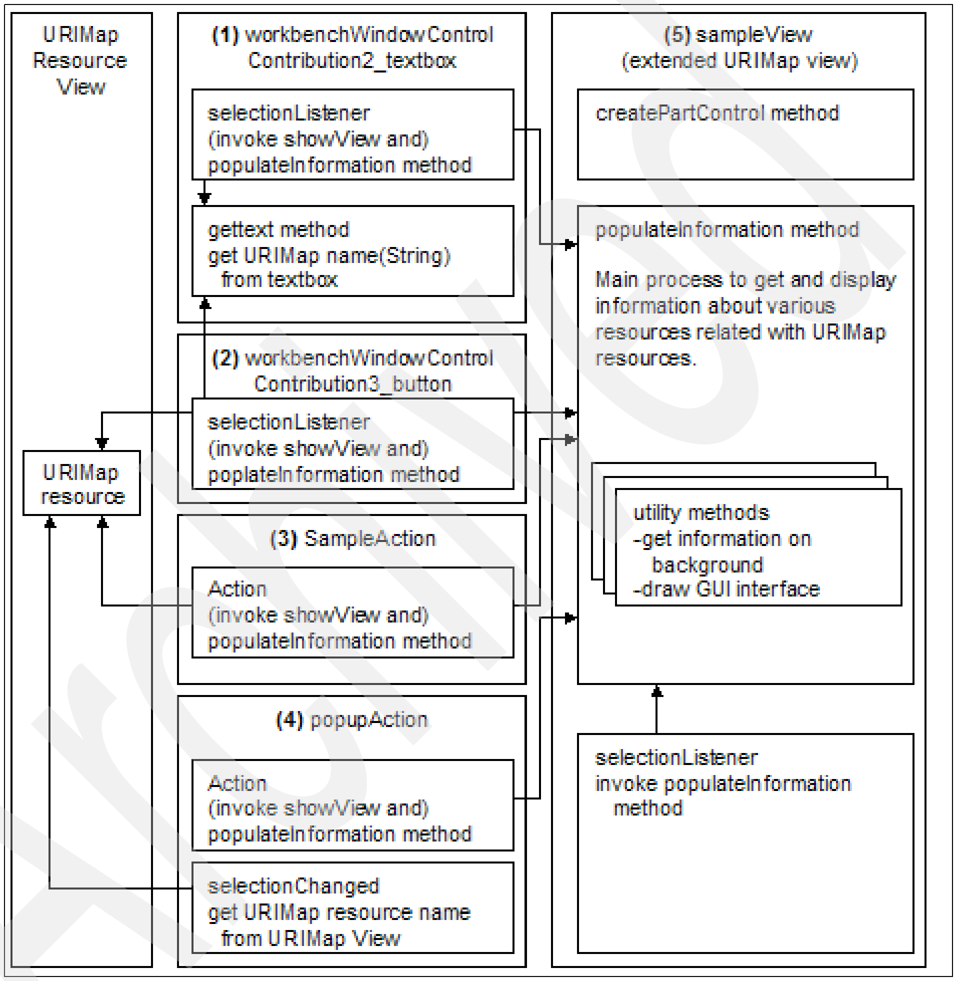
Getting CICS Resour...ation: (75%)

Epred via WUI

Figure 5-33 Background process indicated with progress bar

5.6 Summary of extending functions

All the extending functions described in this chapter can be implemented into one plug-in, summarized in Figure 5-34. This diagram helps you understand the relationship of all functions, objects, classes, and methods easier.



URIMap Resource View
URIMap resource
(1) workbenchWindowControl Contribution2_textbox
selectionListener (invoke showView and) populateInformation method
gettext method get URIMap name(String) from textbox
(2) workbenchWindowControl Contribution3_button
selectionListener (invoke showView and) poplateInformation method
(3) SampleAction
Action (invoke showView and) populateInformation method
(4) popupAction
Action (invoke showView and) populateInformation method
selectionChanged get URIMap resource name from URIMap View
(5) sampleView (extended URIMap view)
createPartControl method
populateInformation method
Main process to get and display information about various resources related with URIMap resources.
utility methods -get information on background -draw GUI interface
selectionListener invoke populateInformation method

Figure 5-34 Summary of extending functions in this chapter

5.7 Package extending functions into a plug-in

When your new plug-in is complete, it can be packaged as a zip file. The zip file can then be distributed, or deployed into your environment. The plug-in zip file is available for:

- CICS Explorer, running environment
- CICS Explorer SDK, developing environment

5.7.1 Package plug-in into zip file

This chapter has described many ways in which you can extend functions as a plug-in. To make the plug-in an archive file, export your plug-in into a zip file using the following steps.

1. When you finish developing your plug-in, you can open the new project which includes the plug-in function. See Figure 5-35.

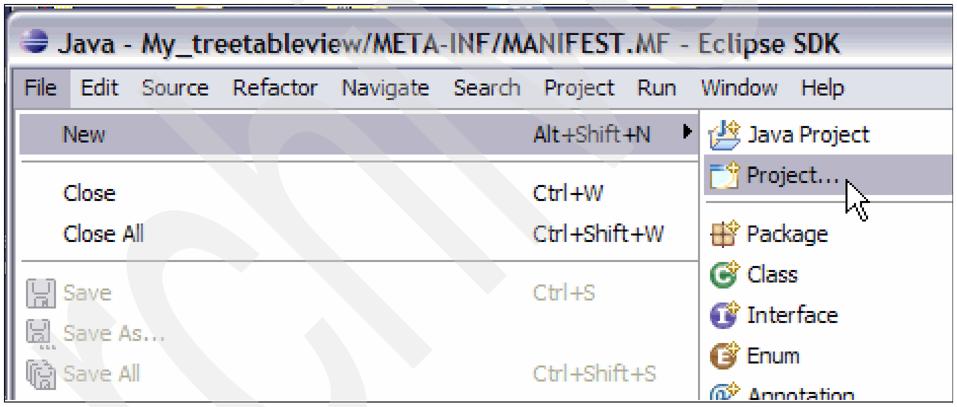


Figure 5-35 Create new project

2. Select **Feature Project** on the “Select a wizard” window (Figure 5-36) to create a feature project. This is an archive plug-in file later.

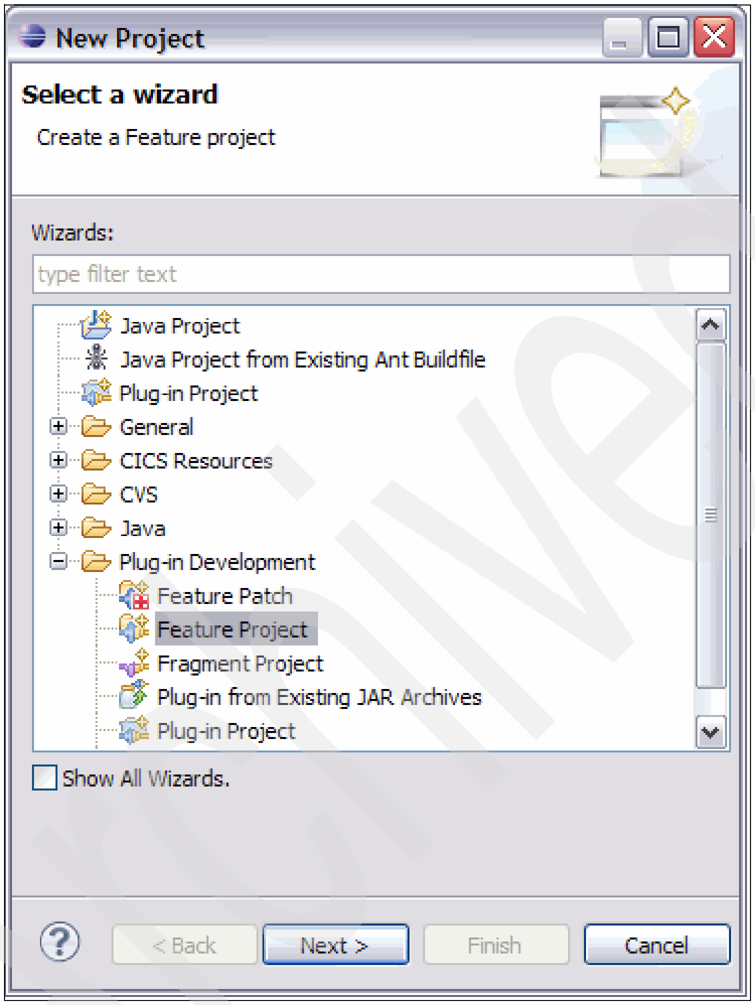
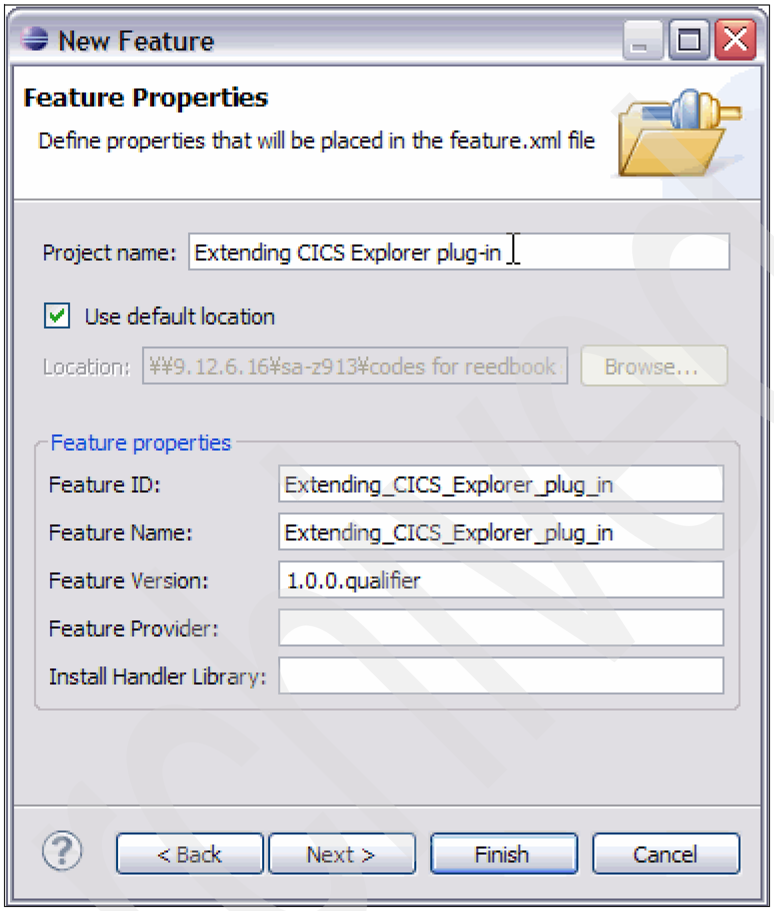


Figure 5-36 Select a Feature project

3. Enter a name in the “Project name” field (Figure 5-37), then click **Next** to specify more information.


New Feature

Feature Properties

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

Project name: Extending CICS Explorer plug-in

[✓] Use default location

Location: ¥¥9.12.6.16¥sa-z913¥codes for reedbook

Feature properties

Feature ID: Extending_CICS_Explorer_plug_in

Feature Name: Extending_CICS_Explorer_plug_in

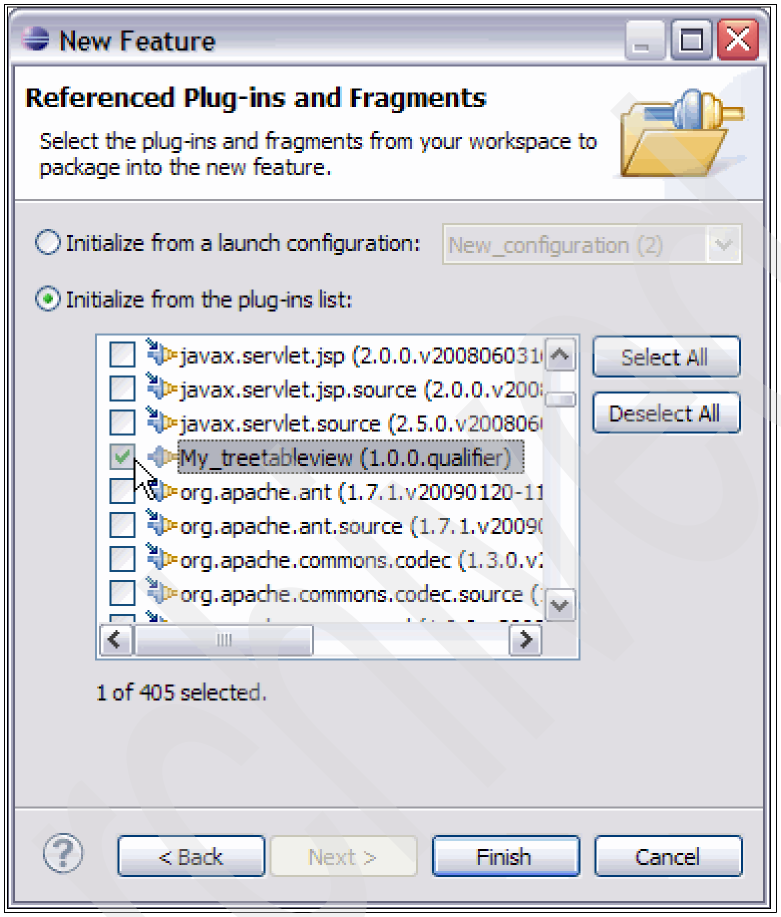
Feature Version: 1.0.0.qualifier

Feature Provider:

Install Handler Library:

Figure 5-37 Feature project properties

4. In the “Referenced the Plug-ins and Fragments” dialog window (Figure 5-38), select and check your plug-in in the plug-ins list. Then, click **Finish**.


New Feature

Referenced Plug-ins and Fragments

Select the plug-ins and fragments from your workspace to package into the new feature.

( ) Initialize from a launch configuration: New_configuration (2)

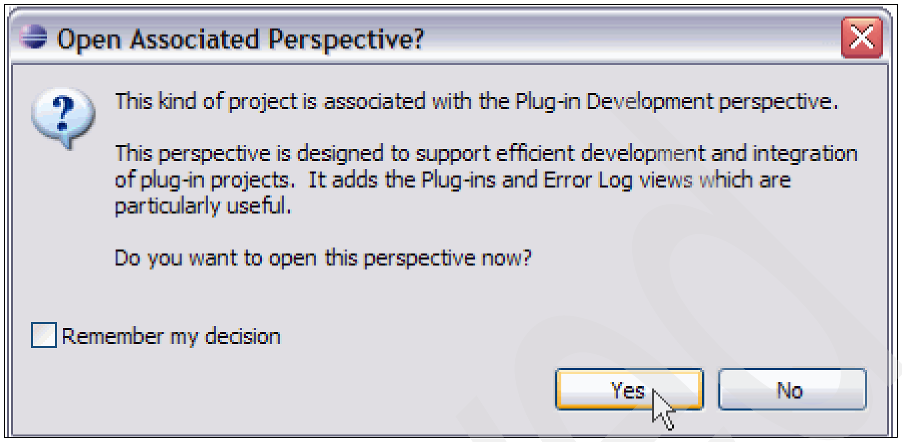
(●) Initialize from the plug-ins list:

- [ ] javax.servlet.jsp (2.0.0.v20080631
- [ ] javax.servlet.jsp.source (2.0.0.v200
- [ ] javax.servlet.source (2.5.0.v200806
- [✓] My_treetableview (1.0.0.qualifier)
- [ ] org.apache.ant (1.7.1.v20090120-11
- [ ] org.apache.ant.source (1.7.1.v2009(
- [ ] org.apache.commons.codec (1.3.0.v
- [ ] org.apache.commons.codec.source (

1 of 405 selected.

Figure 5-38 Select referenced plug-in

5. When the “Open Associated Perspective?” window (Figure 5-39) displays, click **Yes**.


Open Associated Perspective?

This kind of project is associated with the Plug-in Development perspective.

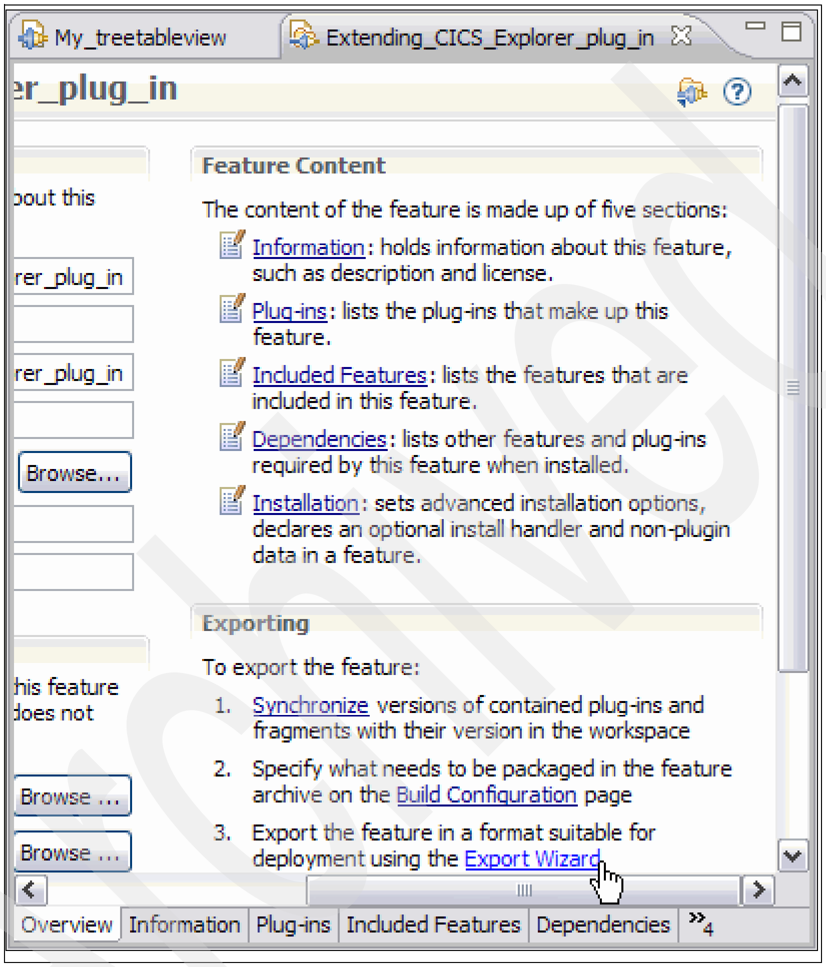
This perspective is designed to support efficient development and integration of plug-in projects. It adds the Plug-ins and Error Log views which are particularly useful.

Do you want to open this perspective now?

Remember my decision

Figure 5-39 Open Associated Perspective dialog

6. The plug-in development perspective (Figure 5-40) displays for this new project. Export the project as an archive file. You can export the project using Export Wizard which is launched from Overview tab.



Feature Content

The content of the feature is made up of five sections:

- Information: holds information about this feature, such as description and license.
- Plug-ins: lists the plug-ins that make up this feature.
- Included Features: lists the features that are included in this feature.
- Dependencies: lists other features and plug-ins required by this feature when installed.
- Installation: sets advanced installation options, declares an optional install handler and non-plugin data in a feature.

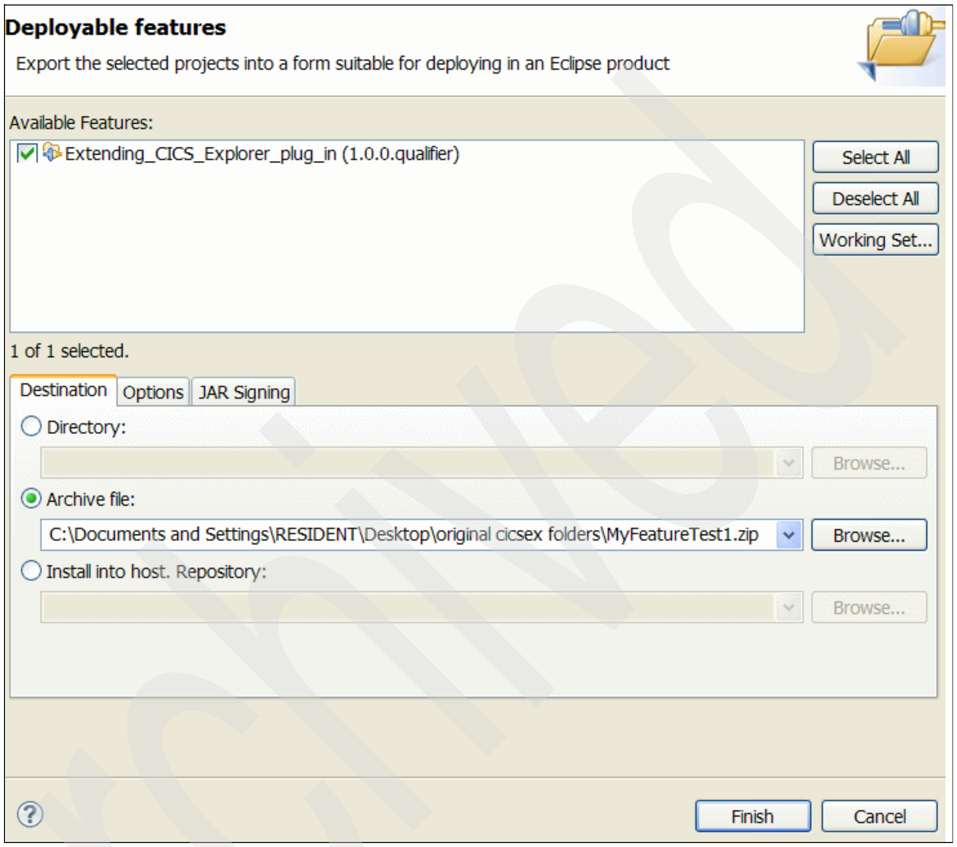
Exporting

To export the feature:

1. Synchronize versions of contained plug-ins and fragments with their version in the workspace
2. Specify what needs to be packaged in the feature archive on the Build Configuration page
3. Export the feature in a format suitable for deployment using the Export Wizard

Figure 5-40 Start Export Wizard to export the feature project

7. In the Deployable features window, confirm selected features, and specify the zip file name and target directory by clicking **Browse**. Then click **Finish** (Figure 5-41).


Deployable features

Export the selected projects into a form suitable for deploying in an Eclipse product

Available Features:

[✓] Extending_CICS_Explorer_plug_in (1.0.0.qualifier)

1 of 1 selected.

Destination | Options | JAR Signing

Directory:

[✓ Archive file:] C:\Documents and Settings\RESIDENT\Desktop\original cicsex folders\MyFeatureTest1.zip

Install into host. Repository:

Figure 5-41 Select Available Features and specify Archive file name

The Export Features window shows the progress of the export. This window disappears automatically.

8. After you have archived your plug-in to an archive file, this zip file is deployable to both CICS Explorer (as running environment) and CICS Explorer SDK (as developing environment).

5.7.2 Deploying plug-in to CICS Explorer

The following procedure details how to deploy your plug-in archive file to a CICS Explorer running environment.

1. Start CICS Explorer, which is installed into your environment.
2. Select **Help** → **Software Updates** on the menu bar (Figure 5-42).

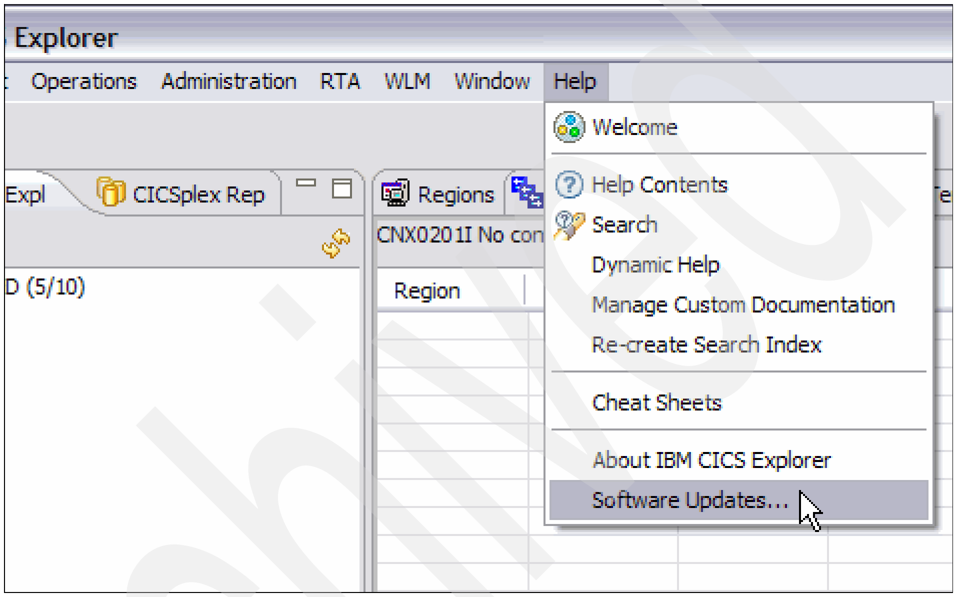
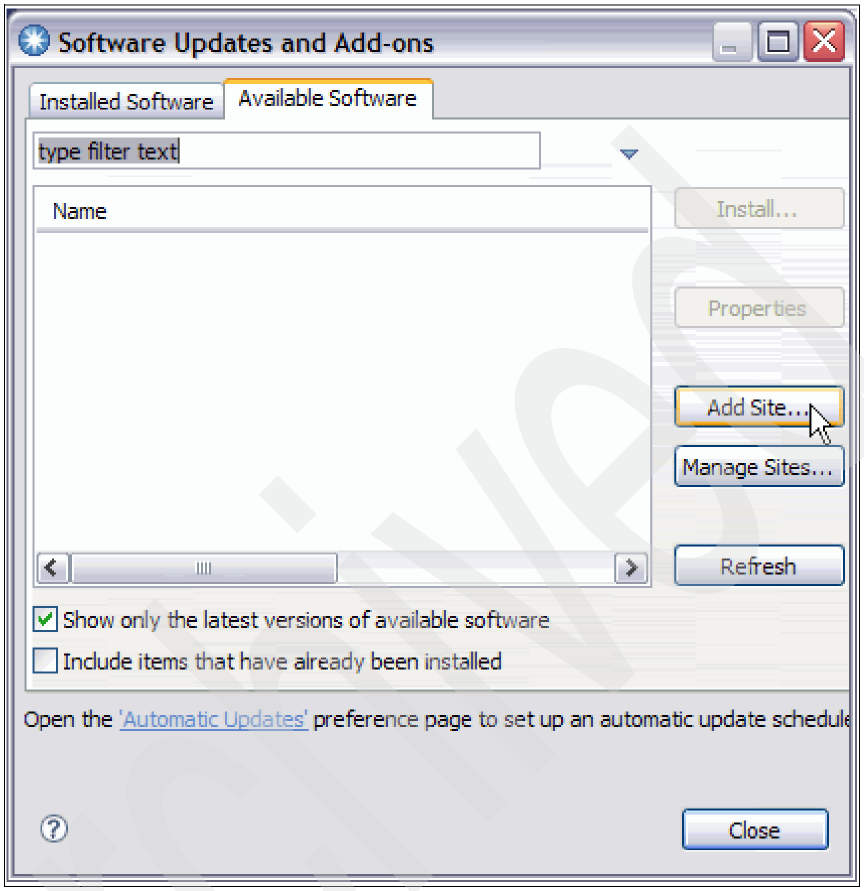


Figure 5-42 Select Help → Software Updates on the menu bar

3. Click **Add Site** to specify plug-in archive file and file path (Figure 5-43).



☑ Show only the latest versions of available software

☐ Include items that have already been installed

Open the 'Automatic Updates' preference page to set up an automatic update schedule

Figure 5-43 Push "Add Site" button to specify the plug-in

4. Click **Archive** to specify the plug-in location as file path and archive file (Figure 5-44).

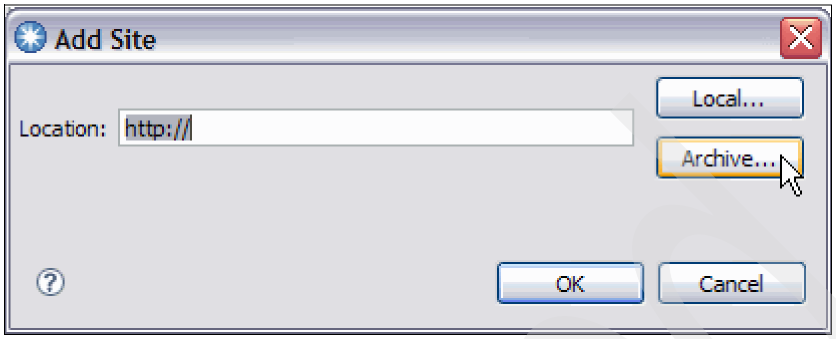


*Figure 5-44 Click **Archive** to specify plug-in archive file*

5. Select your previously created plug-in archive file (Figure 5-45).

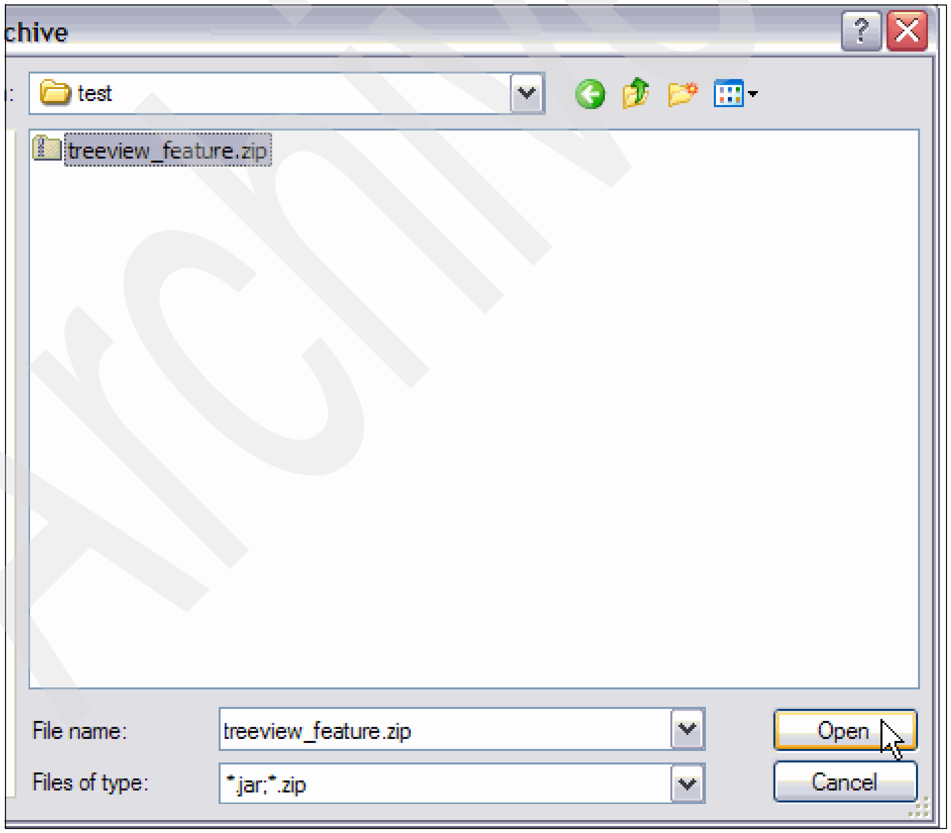
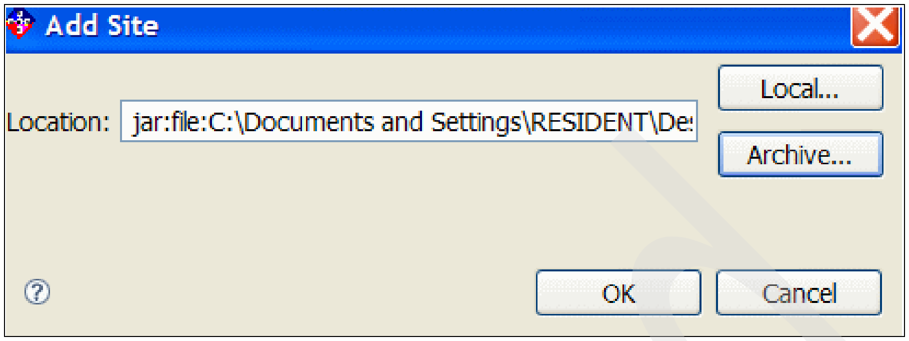


Figure 5-45 Select zip file

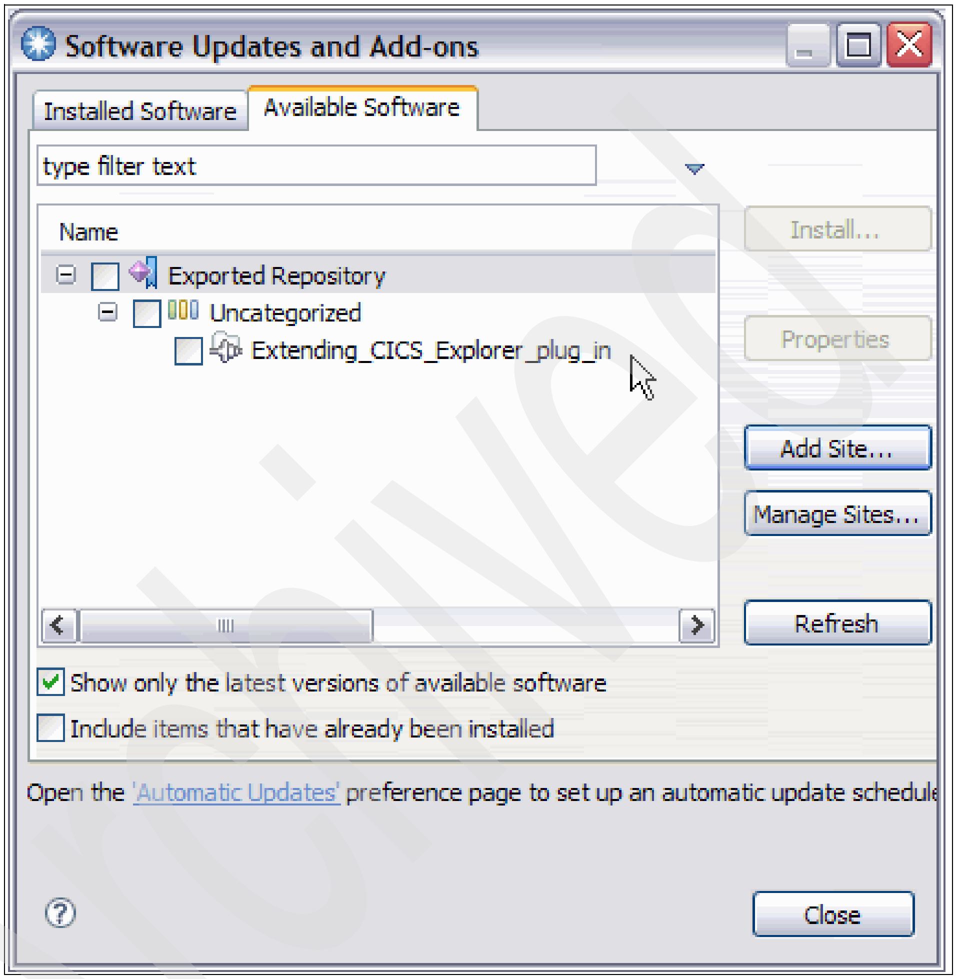
6. Confirm location information and click **OK** (Figure 5-46).



Location: jar:file:C:\Documents and Settings\RESIDENT\De:

Figure 5-46 Click OK to proceed

The deployable plug-in name is shown on the Available Software tab (Figure 5-47).



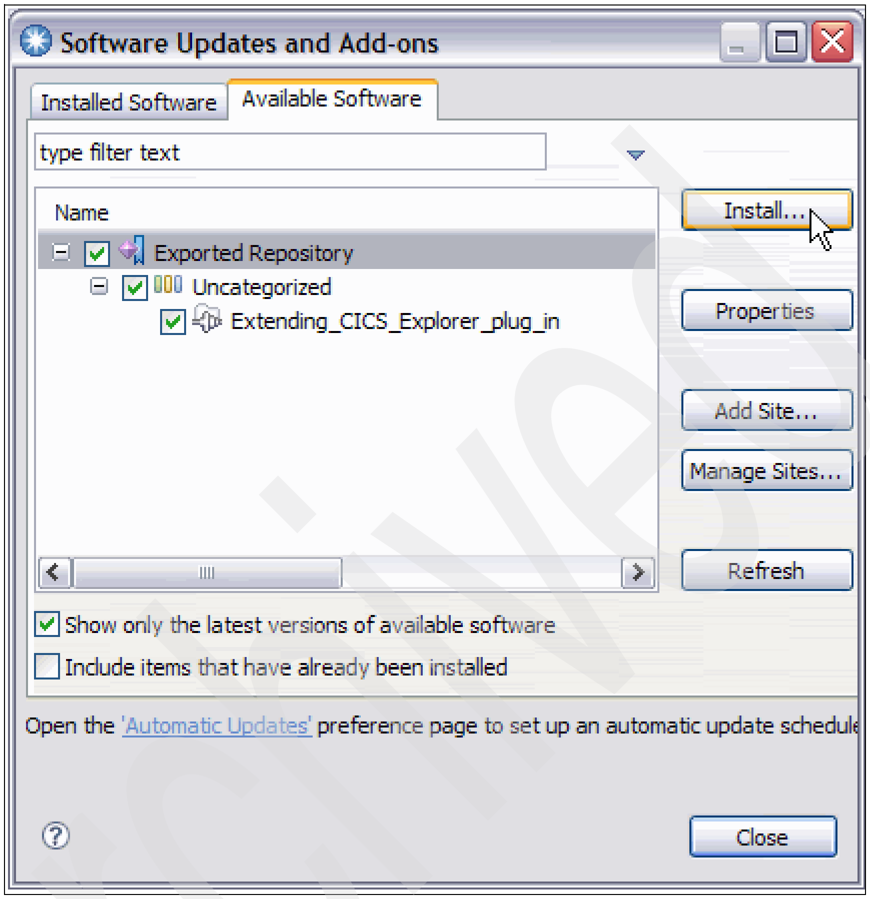
Exported Repository
Uncategorized
Extending_CICS_Explorer_plug_in

[✓] Show only the latest versions of available software
[ ] Include items that have already been installed

Open the 'Automatic Updates' preference page to set up an automatic update schedul

Figure 5-47 Plug-in name is shown

7. Select the plug-in you want to deploy, and click **Install** (Figure 5-48).



Exported Repository
Uncategorized
Extending_CICS_Explorer_plug_in

Show only the latest versions of available software [✓]

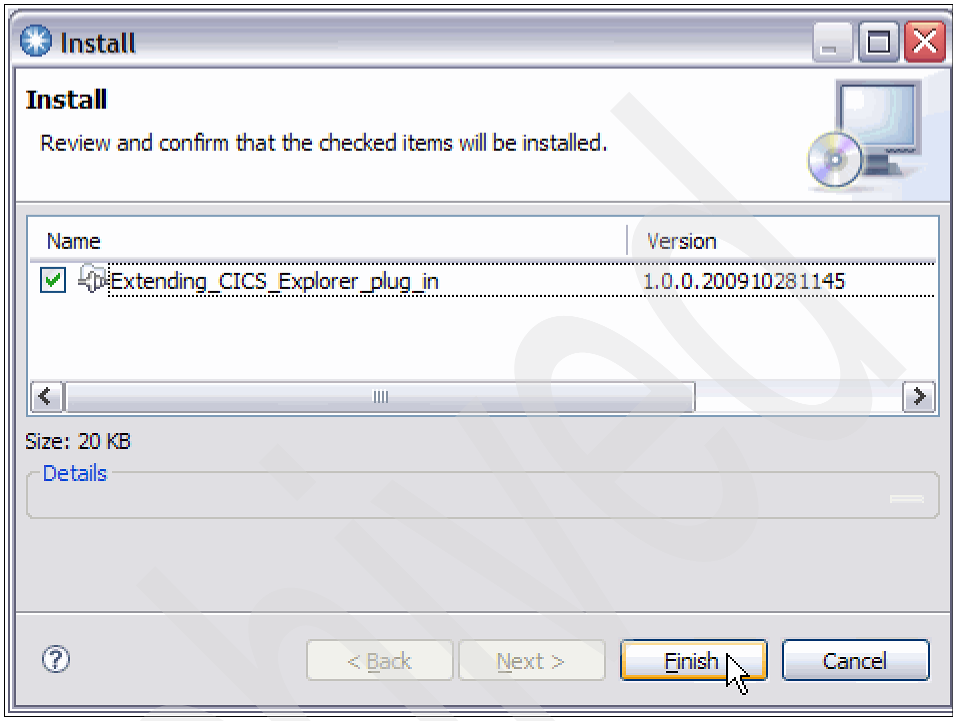
Include items that have already been installed [ ]

Open the 'Automatic Updates' preference page to set up an automatic update schedule

Figure 5-48 Select plug-in

The Progress Information window shows progress of the export. This window disappears automatically.

8. Confirm whether the selected plug-in is the one you want to deploy in the "Install" window (Figure 5-49). If correct, click **Finish** to start install process.



Install

Review and confirm that the checked items will be installed.

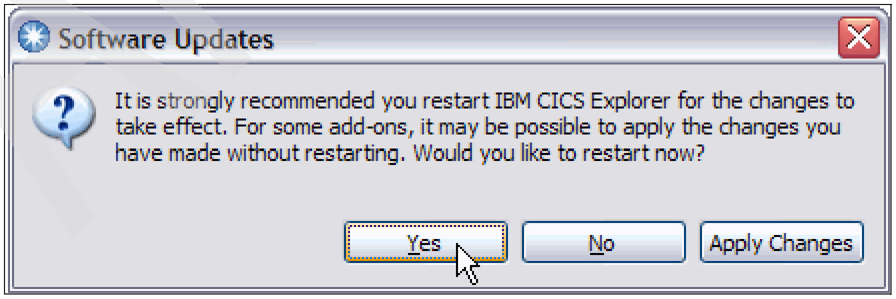
Name	Version
✓ Extending_CICS_Explorer_plug_in	1.0.0.200910281145

Size: 20 KB

Figure 5-49 Confirm plug-in to be installed

The Install window shows installation progress. This window disappears automatically.

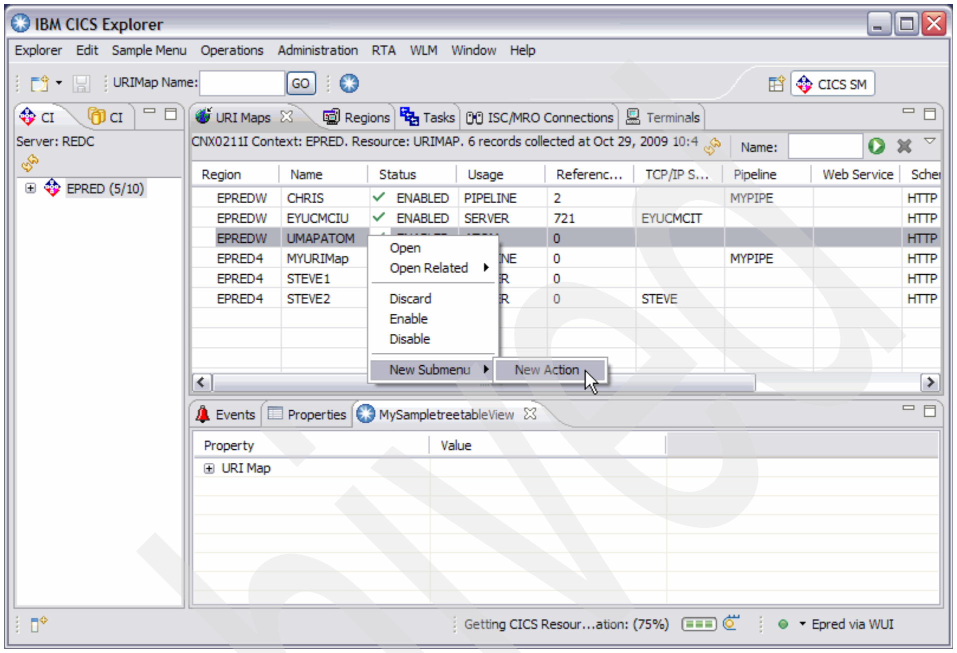
9. Restart CICS Explorer to make the installation effective. Click **Yes** in the Software Updates window (Figure 5-50).



It is strongly recommended you restart IBM CICS Explorer for the changes to take effect. For some add-ons, it may be possible to apply the changes you have made without restarting. Would you like to restart now?

Figure 5-50 CICS Explorer restart confirmation

This completes the deployment of your plug-in into the CICS Explorer perspective (Figure 5-51). You can now use the plug-in function in the CICS Explorer.



CNX0211I Context: EPRED. Resource: URIMAP. 6 records collected at Oct 29, 2009 10:4

Region	Name	Status	Usage	Referenc…	TCP/IP S…	Pipeline	Web Service	Sche
EPREDW	CHRIS	✓ ENABLED	PIPELINE	2		MYPIPE		HTTP
EPREDW	EYUCMCIU	✓ ENABLED	SERVER	721	EYUCMCIT			HTTP
EPREDW	UMAPATOM			0				HTTP
EPRED4	MYURIMap		NE	0		MYPIPE		HTTP
EPRED4	STEVE1		R	0				HTTP
EPRED4	STEVE2		R	0	STEVE			HTTP

Figure 5-51 Plug-in deployed CICS Explorer perspective

Part 4

Integrating CICS Explorer with other Eclipse Components

In this part we write a small collection of plug-ins that display data from an OMEGAMON server. We also describe the necessary steps to create an CICS Explorer plug-in that allows you to set CICS trace levels dynamically for each trace component, similar to the functionality provided by the CICS CETR transaction.

Finally we show you how to extend the functionality provided by the Operations view of the CICS Explorer, to implement a CEBR-like interface which, when you have right-clicked a TS queue, allows you to view the contents of that queue.

6

Combining OMEGAMON data with CICS Explorer

In this chapter we write a small collection of plug-ins that display data from an OMEGAMON server. Although this chapter is aimed specifically at displaying OMEGAMON data, many of the techniques and code samples can be re-used with minimal modifications when accessing data from other products.

First we briefly describe OMEGAMON and the Web services interface that we are using to access the data. We then create a simple plug-in to fetch data from the Web services interface and parse it to find the interesting information.

Next, we create several simple plug-ins that display data that has been fetched from OMEGAMON.

Finally, we add processing to drive an OMEGAMON plug-in to display the OMEGAMON region overview report for a CICS region when that region is selected in the CICS Explorer.

6.1 Environment and configuration

We used the following software levels for this section of this book, in addition to those specified earlier for the CICS Explorer environment:

- IBM Tivoli Monitoring version 6.2 (minimum version 6.1 required for the SOAP interface
- OMEGAMON XE for CICS version 4.1
- For the examples of interaction between CICS Explorer and OMEGAMON we required CICS systems that were both monitored by OMEGAMON for CICS and managed by CPSM. We used CICS TS version 3.2 regions for this, but earlier or later versions work equally well.

6.2 Introduction to OMEGAMON

OMEGAMON XE for CICS on z/OS is a complete monitoring solution that enables you to monitor and manage complex CICS systems by addressing potential problems quickly.

OMEGAMON XE for CICS is part of the IBM Tivoli Monitoring (ITM) architecture and uses the Tivoli Enterprise Portal (TEP) as its user interface. The TEP is a highly customizable Java application that can either be run as a desktop application or run as an applet in a browser. Through this interface, you can monitor all your CICS regions, and the other systems in your enterprise.

Figure 6-1 on page 163 shows a simplified ITM architecture working in conjunction with the CICS Explorer. The CICS regions are monitored by the CICS agents, which pass their data along to the Tivoli Enterprise Monitoring Server (TEMS). The data is presented to the user through the TEP client, which extracts its data from the TEMS through the Tivoli Enterprise Portal Server (TEPS). The CICS Explorer extracts its data from CICS through the CMCI interface. There is also the option of the CICS Explorer accessing CICS data through CPSM, but that is not shown here, because it complicates the diagram further.

The TEP, however, is not an Eclipse application, and as such, does not integrate readily with the CICS Explorer. The ITM architecture also provides a Web services interface. This chapter discusses how we used this interface to access OMEGAMON data in an Eclipse plug-in by sending and receiving SOAP messages. Accessing OMEGAMON data in an Eclipse plug-in makes it easier to use these complementary CICS tools together.

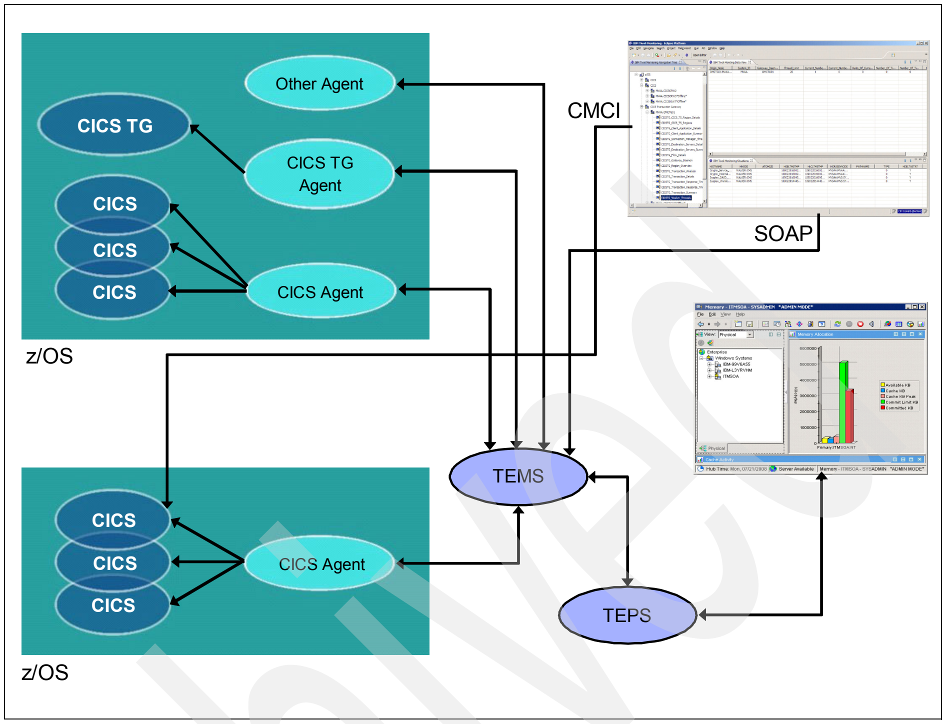


Figure 6-1 The ITM Architecture with the CICS Explorer

6.3 The Tivoli Enterprise Web Services interface

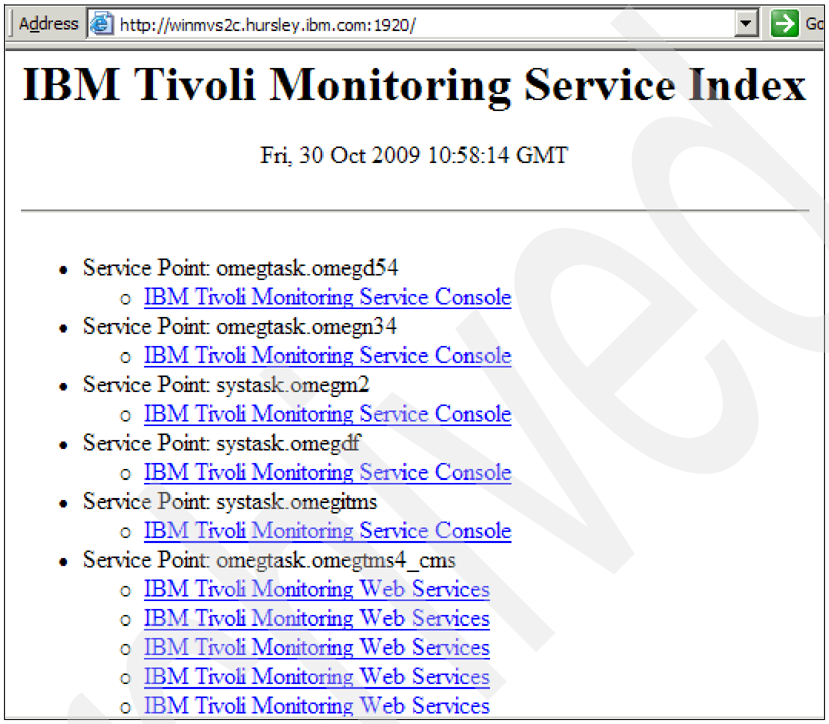
Tivoli Enterprise Web Services is an open interface into Tivoli Monitoring and although it provides access to data from most Tivoli monitoring applications, we are concerned solely with looking at CICS data.

Tivoli Enterprise Web Services implements client/server architecture. The client sends SOAP requests to the Tivoli Monitoring SOAP server, which is part of the TEMS. The server receives and processes the SOAP requests on behalf of the client, and returns with an appropriate response. Although there is much that can be achieved through the Tivoli Web Services interface, this plug-in only uses the CT_Get command, which allows us to request OMEGAMON data from server.

This plug-in communicates with the TEMS through the TEMS SOAP interface, which must be configured as described in the *IBM Tivoli Monitoring: Administrator's Guide*. We do not include support for connections to the SOAP interface over secure (SSL) connections in this chapter.

6.3.1 Connecting to the TEMS SOAP interface with the Web client

Note the host name (or IP address) and port number for the SOAP interface, as this is required later. When we point a browser at the host system containing the TEMS (and port 1920), we see a view similar to that shown in Figure 6-2.



IBM Tivoli Monitoring Service Index

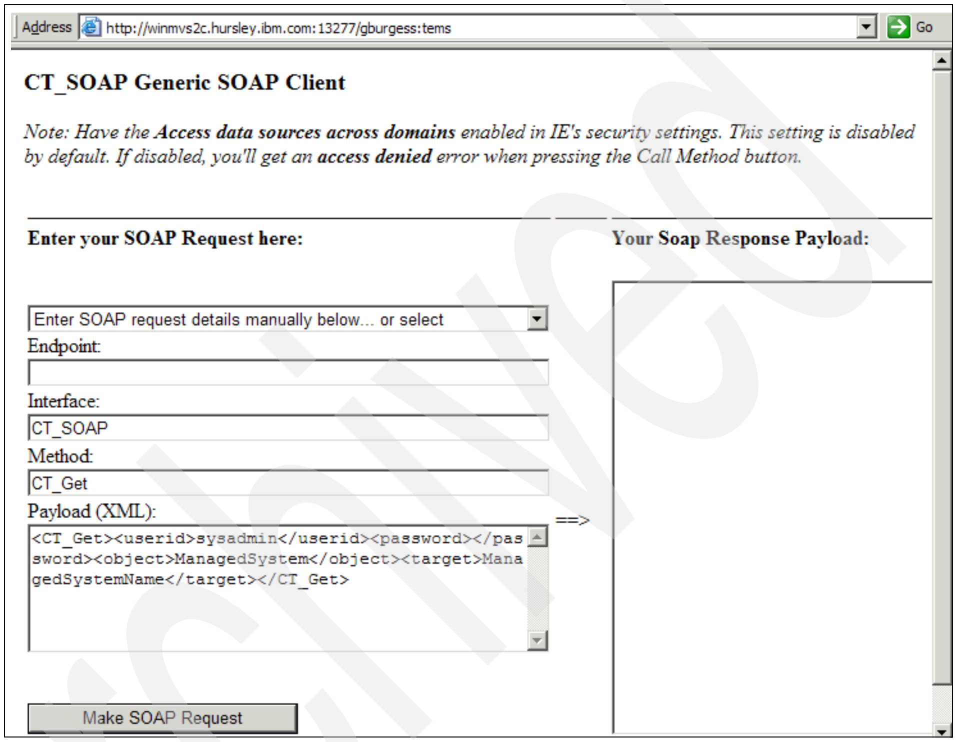
Fri, 30 Oct 2009 10:58:14 GMT

- Service Point: omegtask.omegd54
  - IBM Tivoli Monitoring Service Console
- Service Point: omegtask.omegn34
  - IBM Tivoli Monitoring Service Console
- Service Point: systask.omegm2
  - IBM Tivoli Monitoring Service Console
- Service Point: systask.omegdf
  - IBM Tivoli Monitoring Service Console
- Service Point: systask.omegitms
  - IBM Tivoli Monitoring Service Console
- Service Point: omegtask.omegtms4_cms
  - IBM Tivoli Monitoring Web Services
  - IBM Tivoli Monitoring Web Services
  - IBM Tivoli Monitoring Web Services
  - IBM Tivoli Monitoring Web Services
  - IBM Tivoli Monitoring Web Services

Figure 6-2 ITM Service Index list

When we click one of the listed IBM Tivoli Monitoring Web Services we see the Tivoli generic SOAP client, as shown in Figure 6-3.

We can use this simple Web client to test the TEMS SOAP interface and to try out commands that we might like to implement in the Eclipse plug-ins that we are implementing.


Address http://winmvs2c.hursley.ibm.com:13277/gburgess:tems

CT_SOAP Generic SOAP Client

Note: Have the ***Access data sources across domains*** *enabled in IE's security settings. This setting is disabled by default. If disabled, you'll get an* ***access denied*** *error when pressing the Call Method button.*

Enter your SOAP Request here:

Enter SOAP request details manually below... or select

Endpoint:

Interface: CT_SOAP

Method: CT_Get

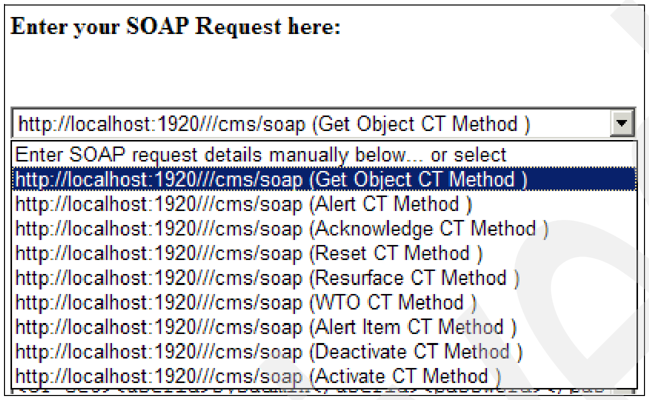
Payload (XML):

```
<CT_Get><userid>sysadmin</userid><password></pas
sword><object>ManagedSystem</object><target>Mana
gedSystemName</target></CT_Get>
```

Your Soap Response Payload:

Figure 6-3 ITM Generic SOAP client

We can issue a simple query against the TEMS server's SOAP interface using the Web client. We shall issue a request for the Managed System List. This is a list of the Tivoli monitoring systems and the systems they are monitoring (such as your CICS regions). The Web client offers this query as one of its predefined queries. The list of queries is shown in Figure 6-4.



http://localhost:1920///cms/soap (Get Object CT Method)

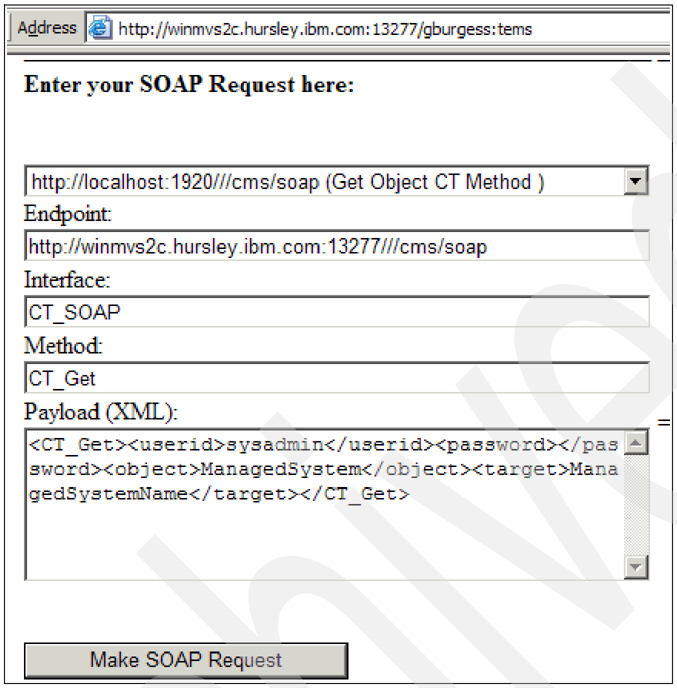
Enter SOAP request details manually below... or select
http://localhost:1920///cms/soap (Get Object CT Method)
http://localhost:1920///cms/soap (Alert CT Method)
http://localhost:1920///cms/soap (Acknowledge CT Method)
http://localhost:1920///cms/soap (Reset CT Method)
http://localhost:1920///cms/soap (Resurface CT Method)
http://localhost:1920///cms/soap (WTO CT Method)
http://localhost:1920///cms/soap (Alert Item CT Method)
http://localhost:1920///cms/soap (Deactivate CT Method)
http://localhost:1920///cms/soap (Activate CT Method)

Figure 6-4 List of queries available in the SOAP Web client

We select the option for the `Get Object CT` Method (CT_GET), and this fills in the other entry text boxes for us. The endpoint text box gets filled in with the default URL for a SOAP server running on the local machine.

As we are not connecting to a server on the local machine, we have to overwrite this value with the host and port from the browser's address bar. We then look at the Payload text box, which contains the message that is sent to the SOAP sever.

Because the TEMS server to which we are connecting does not have security enabled, we do not have to change the user ID from “sysadmin”, or specify a password. This gives us a request ready to be issued, as shown in Figure 6-5.


Address http://winmvs2c.hursley.ibm.com:13277/gburgess:tems

Enter your SOAP Request here:

http://localhost:1920///cms/soap (Get Object CT Method)

Endpoint: http://winmvs2c.hursley.ibm.com:13277///cms/soap

Interface: CT_SOAP

Method: CT_Get

Payload (XML):

```
<CT_Get><userid>sysadmin</userid><password></pas
sword><object>ManagedSystem</object><target>Mana
gedSystemName</target></CT_Get>
```

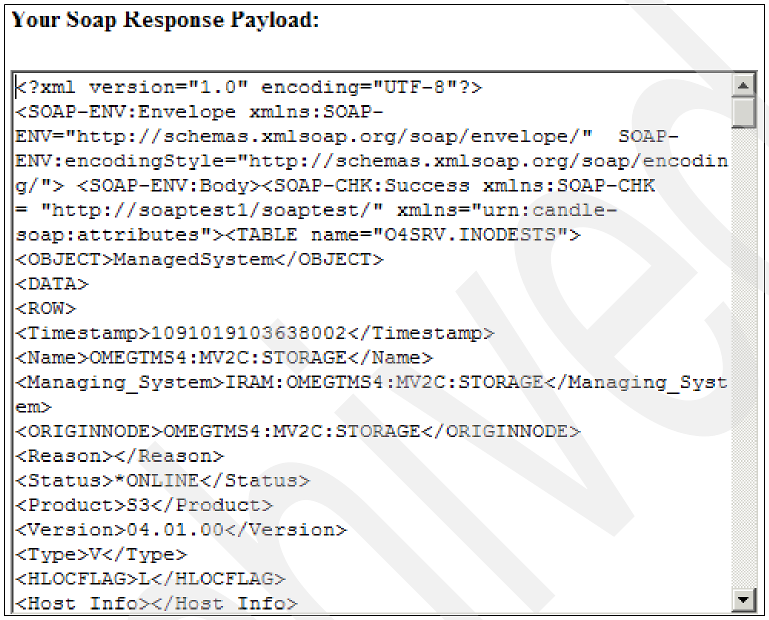
Figure 6-5 Web client Managed System List request

Click **Make SOAP Request** and the client issues the SOAP request to the server and displays the response. The response can be quite large, depending on how many Tivoli monitoring systems you have and how many systems they are monitoring. The start of our results are shown in Figure 6-6 on page 168.

We can see in these results that our request was successful as the response contains the text `SOAP-CHK:Success` near the start of the message, and we have a long response made up of numerous XML fields containing information about our managed systems.

The Web client does not parse this XML into a more meaningful format, such as a table for us, but we write suitable parsing and formatting code as part of our Eclipse plug-ins.

Tip: The SOAP Web client can be a useful tool when you are trying to test a new request against the SOAP interface, as it can be invoked from a browser without having to write any code. You can see exactly what response is returned from the server.



Your Soap Response Payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"  SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encodin
g/"> <SOAP-ENV:Body><SOAP-CHK:Success xmlns:SOAP-CHK
= "http://soaptest1/soaptest/" xmlns="urn:candle-
soap:attributes"><TABLE name="O4SRV.INODESTS">
<OBJECT>ManagedSystem</OBJECT>
<DATA>
<ROW>
<Timestamp>1091019103638002</Timestamp>
<Name>OMEGTMS4:MV2C:STORAGE</Name>
<Managing_System>IRAM:OMEGTMS4:MV2C:STORAGE</Managing_Syst
em>
<ORIGINNODE>OMEGTMS4:MV2C:STORAGE</ORIGINNODE>
<Reason></Reason>
<Status>*ONLINE</Status>
<Product>S3</Product>
<Version>04.01.00</Version>
<Type>V</Type>
<HLOCFLAG>L</HLOCFLAG>
<Host_Info></Host_Info>
```

Figure 6-6 Extract from the SOAP Web client response

We have issued a simple request against the SOAP interface, we can be confident that our SOAP interface is working, and that we know exactly which host name and port on which it is listening for requests. Now we can move on to writing an Eclipse plug-in that issues the same request, and build upon that plug-in.

6.4 A simple OMEGAMON SOAP interface plug-in

We start with a simple plug-in that can connect to the TEMS SOAP interface, issue a request, and receive the returned SOAP data. We then add the ability to parse the returned data.

We request the Managed System List. This is a list of all the systems that are known to the Tivoli monitoring system, and the Tivoli servers that make up the monitoring architecture. The returned data includes:

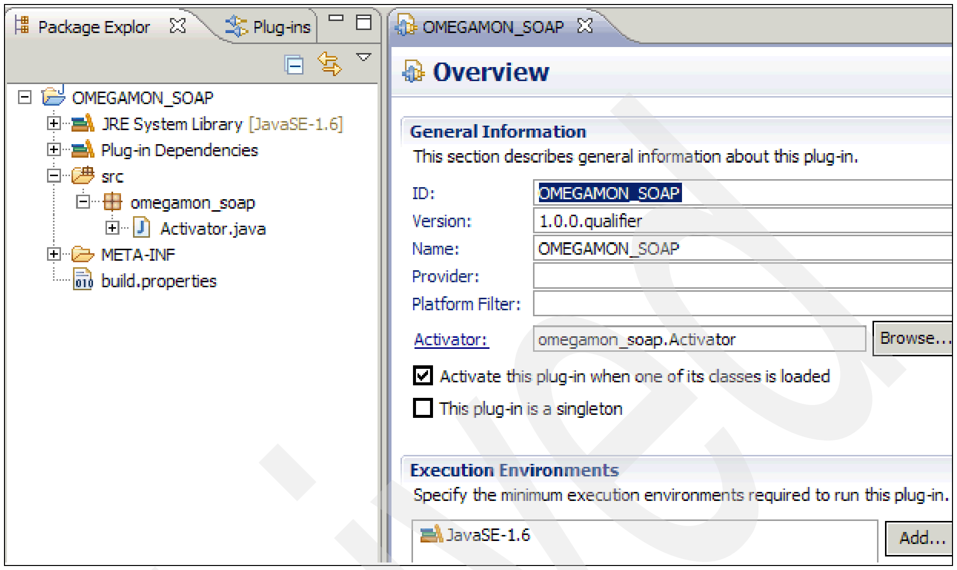
- The name of the monitored system
- An indicator of whether the system is online or offline
- A product code identifier that identifies the Tivoli monitoring product

The returned list includes products other than CICS if you have other Tivoli monitoring products installed

The plug-in is written with a JUnit (v3) test harness to allow us to build up the code in small chunks, explaining and testing each as we go. This plug-in initially does the minimum required to issue a request to the SOAP server and receive a response. We then build on this functionality.

First, we create the plug-in project. We can create the plug-in using the plug-in wizard (Navigate to **File** → **New** → **project** and select **Plug-in project** in the New Project dialog box). We name the plug-in `OMEGAMON_SOAP` and accept the defaults, with one exception. Clear the **This plug-in will make contributions to the UI** option, and we do not use a template for the plug-in. This plug-in does not make any contributions to the user interface, because its role is to capture data that is presented to the user by other plug-ins.

After we have created the plug-in project, the workspace looks something like Figure 6-7.


Package Explor | Plug-ins | OMEGAMON_SOAP

OMEGAMON_SOAP
- JRE System Library [JavaSE-1.6]
- Plug-in Dependencies
- src
  - omegamon_soap
    - Activator.java
- META-INF
- build.properties

Overview

General Information

This section describes general information about this plug-in.

ID: OMEGAMON_SOAP
Version: 1.0.0.qualifier
Name: OMEGAMON_SOAP
Provider:
Platform Filter:
Activator: omegamon_soap.Activator [Browse...]

[✓] Activate this plug-in when one of its classes is loaded
[ ] This plug-in is a singleton

Execution Environments

Specify the minimum execution environments required to run this plug-in.

JavaSE-1.6 [Add...]

Figure 6-7 Newly created OMEGAMON_SOAP plug-in

We now need to create the class that fetches the data from the Web services interface. We create a new class SoapInterface in the 'omegamon_soap' package. See Example 6-1.

Example 6-1 SoapInterface class listing

```
package omegamon_soap;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ConnectException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.SocketException;
import java.net.URL;

public class SoapInterface {

   public static final String MANAGED_SYSTEM_LIST_REQUEST = 
"<object>ManagedSystem</object><target>ManagedSystemName</target>";
                     //(1)

   private String ctGet;
   private URL url;

   public SoapInterface(String hostName, int port, String userid, String
password) {            //(2)

      if (password == null) {
         password = "";
      }
      ctGet = "<CT_Get><userid>" + userid + "</userid><password>" +
               password + "</password>";
      try {
         this.url = new URL("http://" + hostName + ":" + port +
"///cms/soap");
      }
      catch (MalformedURLException e) {
         e.printStackTrace();
      }
   }


   /**
    * issue a request for some data from the SOAP interface and return
    * the response
    * @param payload the request to be issued
    * @param url the url of the SOAP interface we wish to connect to
    * @return the results of the request
    */
   public String getData(String request) {
                                                                    //(3)
      String payload = ctGet + request + "</CT_Get>";
      HttpURLConnection urlConnection = null;
       try {
       // create a HTTP connection configure it for a SOAP request
       urlConnection = (HttpURLConnection) url.openConnection();
       urlConnection.setRequestMethod("POST");
       urlConnection.setDoOutput(true);

       // issue our request
       urlConnection.getOutputStream().write(payload.getBytes());

       // ensure the request has been issued and close the output stream
       urlConnection.getOutputStream().flush();
       urlConnection.getOutputStream().close();

       // read the response from the SOAP interface
           InputStream in = urlConnection.getInputStream();
           BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
       StringBuffer result = new StringBuffer();
```

```
        String line;
            do {
                line = reader.readLine();
                if (line != null) {
                result.append(line);
                }
            } while ( line !=  null );

        return result.toString();
        }
        catch (ConnectException e) {
        System.out.println("Connect exception - check host and port");
        e.printStackTrace();
        return null;
    }
        catch (SocketException e) {
        System.out.println("Socket exception- check host and port");
        e.printStackTrace();
        return null;
        }
        catch (Exception e)
    {
        System.out.println("Other exception.");
        e.printStackTrace();
        return null;
    }
        finally
        {
        // always clean up before we leave
            if (urlConnection != null) {
            urlConnection.disconnect();
            }
        }

    }
}
```

Notes on Example 6-1 on page 170:

- The static final String MANAGED_SYSTEM_LIST_REQUEST contains a simple request for a list of managed systems from the Web services interface
- The constructor builds up the URL for the Web service and constructs a ‘CT_GET’ request String that can be combined with a request for data to drive the Web service.
- The `getData(…)` method constructs a request for data, sends it to the Web service, and returns the reply from the Web service. This reply is in the form of an XML String.

We now want to drive this code to ensure that it works. We do this with a JUnit test class.We create the JUnit test class in the omegamon_soap_tests package. This is shown in Example 6-2.

Example 6-2 SoapInterfaceTester class listing

```
/* Get the value of ARRAY_VAL. Note: arrayIndex starts at 0 */
package omegamon_soap_tests;

import omegamon_soap.SoapInterface;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class SoapInterfaceTester extends TestCase {

   // useful test data

   /**
    * create a test suite for the tests in this class
    * @return the test suite
    */
   public static Test suite() {
      return new TestSuite(SoapInterfaceTester.class);
   }


   @Override
   protected void setUp() throws Exception {
      // TODO Auto-generated method stub
      super.setUp();
   }

   public void testNothing() {

   }

   public void testSoapRequest() {
      SoapInterface soapInterface = new SoapInterface("winmvs2c.hursley.ibm.com",
47587, "sysadmin", "");
      String result = soapInterface.getData(SoapInterface.MANAGED_SYSTEM_LIST_REQUEST);
      assertNotNull("Result should not be null", result);
      assertTrue("Result length should be greater than zero", result.length() > 0);
      assertTrue("SOAP request failed. result {" + result + "}",
result.contains("<SOAP-CHK:Success"));
   }

   @Override
   protected void tearDown() throws Exception {
      // TODO Auto-generated method stub
      super.tearDown();
   }
}
```

You have to change the host name, port, user ID and password to match those for your system. These values are those used in 6.3.1, “Connecting to the TEMS SOAP interface with the Web client” on page 164.

This test code merely creates an instance of the SoapInterface class and issues a request for a managed system list. The returned data is checked to ensure that something is returned, and that the returned data includes the <SOAP-CHK:Success string. If this string is found, the request has been processed successfully.

6.4.1 Parsing the returned data

Now that we have data being returned from OMEGAMON, we need to parse it to find the interesting data, and to convert it into a more useful format. Because we are reading the complete SOAP reply before processing it, we can use the DOM SOAP parser, which simplifies the parsing code.

The XML parsing code is contained in a new SoapXMLParser class in the omgamon_soap package. The XML parsing reuses a single DocumentBuilder instance, so this can be declared as a static instance field and initialized in a static code block. This means that the SoapXMLParser class starts off looking like the listing shown in Example 6-3.

Example 6-3 SoapXMLParser static code

```
public class SoapXMLParser {
   // the DocumentBuilder will enable us to parse the XML easily
   private static DocumentBuilder documentBuilder;

   /**
    * Create the document builder that will be used to parse the SOAP data
    */
   static {
      // Create the factory that will be used to create the document builder
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      try {
         // and create the document builder
         documentBuilder = dbf.newDocumentBuilder();
      }
      catch (ParserConfigurationException e) {
         System.out.println(e);
         e.printStackTrace();
      }
   }
}
```

Note: We are reusing a single DocumentBuilder instance to parse all our XML. This might not be a good idea if you expect to be fetching data on multiple threads that might overlap, and would be bad if you wanted to do any parsing outside of the constructor call as there might be a clash with another instance using the same DocumentBuilder

The parser extracts all the interesting data from the XML when the constructor is invoked, so we do not need instance fields to store the XML, but we do need instance fields to store the interesting information extracted from the XML. We add the following instance fields to the class:

```
// the parsed data
private String[][] dataTable;
private String[] columnNames;
private int rowCount;
```

Next, we need the constructor that accepts XML to be parsed as its input, and initialize the instance fields with data extracted from the XML. This method is shown in Example 6-4.

Example 6-4 SoapXMLParser constructor

```
/**
 * parse the passed String of XML data
 * @param xml the data to be parsed
 * @throws Exception thrown if we fail to parse the passed data
 */
public SoapXMLParser(String xml) throws Exception {
        NodeList nodeList = null;
        try {
                // load the XML String into an XML Document
                Document document = documentBuilder.parse(
            new ByteArrayInputStream(xml.getBytes("UTF-8")));

                // first find the root of the XML document
                Element root = document.getDocumentElement();

                // extract the interesting data as a list of rows
                nodeList = root.getElementsByTagName("ROW");
        }
        catch (DOMException dom) {
                System.out.println(dom.getMessage());
                dom.printStackTrace();
                throw dom;
        }
        catch (IOException ioe) {
                System.out.println(ioe);
                ioe.printStackTrace();
                throw ioe;
        }

        // now construct a table from the XML document
        rowCount = nodeList.getLength();
        if (rowCount > 0) {
                dataTable = new String[rowCount][];
                for (int i = 0; i < rowCount; i++) {
                        Node rowNode = nodeList.item(i);
                        dataTable[i] = getRowValues(rowNode);
                }
                // extract the column headers
                columnNames = getRowFieldNames(nodeList.item(0));
        }
}
```

This method introduces compile errors as we have not yet added the methods it calls. One of the advantages of parsing the XML in the constructor is that we know that if any problems are found, then we are not left with an invalid instance of the class. The constructor either completes successfully with valid data or throws an Exception. Extracting the data we want from the XML is implemented in two methods:

- `getRowValues(…)`
- `getRowFieldNames(…)`

These methods are shown in Example 6-5.

Example 6-5 XML data extraction methods

```
/**
 * return the passed node's values as an array of Strings
 * @param node the Node to be processed
 * @return a String[] containing the Node's values
 */
private static String[] getRowValues(Node node) {
        NodeList children = node.getChildNodes();
        String[] values = null;
        if (children != null && children.getLength() > 0) {
                int childCount = children.getLength();
                values = new String[childCount];

                for (int i = 0; i < childCount; i++) {
                        Node childNode = children.item(i);
                        Text valueNode = (Text)
childNode.getFirstChild();
                        if (valueNode != null) {
                                values[i] = valueNode.getNodeValue();
                        }
                }
        }
        return values;
}


/**
 * return the names of the nodes that make up the passed row.
 * Effectively, these are the column headers for a table containing
this row.
 * @param row the row from which to extract the names
 * @return a list of names, or null if the row has no elements
 */
private String[] getRowFieldNames(Node row) {
        NodeList children = row.getChildNodes();
        if (children != null && children.getLength() > 0) {
                int childCount = children.getLength();
                String[] values = new String[childCount];
                for (int i = 0; i < childCount; i++) {
                        Node child = children.item(i);
                        values[i] = child.getNodeName();
                }
                return values;
        }
        else {
                return null;
        }
}
```

These two methods are quite similar, which is unsurprising as they both walk through the XML document looking for fields of interest. There is other data in the returned SOAP message, such as the data type of the returned data, but we do not extract that data here. These plug-ins treat all returned data as strings. If we were to add code that checks the data types, then we could implement better formatting of the returned data, such as digit grouping for numbers and display of timestamps as date-times rather than long strings of digits.

Finally, we need getter methods to expose the parsed data to any callers. These are shown in Example 6-6. There is no need for `setter` methods, as these fields are initialized in calls from the constructor.

Example 6-6 SoapXMLParser getter methods

```
public String[][] getDataTable() {
   return dataTable;
}
public int getRowCount() {
   return rowCount;
}
public String[] getColumnNames() {
   return columnNames;
}
```

In the writing of this parsing code, we wrote unit test code. This code was implemented as another JUnit test class, as shown in <REF SoapXMLParserTester class>. The test data (field testData1) was copied from the returned data in a request issued using the SOAP Web client, as described in Example 6-7.

Example 6-7 SoapXMLParserTester class

```
package omegamon_soap_tests;

import omegamon_soap.SoapXMLParser;
import junit.framework.TestCase;

public class SoapXMLParserTester extends TestCase {

   // useful test data
   private String testData1
   = "<?xml version=\"1.0\" encoding=\"UTF-8\"?><SOAP-ENV:Envelope " +
   "xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\"  " +
   "SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/" +
   "encoding/\"> <SOAP-ENV:Body><SOAP-CHK:Success xmlns:SOAP-CHK " +
   "= \"http://soaptest1/soaptest/\" xmlns=\"urn:candle-soap:" +
   "attributes\"><TABLE name=\"O4SRV.ISITSTSH\"><OBJECT>" +
   "O4SRV.ISITSTSH</OBJECT><DATA>" +
   "<ROW><HSITNAME>KS3_Vol_Fragment_Index_Critical" +
   "</HSITNAME><HNODE>GBURGESS</HNODE><ATOMIZE>" +
   "</ATOMIZE><HGBLTMSTMP>1090906205846000</HGBLTMSTMP>" +
   "<HORIGINNODE>OMEGTMS4:MV2C:STORAGE</HORIGINNODE>" +
   "<PATHNAME></PATHNAME><TYPE dt=\"number\">0</TYPE></ROW>" +
   "<ROW><HSITNAME>KS3_LCU_IO_Rate_Sec_Warning</HSITNAME>" +
   "<HNODE>GBURGESS</HNODE><ATOMIZE></ATOMIZE><HGBLTMS"+
   "TMP>1090924201345001</HGBLTMSTMP><HORIGINNODE>OMEGTMS4:"+
   "MV2C:STORAGE</HORIGINNODE><PATHNAME></PATHNAME><TYPE" +
   " dt=\"number\">0</TYPE></ROW></DATA></TABLE>" +
   "</SOAP-CHK:Success></SOAP-ENV:Body></SOAP-ENV:Envelope>";

   public void testConstructor() {
      try {
         SoapXMLParser parser1 = new SoapXMLParser(testData1);
         assertNotNull("Null parser", parser1);
         assertEquals("Incorrect row count. Expected 2. Found " +
            parser1.getRowCount(), 2, parser1.getRowCount());

         String[][] dataTable = parser1.getDataTable();
         assertNotNull("Null data table", dataTable);
         assertEquals("Incorrect table row count. Expected 2. Found " +
            dataTable.length, 2, dataTable.length);

         String[] firstRow = dataTable[0];
         assertNotNull("Null First Row", firstRow);
         assertEquals("Incorrect column count. Expected 7. Found "
            + firstRow.length, 7, firstRow.length);

         String[] columnHeaders = parser1.getColumnNames();
         assertNotNull("Null column headers", columnHeaders);
         assertEquals("Incorrect column count. Expected 9. Found "
            + columnHeaders.length, firstRow.length,
columnHeaders.length);

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

   }
}
```

The next section uses this parser with real data returned from the SOAP interface to provide suitable data to be displayed in tables in plug-in views.

6.4.2 Exposing the plug-in's functions

Now that we have a plug-in that can retrieve OMEGAMON data, we need to make this functionality available to other plug-ins. In production code, we create a package containing just the functions that we want to expose, but here we export the package containing all of our interesting code, the omegamon_soap package.

Usually when you want to export a package in a plug-in to be used by other plug-ins, you open the `plugin.xml` file in the plug-in development perspective and add the package to the exported packages section on the Runtime tab. The options that we selected when we created this plug-in using the wizard meant that we did not get a `plugin.xml` file created for us, so we have to use a slightly different approach. In the plug-ins view, double-click the OMEGAMON_SOAP plug-in to bring up the OMEGAMON_SOAP plug-in's properties. Click the Runtime tab and click **Add** in the exported packages section. This brings up the exported packages dialog panel, shown in Figure 6-8.

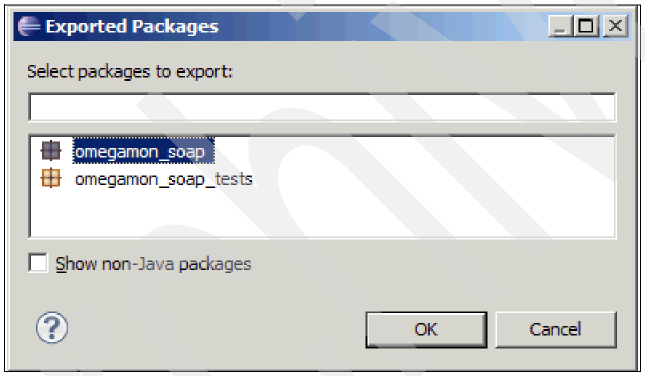
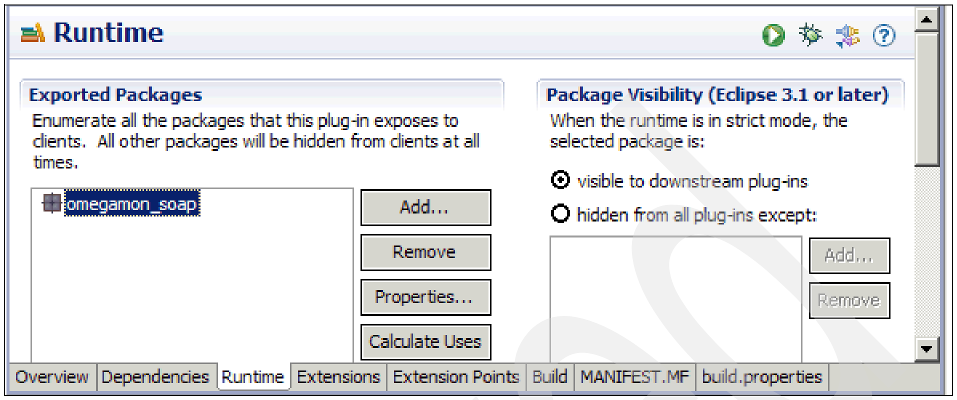


Figure 6-8 Exported Packages dialog menu

Select the omegamon_soap package and click **OK**. The package is then added to the list of exported packages. See Figure 6-9.


Runtime

Exported Packages

Enumerate all the packages that this plug-in exposes to clients. All other packages will be hidden from clients at all times.

omegamon_soap

Package Visibility (Eclipse 3.1 or later)

When the runtime is in strict mode, the selected package is:

[✓ visible to downstream plug-ins]

hidden from all plug-ins except:

Figure 6-9 Exported packages

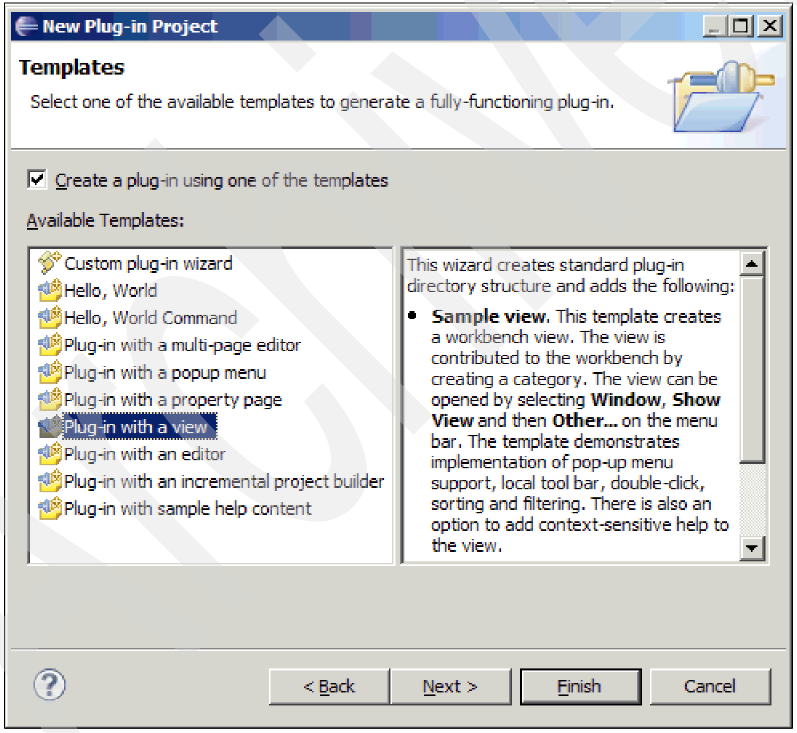
We have made all the code in the omegamon_soap package available to other plug-ins (subject to the usual Java code visibility restrictions). Now we can write code in other plug-ins that calls the public methods in these classes.

6.5 Displaying the OMEGAMON Managed System List

Now that we have a plug-in that retrieves data, we need to present that data in an Eclipse view. Our first view displays the Managed System List.

6.5.1 Creating the Managed System List plug-in

We need a new plug-in to display the data, so we start with the new plug-in wizard (Navigate to **File** → **New** → **project** and select **Plug-in project** in the New Project dialog box). Name the plug-in `ManagedSystemList` and accept the defaults on the first two panels. Ensure that the **This plug-in will make contributions to the UI** check box is selected. On the third panel, create the plug-in using a template, and select **plug-in with a view**, as shown in Figure 6-10.

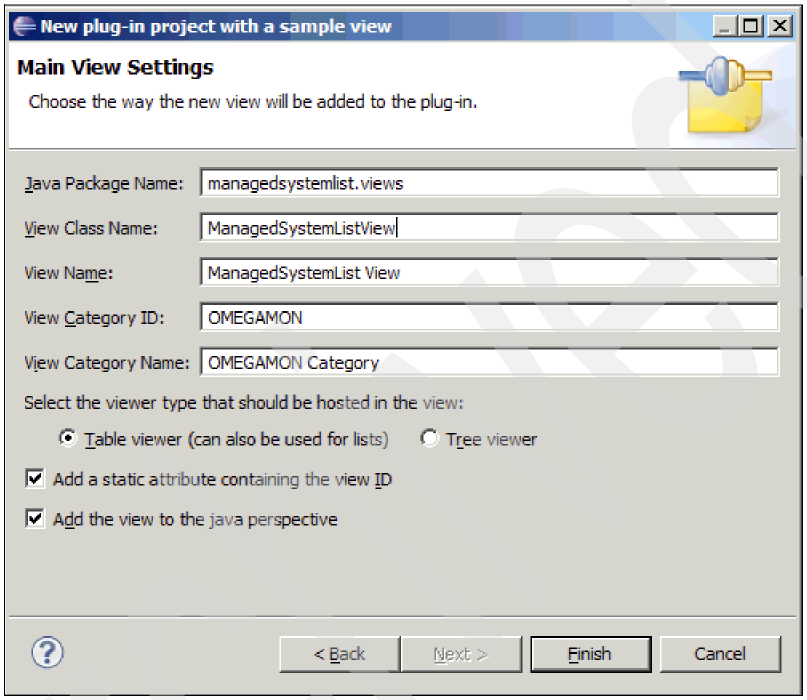

New Plug-in Project
Templates
Select one of the available templates to generate a fully-functioning plug-in.
Create a plug-in using one of the templates
Available Templates:
Custom plug-in wizard
Hello, World
Hello, World Command
Plug-in with a multi-page editor
Plug-in with a popup menu
Plug-in with a property page
Plug-in with a view
Plug-in with an editor
Plug-in with an incremental project builder
Plug-in with sample help content
< Back
Next >
Finish
Cancel

This wizard creates standard plug-in directory structure and adds the following:

- **Sample view.** This template creates a workbench view. The view is contributed to the workbench by creating a category. The view can be opened by selecting **Window**, **Show View** and then **Other...** on the menu bar. The template demonstrates implementation of pop-up menu support, local tool bar, double-click, sorting and filtering. There is also an option to add context-sensitive help to the view.

Figure 6-10 Select plug-in with a view

We are creating other visual plug-ins, so it is a good idea to group them so that they can be found together when a user selects the menu option **Window** → **Show View**. We create our own category for these views, called OMEGAMON. We achieve this on the next panel, by overriding several of the default names. We also select a table viewer as the type of viewer that is added to the view, as shown in Figure 6-11.


New plug-in project with a sample view
Main View Settings
Choose the way the new view will be added to the plug-in.
< Back
Next >
Finish
Cancel

Java Package Name: managedsystemlist.views
View Class Name: ManagedSystemListView
View Name: ManagedSystemList View
View Category ID: OMEGAMON
View Category Name: OMEGAMON Category

Select the viewer type that should be hosted in the view:
[✓ Table viewer (can also be used for lists)] Tree viewer

[✓] Add a static attribute containing the view ID
[✓] Add the view to the java perspective

Figure 6-11 Managed System List Main view settings dialog

Click **Finish** to complete the creation of the sample plug-in that we use as the basis for our view.The wizard has created a plug-in project for us with two packages, each containing one class.

We need to modify the ManagedSystemListView class, but before we start modifying it, we can run it in its current formIf. We invoke run (From the Run menu or the Run icon, select **Run Eclipse Application**. Then launch another Eclipse instance. In this instance, navigate to **Window** → **Show View** → **Other**. This brings up the "Show View" dialog box, shown in Figure 6-12.

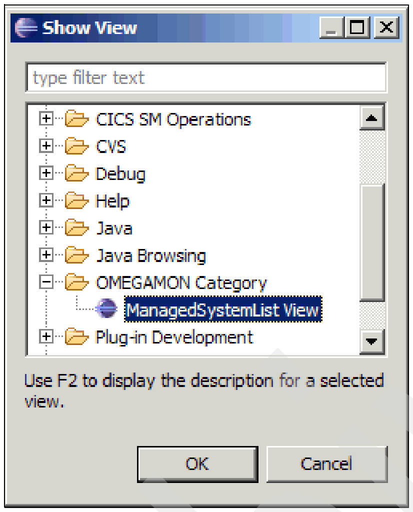


Figure 6-12 Show view (ManagedSystemList) dialog

The views listed in your dialog box might vary a little, depending on what plug-ins you have been writing or have imported into your Eclipse environment. Select ManagedSystemList View and click **OK**. This adds a new view to your Eclipse workspace that looks like Figure 6-13.

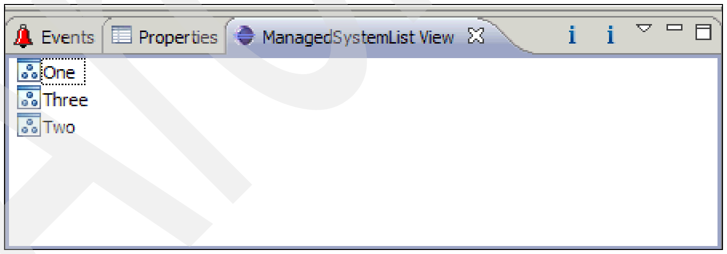


Figure 6-13 Sample ManagedSystemList view before code changes

We can see that the plug-in view generated by the wizard already works and displays sample data. We just need to change the code to display our Managed System List as returned by the SOAP interface.

A brief inspection of the generated code is useful at this time. There are several key components in the code that we are modifying so they are worth mentioning at this time:

- This view uses a TableViewer instance to display its data.
- The data that is displayed by the table is provided by the ViewContentProvider inner class.
- The presentation of the data in the table is handled by the ViewLabelProvider inner class.
- Sorting of the data in the class is implemented in the NameSorter inner class (which currently has an empty implementation).
- The `createPartControl(…)` method creates all the required objects and connects them to each-other.

The generated code also provides dummy implementations of context menus and toolbar icons and dummy actions to be invoked from the toolbar and menu options.

6.5.2 Displaying the Managed System List data in the plug-in

Now we have a sample plug-in that displays a table of data, we can start modifying this code to display our managed system list.

Because we are going to use the OEMGAMON_SOAP plug-in to provide our data, we need to specify a dependency on this plug-in. To do this, from the ManagedSystemList Overview panel, select the Dependencies tab at the bottom of the panel.

Note: Double-click `plugin.xml` in the center of the window if you cannot see this panel

On the Dependencies tab, add OMEGAMON_SOAP to the list of required plug-ins, and save. This ensures that we can access any exported code from that plug-in.

We next update the ViewContentProvider to display our data rather than the sample data. We add a tableData private instance field to hold the data to be displayed, a setter method for this field, and we update the `getElements()` method to return this field rather than the sample data. The updated version of the ViewContentProvider is shown in Example 6-8 on page 186.

Example 6-8 Updated ViewConentProvider

```
class ViewContentProvider implements IStructuredContentProvider {
        // the data to be displayed in the table
        private String[][] tableData = new String[0][0];

        public void setTableData(String[][] tableData) {
                this.tableData = tableData;
        }

        public Object[] getElements(Object parent) {
                return tableData;
        }

        public void inputChanged(Viewer v, Object oldInput, Object
newInput) {
        }
        public void dispose() {
        }
}
```

We need a new instance field in the ManagedSystemListView to hold a reference to the SoapInterface object that provides our data, and instance fields that contains the connection details so add the following lines after the existing field declarations in the managedSystemListView class.:

```
// the source of our managed system list data
private SoapInterface soapInterface;
// connection details for the SOAP interface
private String hostname = "winmvs2c.hursley.ibm.com";
private int port = 47587;
private String userid = "sysadmin";
private String password = "";
```

You have to change the host name, port, user ID, and password to match those for your system. These values are those used in 6.3.1, “Connecting to the TEMS SOAP interface with the Web client” on page 164.

At the end of the ManagedSystemListView constructor, add a call to a new `requestData()` method. This method requests the ManagedSystemList and update the table with the returned data.

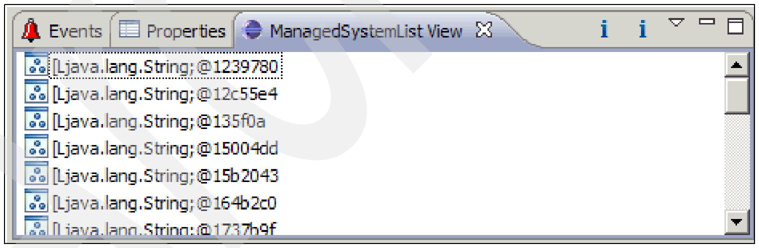
The implementation of the `requestData()` method is shown in Example 6-9 on page 187.

Example 6-9 requestData() method

```
private void requestData() {
        if (soapInterface == null) {
                soapInterface = new SoapInterface(hostname, port, 
userid, password);
        }
        SoapXMLParser parser = 
soapInterface.getParsedData(SoapInterface.MANAGED_SYSTEM_LIST_REQUEST);
        if (parser != null) {
                String[][] dataTable = parser.getDataTable();
                ((ViewContentProvider) 
viewer.getContentProvider()).setTableData(dataTable);
   viewer.refresh();
        }
        else  {
                showMessage("No data returned - Check host & port");
        }
}
```

This method creates an instance of the SoapInterface class if it has not already been created, and requests data from this instance. If data is returned, it is passed to the ViewContentProvider to be displayed in the table. If no data is returned, this is reported in a message.

If we run the code with these changes, we see a separate but not useful version of the managed system list, as displayed in Figure 6-14.



```
[Ljava.lang.String;@1239780
[Ljava.lang.String;@12c55e4
[Ljava.lang.String;@135f0a
[Ljava.lang.String;@15004dd
[Ljava.lang.String;@15b2043
[Ljava.lang.String;@164b2c0
[Ljava.lang.String;@1737b9f
```

Figure 6-14 Managed System List without columns

Each row in this panel is a reference to an entry in the managed system list table, but we have not yet added the code to extract each row's data and display it in columns. We now add that code.

We need to change the presentation of the table so that the column headers are displayed and the lines that mark the cells in the table are visible. Do this in `createPartControl(…)` method. After the `viewer.setInput(getViewSite());` method, add the information in Example 6-10.

Example 6-10 Changing the table presentation

```
Table table = viewer.getTable();
table.setHeaderVisible(true);
table.setLinesVisible(true);
```

We need to update the ViewLabelProvider (which handles the presentation of the data in the table) to display each row of data as multiple columns, rather than the single reference that we saw previously. This is achieved by replacing the existing `getColumnText(…)` method with the information in Example 6-11.

Example 6-11 Updating the ViewLabelProvider

```
public String getColumnText(Object obj, int index) {
        if (obj != null) {
                String[] row = (String[]) obj;
                return row[index];
        }
        else {
                return null;
        }
}
```

This method simply returns the appropriate cell data from the passed row according to the passed column number. Although we are looking at the ViewLabelProvider, we can stop icons being displayed in the cells by changing the `getColumnImage(…)` method to return null.

Finally, we need define the new columns in the table, and specify the text for the column headers. We define a new method in the ManagedSystemListView class, which creates these column definitions as in Example 6-12.

Example 6-12 Defining new columns

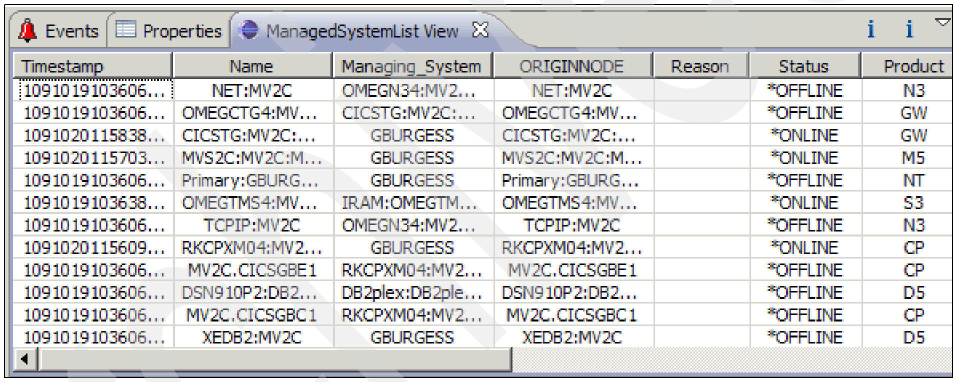
```
private void defineTableColumns(Table table, SoapXMLParser parser) {
        String[] columnHeaders = parser.getColumnNames();
        for (int i = 0; i < columnHeaders.length;i++) {
                TableColumn column = new TableColumn(table, SWT.CENTER);
                column.setText(columnHeaders[i]);
                column.setWidth(100);
        }
}
```

This method can only be called when we have data returned from the SOAP interface, as our columns and their names are specified in the returned data. We invoke this method in the `requestData()` method. Add the information in Example 6-13 before the `viewer.refresh()` method.

Example 6-13 Adding before the viewer.refresh() method

```
// if we have not previously defined the columns that make up the table
do it now
Table table = viewer.getTable();
if (table.getColumnCount() == 0) {
        defineTableColumns(table, parser);
```

With these changes in place, we can run the code again. We see a table with multiple columns and with headers on each of the columns. It resembles Figure 6-15.



Timestamp	Name	Managing_System	ORIGINNODE	Reason	Status	Product
1091019103606...	NET:MV2C	OMEGN34:MV2...	NET:MV2C		*OFFLINE	N3
1091019103606...	OMEGCTG4:MV...	CICSTG:MV2C:...	OMEGCTG4:MV...		*OFFLINE	GW
1091020115838...	CICSTG:MV2C:...	GBURGESS	CICSTG:MV2C:...		*ONLINE	GW
1091020115703...	MVS2C:MV2C:M...	GBURGESS	MVS2C:MV2C:M...		*ONLINE	M5
1091019103606...	Primary:GBURG...	GBURGESS	Primary:GBURG...		*OFFLINE	NT
1091019103638...	OMEGTMS4:MV...	IRAM:OMEGTM...	OMEGTMS4:MV...		*ONLINE	S3
1091019103606...	TCPIP:MV2C	OMEGN34:MV2...	TCPIP:MV2C		*OFFLINE	N3
1091020115609...	RKCPXM04:MV2...	GBURGESS	RKCPXM04:MV2...		*ONLINE	CP
1091019103606...	MV2C.CICSGBE1	RKCPXM04:MV2...	MV2C.CICSGBE1		*OFFLINE	CP
1091019103606...	DSN910P2:DB2...	DB2plex:DB2ple...	DSN910P2:DB2...		*OFFLINE	D5
1091019103606...	MV2C.CICSGBC1	RKCPXM04:MV2...	MV2C.CICSGBC1		*OFFLINE	CP
1091019103606...	XEDB2:MV2C	GBURGESS	XEDB2:MV2C		*OFFLINE	D5

Figure 6-15 Managed System List view with columns

In this view, we can see the names of all the managed systems in our OMEGAMON XE monitoring environment, and we also see any other IBM Tivoli Monitoring systems visible to our TEMS. One column of particular interest is the Status column, which tells us whether the systems are online or offline.

Now that we have a list of the systems being monitored, we have several options for what we can do next with the plug-in. These include:

- Moving the requests for data to a more suitable thread
- Refreshing the data in the table either on demand, or at regular intervals
- Sorting the data in the table
- Filtering the data displayed in the table by specific values.

We discuss sorting of the data in the table and refreshing the data at a regular interval in 6.5.3, “Sorting the Managed System List” on page 190, and leave the filtering item and the refreshing of the data when prompted by the user as an exercise for the user. A useful side effect of the code that are added to fetch the data at regular intervals is that the requests for data is moved to a more suitable thread.

6.5.3 Sorting the Managed System List

The column to be sorted can be selected by clicking the column header, so we need to add code to detect the user clicking the column header. We start by defining a Listener class as in Example 6-14. Add this code as an inner class in the ManagedSystemListView class. This simple code detects column header selection events. When a selection event is detected, it sets the selected column and drives a re-sort of the data in the table.

Example 6-14 ColumnHeaderSelectionListener class

```
// the selection listener for the column headers
class ColumnHeaderSelectionListener implements Listener {
        private final ManagedSystemListView owner;

        ColumnHeaderSelectionListener(ManagedSystemListView owner) {
                this.owner = owner;
        }

        public void handleEvent(Event e) {
                TableColumn currentColumn = (TableColumn)e.widget;
        Table table = currentColumn.getParent();

        int colIndex = -1;
        for (int i = 0;i<table.getColumnCount();i++) {
            if (table.getColumn(i) == currentColumn) {
                colIndex = i;
            }
        }
        owner.setSortColumn(colIndex);
        table.setRedraw(false);
        owner.sort();
        table.setRedraw(true);
        viewer.refresh();
        }
}
```

Adding the ColumnHeaderSelectionListener class introduces a couple of errors, as we need to add further code to the class. To remove the errors, add the missing methods and fields to the ManagedSystemListView class. This code is shown in Example 6-15.

Example 6-15 'Sort' fields and methods for ManagedSystemListView class

```
// support for column sorting
private int sortColumn = 0;
private boolean sortUp = false;

void setSortColumn(int column) {
        if (sortColumn == column) {
                sortUp = !sortUp;
        }
        this.sortColumn = column;
}

void sort() {
        viewer.getSorter().sort(viewer, ((IStructuredContentProvider)
viewer.getContentProvider()).getElements(null));
}
```

We also need to associate the selection listener with the table columns, so we modify the `defineTableColumns(…)` method to look like that shown in Example 6-16. The only change is to add the line in bold that registers the listener with each of the columns as they are created.

Example 6-16 Updated defineTableColumns(...) method

```
private void defineTableColumns(Table table, SoapXMLParser parser) {
        String[] columnHeaders = parser.getColumnNames();
        for (int i = 0; i < columnHeaders.length;i++) {
                TableColumn column = new TableColumn(table,
SWT.CENTER);
                column.setText(columnHeaders[i]);
                column.setWidth(100);
                column.addListener(SWT.Selection, new
ColumnHeaderSelectionListener(this));
        }
}
```

Finally, we need to add code to the ColumnSorter class that was created for us when the plug-in was created by the wizard. Replace the (currently empty) existing implementation of the NameSorter inner class with the version shown in Example 6-17. This class provides an implementation of a ViewerSorter class that is invoked by the TableViewer's underlying code when its `sort()` method is invoked.

Example 6-17 Updated NameSorter class

```
class NameSorter extends ViewerSorter {
        private final ManagedSystemListView owner;

        public NameSorter(ManagedSystemListView owner) {
                this.owner = owner;
        }

        public int compare(Viewer viewer, Object e1, Object e2) {
                // find the two values to be compared (depends on the
current sort column)
                String value1 = ((String[]) e1)[sortColumn];
                String value2 = ((String[]) e2)[sortColumn];

                // guard against null values
                if (value1 == null || value2 == null) {
                        return 0;
                }
                if (owner.sortUp) {
                        return value1.compareTo(value2);
                }
                else {
                        return value2.compareTo(value1);
                }
        }
}
```

The final change is in the createPartControl(…) class. Replace `viewer.setSorter(new NameSorter())` with `viewer.setSorter(new NameSorter(this));`

We now have a plug-in that displays the managed system list and can sort the displayed data by the values in any of the columns. This can be useful to group systems by their product code, or to group all the offline systems at the top of the view.

6.5.4 Requesting the Managed System List repeatedly

The plug-in currently requests the managed system list once when it is created. We have two concerns with this:

- We are doing network I/O on the Display thread. The GUI does not respond while the I/O operation is in process.
- The list is not updated and so does not reflect changes to system status.

We can address both of these problems by driving the request for data regularly on another thread. The changes are simple.

First, we update the `createPartControl(…)` method. Remove the last line ‘requestData();’ and replace it with the code shown in Example 6-18.

Example 6-18 requestThread

```
Thread requestThread = new Thread() {
        public void run() {
                while (true) {
                        // request the situations
                        requestData();

                        // and sleep for a bit before fetching the 
situations again
                        try {
                                Thread.sleep(60000);
                        }
                        catch (InterruptedException e) {
                                // woken early - ignore
                        }
                }
        }
};
requestThread.start();
```

This code creates a thread that issues and processes the request and then waits for one minute (60,000 milliseconds) before repeating, and then starts the thread.

If we run the plug-in now, we see an org.eclipse.swt.SWTException, as we are trying to update the table (an SWT widget) on a thread other than the Display thread (the thread that we created previously). We need to change the `requestData()` method to make the updates on the Display thread. The updated version of the requestData() method is shown in Example 6-19 on page 194.

Example 6-19 Updated requestData() method

```
private void requestData() {
        if (soapInterface == null) {
                soapInterface = new SoapInterface(hostname, port,
userid, password);
        }
        final SoapXMLParser parser =
soapInterface.getParsedData(SoapInterface.MANAGED_SYSTEM_LIST_REQUEST);
        if (parser != null) {
                Display display = viewer.getTable().getDisplay();
                display.asyncExec(new Runnable() {
                        public void run() {
                                String[][] dataTable =
parser.getDataTable();

                                // update the table model with the
passed data
                                ((ViewContentProvider)
viewer.getContentProvider()).setTableData(dataTable);

                                // if we have not previously defined
the columns that make up the table, do it now
                                Table table = viewer.getTable();
                                if (table.getColumnCount() == 0) {
                                        defineTableColumns(table,
parser);
                                }
                                viewer.refresh();
                        }
                });
        }
        else  {
                showMessage("No data returned - Check host & port");
        }
}
```

The changes (all additions) are highlighted in bold. The reference to the SoapXMLParser has been flagged as final so it can be access by the Display thread. The code that updates the table is now invoked in an anonymous inner class that is invoked on the Display thread.

We can now run the plug-in knowing that the managed system list updates every minute.

6.6 Displaying OMEGAMON situations

Now that we have a plug-in that requests the managed system list from the TEMS SOAP interface and displays the list, it is easy to create another plug-in that requests and displays Tivoli monitoring situations. These situations are user-defined alerts that can be defined to trigger if certain criteria evaluate to true. These criteria are typically undesirable performance values (such as CPU usage too high, short or storage or transactions blocked waiting for resources) or necessary resources becoming unavailable (connection not open or file unavailable). Tivoli situations are defined through the TEP interface and can be viewed there, but we can also retrieve a list of situations from the SOAP interface.

First, we need to define the new request that is used to request the situations. Add the following instance variable to the SoapInterface class:

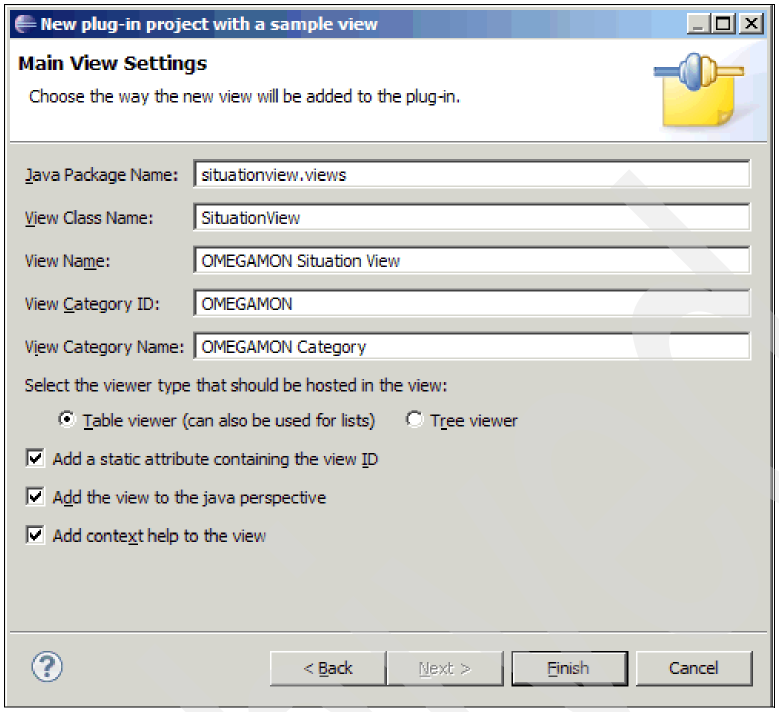
```
public static final String SITUATIONS_REQUEST =
      "<table>O4SRV.ISITSTSH</table><sql>SELECT SITNAME, NODE, ATOMIZE,
GBLTMSTMP, LCLTMSTMP, ORIGINNODE, PATHNAME, TYPE, DELTASTAT FROM
O4SRV.ISITSTSH WHERE DELTASTAT = 'Y' </sql>";
```

This SOAP request uses native SQL to request the data from the TEMS. As you can see, the SOAP requests can be constructed from tags that describe the table from which the data is extracted, what data is required, and any filters that are used to restrict the returned data. In this case, we use SQL to specify this information.

The request syntax is consistent with SQL statements issued against database engines and so requires little explanation. The Select statement specifies that we want data returned and the columns that are required, the From clause specifies the table that is searched for the data, and the Where clause specifies the constraints that is applied when choosing which rows from the table are returned.

Creating a plug-in that displays current situations requires little more than copying the plug-in we created to display the managed system list and changing the request that is issued to the SOAP interface. Obviously this approach creates a significant amount of duplicate code in the two plug-ins and it is a good idea to factor out much of this code into common code that is shared by the plug-ins, but we do not discuss that here.

To create the new view, create a new plug-in project, called “SituationView” with the new plug-in wizard, following the steps described in 6.5.1, “Creating the Managed System List plug-in” on page 182. This time, select the same options in the wizard, but with different names on the final panel. See Figure 6-16 on page 196.


New plug-in project with a sample view

Main View Settings

Choose the way the new view will be added to the plug-in.

Java Package Name:	situationview.views
View Class Name:	SituationView
View Name:	OMEGAMON Situation View
View Category ID:	OMEGAMON
View Category Name:	OMEGAMON Category

Select the viewer type that should be hosted in the view:

[✓ Table viewer (can also be used for lists)] Tree viewer

- [✓] Add a static attribute containing the view ID
- [✓] Add the view to the java perspective
- [✓] Add context help to the view

Figure 6-16 Create Situation View - wizard last panel

We are going to use the OEMGAMON_SOAP plug-in to provide our data so we need to specify a dependency on this plug-in. We select the Dependencies tab, this time in the `plugin.xml` file in the SituationsView project that we have just created. On the Dependencies tab, add OMEGAMON_SOAP to the list of required plug-ins (and save) to allow us to access the code exported from that plug-in.

Next, replace the generated code in the SituationView class with the code from the class ManagedSystemListView, but with a few modifications to fetch OMEGAMON Situation data rather than the ManagedSystemList.

The steps to make these changes are as follows:

1. Delete all the code after the package statement in class SituationView, and copy the entire contents of ManagedSystemListView (other than the package statement in its place.
2. Replace all instances of 'ManagedSystemListView' with 'SituationView'.
3. In the requestData() method, change the following line:

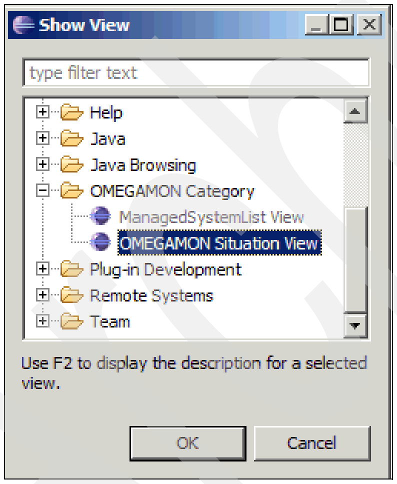
```
final SoapXMLParser parser =
soapInterface.getParsedData(SoapInterface.MANAGED_SYSTEM_LIST_REQUEST);
```

Replace the preceding code with the following line

```
final SoapXMLParser parser =
soapInterface.getParsedData(SoapInterface.SITUATIONS_REQUEST);
```

Save.

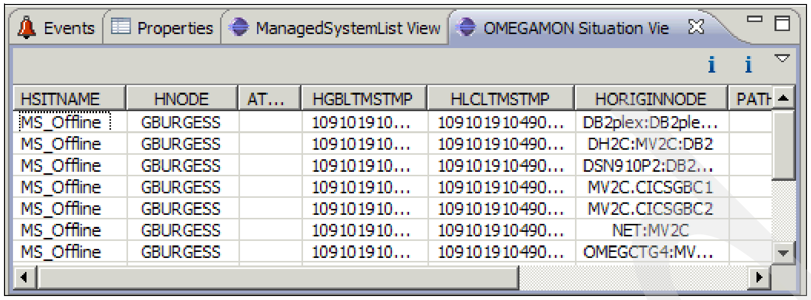
Run the program and we now have a new view that we can select from the **Window** → **Show View** → **Other** menu option as shown in: Figure 6-17.



Use F2 to display the description for a selected view.

Figure 6-17 Show view - Situation view

Selecting this view displays the current situations (if any) in the monitored systems, as shown in Figure 6-18.



HSITNAME	HNODE	AT...	HGBLTMSTMP	HLCLTMSTMP	HORIGINNODE	PATH
MS_Offline	GBURGESS		109101910...	109101910490...	DB2plex:DB2ple...	
MS_Offline	GBURGESS		109101910...	109101910490...	DH2C:MV2C:DB2	
MS_Offline	GBURGESS		109101910...	109101910490...	DSN910P2:DB2...	
MS_Offline	GBURGESS		109101910...	109101910490...	MV2C.CICSGBC1	
MS_Offline	GBURGESS		109101910...	109101910490...	MV2C.CICSGBC2	
MS_Offline	GBURGESS		109101910...	109101910490...	NET:MV2C	
MS_Offline	GBURGESS		109101910...	109101910490...	OMEGCTG4:MV...	

Figure 6-18 OMEGAMON Situations View

6.7 Displaying OMEGAMON data for specific CICS regions

This chapter has focused on displaying data that applies across the Tivoli monitoring systems. We now look at requesting data for specific CICS regions. There are a large number of reports supplied as part of the OMEGAMON for CICS product, with more being added with new releases to reflect new features in CICS. Most of these reports can be driven through the SOAP interface using a similar request to that used previously to request the managed system list or situations.

6.7.1 Constructing SOAP queries for CICS data

Before we code our next plug-in, now is a good time to look at the SOAP request format a little more closely. The general form of the SOAP request we are going to use to request data about specific CICS regions is as follows:

```
<CT_Get>
<userid>sysadmin</userid><password></password>
<object>report_name</object>
<target>region_originnode</target>
<afilter>column_name;operator;value</afilter>
</CT_Get>
```

The first and last lines mark the start and end of the request. Again, we are using a CT_GET request to request data.

The second line is also familiar. It contains our user ID and password.

The ‘object’ tag (third line) specifies which of the many available reports we are interested in. We discuss how you can find a list of these reports in the text that follows.

The ‘target’ tag (fourth line) contains the ORIGINNODE of the CICS region we are interested in. This value we can get from the managed system list.

The ‘afilter’ tag (fifth line) contains the constraints that are used when the data is requested. There can be more than one of these tags, each specifying additional constraints. These are analogous to the WHERE clause in an SQL SELECT statement. The column_name field is the name of the column to which the filter applies. The value field is the value that is compared with the column value when the server chooses the data to be returned. The operator field is the operator that is used in the comparison. Valid operator field values are the usual comparator values:

- EQ (equals),
- NE (not-equal),
- GT (greater than)
- LT (less than)
- GE (greater than or equal to)
- LE (less than or equal to)

Example 6-20 is a sample request for the CICSplex Region Overview report for CICS region ZT01.CICSDM25.

Example 6-20 Sample request

```
<CT_Get>
<userid>sysadmin</userid><password></password>
<object>CICSplex_Region_Overview</object>
<target>ZT01.CICSDM25</target>
<afilter>Origin_Node;EQ;ZT01.CICSDM25</afilter>
</CT_Get>
```

It can be useful to experiment with values in the SOAP request fields either in the plug-in code that we create in the section that follows, or using the supplied Web client as described in 6.3.1, “Connecting to the TEMS SOAP interface with the Web client” on page 164.

Finding values to use in the XML requests

Because there is a large number of reports that can be driven to request data from OMEGAMON CICS, it can be a problem finding out what reports are available and what their names are. There are several approaches to this, but one approach is as follows:

1. View the TEP client and choose a query that contains the data that you are interested in.
2. Right-click the report of interest, and select **Properties** in the menu.
3. In the Properties dialog box, click the **Click here to assign query** button.
4. Read the query name from the list on the right of the query editor dialog box, as shown in Figure 6-19. In this list, the query has spaces between the words that make up the name, but we require underscore characters instead of spaces.

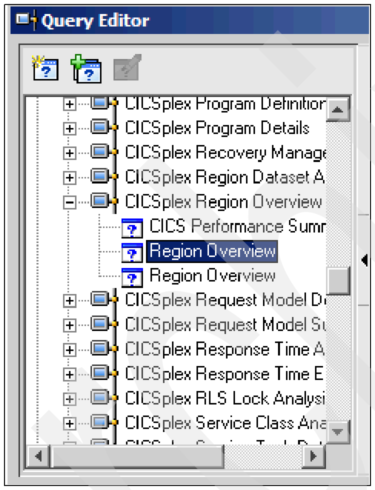


Figure 6-19 TEP Query View

5. Open the DOCKCP file in your TEPS install directory (typically on windows this is in `c:\ibm\itm\cnps`.
6. Find the query name in the DOCKCP file. It should be on a line that starts *OBJECT:
7. Scroll down from here to read the column names. These are on lines that start with the tag *ATTR:

   When you reach a block of text that includes another *OBJECT: tag, you have reached the definitions for another report.

This gives you the appropriate values to use in the object and afilter tags of the SOAP request.

Alternatively, the report names can be derived from the `kcp.atr` and `kcp.cat` files. Although these files are more difficult to read, they can be scanned easily with a simple program to list all the available reports. This alternative approach is described, along with an excellent overview of the SOAP interface, in a technote that can be found at the following Web page:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD104290

6.7.2 Displaying the OMEGAMON CICS Region Overview report

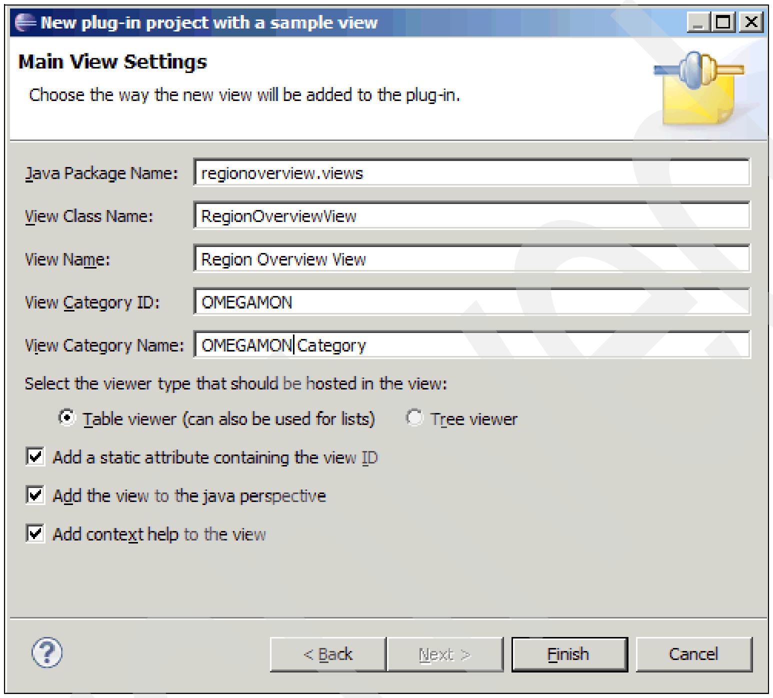
Requesting and displaying the CICS region overview report from OMEGAMON requires a similar approach to display the managed system list and the situations, but now we must construct the request with the name of the CICS system in which we are interested. This requires additional input, as we cannot specify the region when we write the code. We are going to allow the user to specify the CICS region in two ways:

- By selecting a monitored CICS region in the managed system list
- By selecting a CICS region in the CICS Explorer navigation tree.

Before we implement either of these selection methods, we need to write the plug-in that displays the region overview information.

Tip: Although this example demonstrates requesting the CICS region overview, it is a simple matter to change the code to request a different report, or to request one of several reports depending on user input or other context.

We create the Region Overview in the same way that we created the Situation View plug-in; by copying the Managed System List plug-in code and making changes as required. Again, we create a plug-in project using the wizard that has a Table view. We call the plug-in project RegionOverview and specify the values on the wizard's final panel, as shown in Figure 6-20.



Main View Settings

Choose the way the new view will be added to the plug-in.

Java Package Name:	regionoverview.views
View Class Name:	RegionOverviewView
View Name:	Region Overview View
View Category ID:	OMEGAMON
View Category Name:	OMEGAMON Category

Select the viewer type that should be hosted in the view:

[● Table viewer (can also be used for lists)] ○ Tree viewer

- [✓] Add a static attribute containing the view ID
- [✓] Add the view to the java perspective
- [✓] Add context help to the view

Figure 6-20 Create Situation View - wizard last panel

We add OMEGAMON_SOAP to the list of required plug-ins to allow us to access the code exported from that plug-in.

As with the situationView, delete the entire contents of the RegionOverviewView class, with the exception of the package statement at the top. Replace this with the ManagedSystemListView class, and replace all instances of ManagedSystemListView with RegionOverviewView.

Now we can address the code that requests the region overview for a specific CICS region. As we are constructing the SOAP request at runtime, we need an instance field to hold the identifier of the CICS region that we are going to request data for. The code in Example 6-21 on page 203 contains the field with its getters and setters. Add this to the RegionOverviewView class.

Example 6-21 originNode field

```
private String originNode;
public String getOriginNode() {
        return originNode;
}
public void setOriginNode(String originNode) {
        this.originNode = originNode;
}
```

Next, replace the hard-coded SOAP request in the `requestData()` method with the code to construct the request based on the current value of the 'originNode' field. Replace the following line in the `requestData()` method with the code in Example 6-22:

```
final SoapXMLParser parser =
soapInterface.getParsedData(SoapInterface.MANAGED_SYSTEM_LIST_REQUEST);
```

Example 6-22 Construct cics region overview request

```
if (originNode == null) {
        return;
}
String request = "<object>CICSplex_Region_Overview</object>" +
      "<target>" + originNode + "</target>" +
      "<afilter>Origin_Node;EQ;" + originNode + "</afilter>";
final SoapXMLParser parser = soapInterface.getParsedData(request);
```

This code checks to see if the originNode value has been set and returns if it hs not, because we cannot issue a request without it. It then constructs the request, inserting the originNode value as required and then issues the request as before.

Next we need a way to trigger the issuing of the request. We can achieve this by waking the thread that issues the request in the inner class in the `createPartControl(…)` method. We have to gain access to this thread, so we change the requestThread local variable in the `createPartControl(…)` method into an instance field in the RegionOverviewView class:

1. Add an instance variable 'private Thread requestThread; to the RegionOverviewView class.
2. Change the line `Thread requestThread = new Thread() {` in the `createPartControl(…)` method, to `requestThread = new Thread() {`.

3. Replace the `setOriginNode(…)` method with the code shown in Example 6-23, which wakes the request thread when we receive new value for originNode.

Example 6-23 Updated setOriginNode(...) method

```
public void setOriginNode(String originNode) {
        this.originNode = originNode;
        requestThread.interrupt();
}
```

We now need code to drive this view. We make this view display the CICS region overview report for the selected CICS region in the managed system list, and then we add code to react to selection events in the CICS Explorer.

We need to tell the Managed System View to make its selection events visible to other plug-ins, so add the following lines to the end of the `createPartControl(…)` method of the ManagedSystemListView class:

```
// ensure that this table issues selection events that can be detected
by other plug-ins
getSite().setSelectionProvider(viewer);
```

In its current form, the managed system list table only detects selection events in the first column. Although this is sufficient, it is not ideal, so we need to change it to detect selections anywhere in the row. We can do this by changing the first line of the createPartControl method in the ManagedSystemList class. We need to add the SWT.FULL_SELECTION flag in the call to the TableViewer constructor, so that the first line becomes:

```
viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL |
SWT.V_SCROLL | SWT.FULL_SELECTION);
```

Now that the Managed System List is issuing selection events, we can add code to the Region Overview view to listen for them. This code is a new method that is shown in Example 6-24.

Example 6-24 init(..) method

```
public void init(IViewSite site) throws PartInitException {
        super.init(site);
        // register ourselves as interested in selection events in
other views
        site.getPage().addSelectionListener(new ISelectionListener(){
                public void selectionChanged(IWorkbenchPart arg0,
ISelection selection) {
                        if(!selection.isEmpty()) {
                                Object firstElement =
                 ((StructuredSelection)selection).getFirstElement();
                              if (firstElement instanceof String[]) {
                                   String[] selectedRow =
(String[]) firstElement;
                                   if (selectedRow.length < 7) {
                                        // not an MSL row -
insufficient columns
                                        return;
                                   }
                                   String originNode =
selectedRow[1];
                                   String productCode =
selectedRow[6];
                                  if ("CP".equals(productCode) ==
false || originNode == null ||
                    originNode.indexOf('.') < 1) {
                                        // not a CICS region
                                        return;
                                   }
                                   // this is probably a CICS
region in the Managed System List
                                   setOriginNode(originNode);
                              }
                         }
                    }
          });
}
```

As this method overrides a method in the super class, it starts with a call to the super class's implementation. The code then creates an inner class that listens for selection events from plug-ins within Eclipse. As these selection events might come from many plug-ins other than the events we are looking for, the code has to check for and reject unexpected values. Most of the code is checking for unexpected values, and if one is found, the code returns without further processing.

If the selection event has a String[] as its data component, and if the string array is long enough to have come from the Managed System List view, the code attempts to read the product code and origin node values from the row. Only if the product code matches that for CICS (CP) and if the origin node is of the correct format does the code invoke the `setOriginNode(…)` method to drive a request for the region overview for the selected CICS region.

Tip: Because the Managed System List view uses a simple array of string arrays to contain its data, the selection listener for events from it requires detailed knowledge of the order of the columns in the table. If this ordering were to change, the code fails. A better approach is to use a dedicated class to encapsulate the row's values and provide suitable getter methods to return the required values from the row. This approach has not been followed here as it increases the complexity of this example, but it is not be advisable to use this simple approach in production code.

We can now run the application to try out our new plug-in. We need to display the new view. Navigate to **Window** → **Show View** → **Other**, as shown in Figure 6-21.

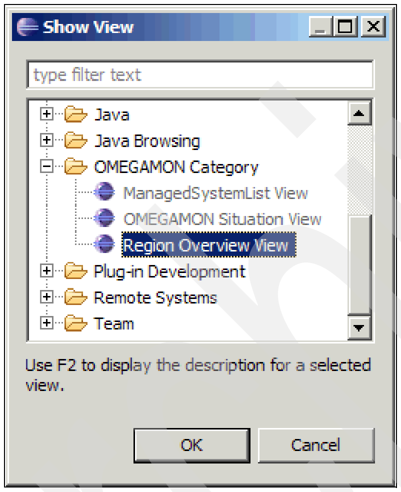
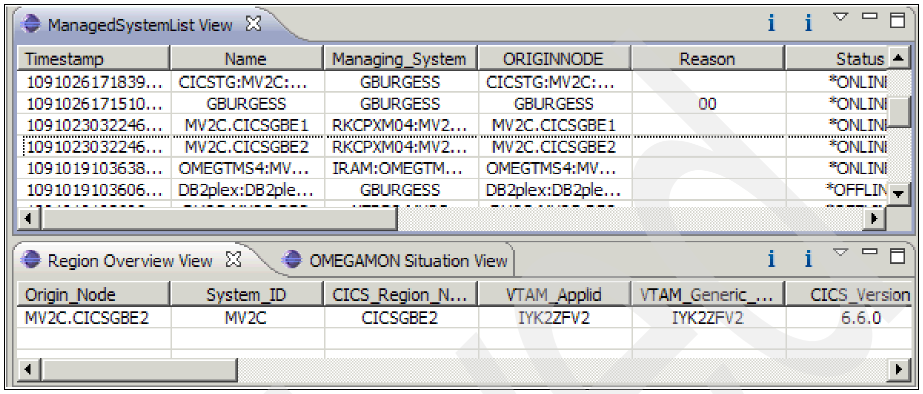


Figure 6-21 Show view: Region Overview view

Selecting this view displays an empty table, as we have not yet selected a row in the managed system list. However, if we select a CICS region in the managed system list, we see the region overview report for that CICS region, as shown in Figure 6-22.



Timestamp	Name	Managing_System	ORIGINNODE	Reason	Status
1091026171839...	CICSTG:MV2C:...	GBURGESS	CICSTG:MV2C:...		*ONLINI
1091026171510...	GBURGESS	GBURGESS	GBURGESS	00	*ONLINI
1091023032246...	MV2C.CICSGBE1	RKCPXM04:MV2...	MV2C.CICSGBE1		*ONLINI
1091023032246...	MV2C.CICSGBE2	RKCPXM04:MV2...	MV2C.CICSGBE2		*ONLINI
1091019103638...	OMEGTMS4:MV...	IRAM:OMEGTM...	OMEGTMS4:MV...		*ONLINI
1091019103606...	DB2plex:DB2ple...	GBURGESS	DB2plex:DB2ple...		*OFFLIN

Origin_Node	System_ID	CICS_Region_N...	VTAM_Applid	VTAM_Generic_...	CICS_Version
MV2C.CICSGBE2	MV2C	CICSGBE2	IYK2ZFV2	IYK2ZFV2	6.6.0

Figure 6-22 Region Overview view

6.8 Driving an OMEGAMON plug-in from CICS Explorer

The Region Overview view reacts to selection events in the managed system list. Although this can be useful, it is useful to make this plug-in also react to selection events in the CICS Explorer.

Note: For the code that follows to work, your CICS regions must be both monitored by OMEGAMON and accessible by the CICS Explorer, either through the CMCI interface or CPSM.

To make the Region Overview plug-in react to selections in the CICS Explorer, we need two extensions to the code we have so far:

- The Region Overview plug-in must detect (and understand) CICS Explorer selection events
- We need to translate between the way that the CICS Explorer identifies a CICS Region, and the way that OMEGAMON identifies a CICS Region.

We deal with these the translation problems first, and then move on to the selection events.

6.8.1 Translating between CICS Explorer and OMEGAMON identifiers

OMEGAMON identifies CICS regions by their ORIGINNODE. This is a concatenation of the z/OS System ID (SMFID) and CICS region name separated by a period character (.).

The CICS Explorer identifies the CICS region by the value in the NAME field of the EYUPARM in the CICS region's JCL. This is typically the CICS region’s APPLID, but it might not be.

Although we are discussing this conversion in the context of the region overview report, it can be used in other plug-ins, so it is useful to place the conversion routine in a common location that can be accessed easily by multiple plug-ins. The OMEGAMON_SOAP plug-in that we created earlier in this chapter is already accessed by several plug-ins, so that is a suitable place for this lookup code to be implemented.

When a CICS Explorer event is received by the Region Overview plug-in, the plug-in queries the CICS Explorer API to discover the CICS region’s jobname, and then it queries the managed system list to find the ORIGINNODE value of that CICS region. We can ease the process of looking up the ORIGINNODE value by extracting the CICS jobname from the ORIGINNODE value when the managed system list is retrieved from the host and maintaining a lookup table that maps CICS jobnames to their ORIGINNODE values. The best time to do this is when we process a request for the managed system list, and the ideal place to do it is in the OMEGAMON_SOAP plug-in.

We need to make several changes to support a lookup of ORIGINNODE values from a CICS jobname. All of these changes are applied to the SoapInterface The first of these changes is simply to add a static field to a Java Map that contains the following data:

```
private static Map<String, String> jobnameMap;
```

Next, add code to the `getParsedData()` method that invokes the creation (or update) of the map if a managed system list has been requested and returned. The updated method is shown in Example 6-25 on page 209.

Example 6-25 Updated getParsedData(…) method

```
public SoapXMLParser getParsedData(String request) {
        String xmlResult = getData(request);
        if (xmlResult != null) {
                try {
                        SoapXMLParser parser = new
SoapXMLParser(xmlResult);

                        // if we were asked for a managed system list,
create a map
                        // of cics jobnames to their ORIGINNODE values
                        if
(request.equals(MANAGED_SYSTEM_LIST_REQUEST)) {
                            createJobnameMap(parser);
                        }
                            return parser;
                }
                catch (Exception e) {
                        e.printStackTrace();
                        return null;
                }
        }
        else {
                return null;
        }
}
```

Finally, we need the createJobnameMap(…) method, which populates the map, and the getOriginNode(…) method, which returns an ORIGINNODE value for a passed jobname if found. These methods are shown in Example 6-26.

Example 6-26 New SoapInterface methods

```
public static String getOriginNode(String jobname) {
        if (jobnameMap == null) {
                return null;
        }
        else {
                return jobnameMap.get(jobname);
        }
}

private void createJobnameMap(SoapXMLParser parser) {
        // ensure we have an empty map to fill
        if (jobnameMap == null) {
                jobnameMap = new HashMap<String, String>();
        }
        else {
                jobnameMap.clear();
        }
        String[][] table = parser.getDataTable();
        if (table != null) {
                for (String[] row : table) {
                        if (row.length > 6) {
                                String originnode = row[3];
                                String productCode = row[6];
                                if ("CP".equals(productCode) &&
originnode != null) {
                                        int separatorIndex =
originnode.indexOf(".");
                                        String jobname = null;
                                        if (separatorIndex > 0 &&
                    separatorIndex < originnode.length()-1) {
                                                jobname =
originnode.substring(separatorIndex+1);
                                        }
                                        if (jobname != null) {
                                               jobnameMap.put(jobname,
originnode);
                                        }
                                }
                        }
                }
        }
}
```

Tip: Once again, we have code that requires detailed knowledge of the managed system list report. Again, this code would be better if we had a dedicated class that encapsulated the report data and provided suitable getter methods

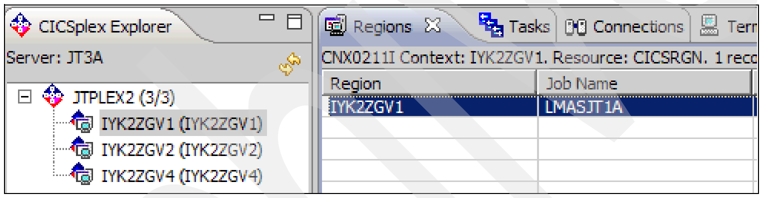
6.8.2 Detecting and understanding CICS Explorer selection events

Our first issue is finding out what events are fired when a selection is made in a CICS Explorer view. We already have most of the code we require to find out about these events. The SelectionListener for the OMEGAMON Region Overview report captures selection events published in Eclipse, including those from the CICS Explorer. We can see the class of the selection events by adding a line to the selection listener immediately after we have assigned the firstElement variable, such as:

```
System.out.println("Selection, class: " + firstElement.getClass() + "
toString " + firstElement);
```

We can now run the code, connect our CICS Explorer to a server, and select items in the CICS Explorer to view their classes in standard output.

The first of these selection events was caused by clicking a CICS region in the CICS Explorer topology view, the second by clicking a row in the CICS Explorer Regions view. Figure 6-23 shows these items selected in the CICS Explorer.



CNX0211I Context: IYK2ZGV1. Resource: CICSRGN. 1 reco

Region	Job Name
IYK2ZGV1	LMASJT1A

Figure 6-23 CICS Explorer CICS region selected

An example of the output for these selection events is shown in Example 6-27, where we have the output for the two events

Example 6-27 Selection events shown in standard output

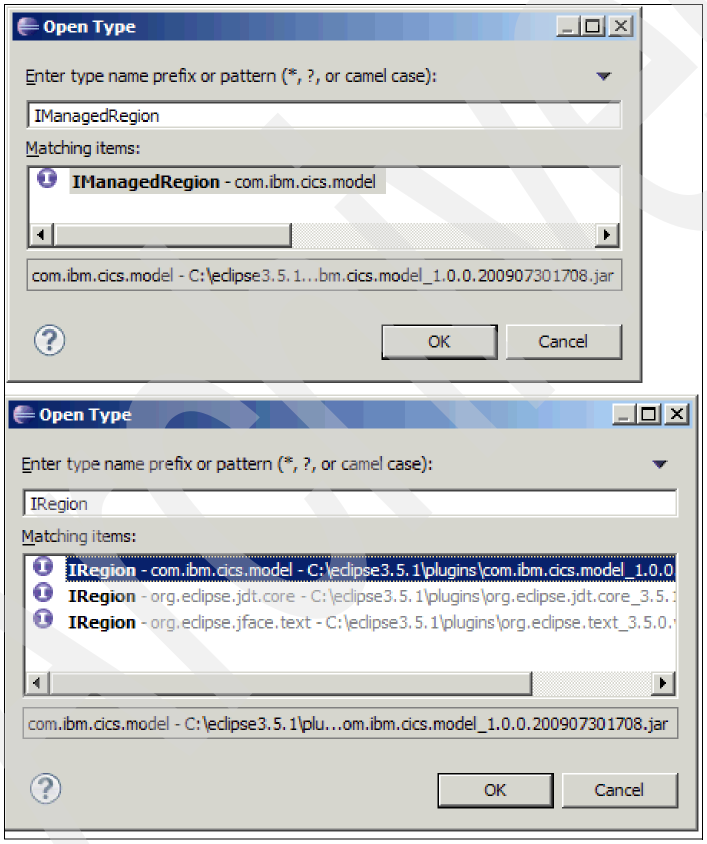
```
Selection, class: class com.ibm.cics.core.model.internal.ManagedRegion
toString: ManagedRegion[IYK2ZGV1]

Selection, class: class com.ibm.cics.core.model.internal.Region
toString: com.ibm.cics.core.model.internal.Region[IYK2ZGV1]
```

We can see from these two events that we have received instances of two separate classes:

- com.ibm.cics.core.model.internal.ManagedRegion
- com.ibm.cics.core.model.internal.Region

We cannot add either of these classes to the dependencies for our plug-ins, as they are not made available to us by the SDK plug-ins, but we can go looking for Interfaces that they implement. For many of the internal CICS Explorer classes, there is a publicly visible interface that has the same name as the class, but with a letter 'I' as a prefix. When we look for these interfaces (Navigate to **Menu** → **Open type**) we see that these interfaces are both defined in the com.ibm.cics.model (as shown in Figure 6-24) package, which we can access from our plug-ins.

Tip: This technique can be applied to detecting and identifying any selection events from the CICS Explorer.



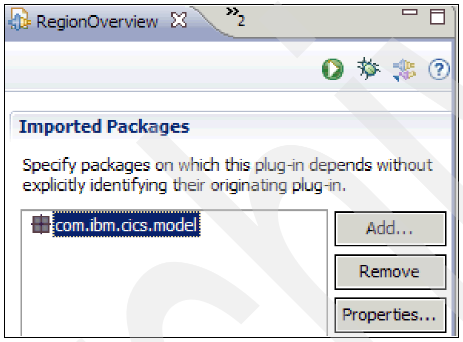
```
IManagedRegion
IManagedRegion - com.ibm.cics.model
com.ibm.cics.model - C:\eclipse3.5.1...bm.cics.model_1.0.0.200907301708.jar
```

```
IRegion
IRegion - com.ibm.cics.model - C:\eclipse3.5.1\plugins\com.ibm.cics.model_1.0.0
IRegion - org.eclipse.jdt.core - C:\eclipse3.5.1\plugins\org.eclipse.jdt.core_3.5.
IRegion - org.eclipse.jface.text - C:\eclipse3.5.1\plugins\org.eclipse.text_3.5.0.
com.ibm.cics.model - C:\eclipse3.5.1\plu...om.ibm.cics.model_1.0.0.200907301708.jar
```

Figure 6-24 Open type IRegion & IManagedRegion

Now that we know what events we are expecting from the CICS Explorer, our next challenge is extracting the information we require from these events that we have received. If we look at the methods in these interfaces we see that the IRegion Interface has a `getJobname()` method that returns a string. This is what we need to find the CICS region in our managed system list, with a little work. The IManagedRegion Interface does not have such a useful method and presents us with additional challenges, so for now we focus on getting the IRegion selection event working and come back to the IManagedRegion event later.

We can react to the IRegion event from the CICS Explorer with a simple piece of code, but we need to give the Region Overview plug-in access to the com..ibm.cics.model package so that we can cast the received event into the appropriate type. Select `plugin.xml` in the RegionOverview and go to the Dependencies tab. Here we add the package to the Imported Packages list as shown in Figure 6-25.



Imported Packages

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

com.ibm.cics.model

Figure 6-25 Region Overview add dependencies

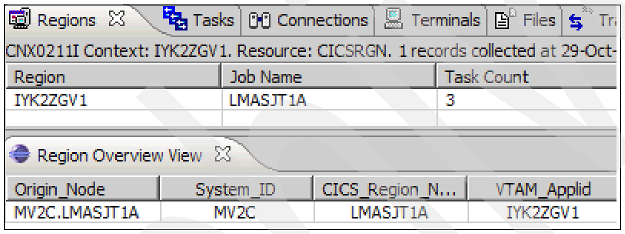
Next, add another clause to the IF statement in our selection listener in RegionOverviewView, as shown in Example 6-28.

Example 6-28 IRegion clause in selection listener

```
if(!selection.isEmpty()) {
        Object firstElement =
((StructuredSelection)selection).getFirstElement();
        if (firstElement instanceof String[]) {
                .
                .
                .
        }
        else if (firstElement instanceof IRegion) {
                IRegion region =(IRegion) firstElement;
                String jobname = region.getJobName();
                // try to find an ORIGINNODE for this jobname
                String originNode =
SoapInterface.getOriginNode(jobname);
                if (originNode != null) {
                        setOriginNode(originNode);
                }
        }
}
```

This additional code extracts the jobname from the received event and uses the code we wrote to look up the ORIGINNODE of CICS region. Given the ORIGINNODE, we can drive the Region Overview report in the same way as when a CICS region was selected in the managed system list.

As the IRegion event is fired when a row is selected in the Regions table view in the CICS Explorer, we can now test this code. We get results like those shown in Figure 6-26.



CNX0211I Context: IYK2ZGV1. Resource: CICSRGN. 1 records collected at 29-Oct-

Region	Job Name	Task Count
IYK2ZGV1	LMASJT1A	3

Origin_Node	System_ID	CICS_Region_N...	VTAM_Applid
MV2C.LMASJT1A	MV2C	LMASJT1A	IYK2ZGV1

Figure 6-26 CICS Explorer Regions and OMEGAMON

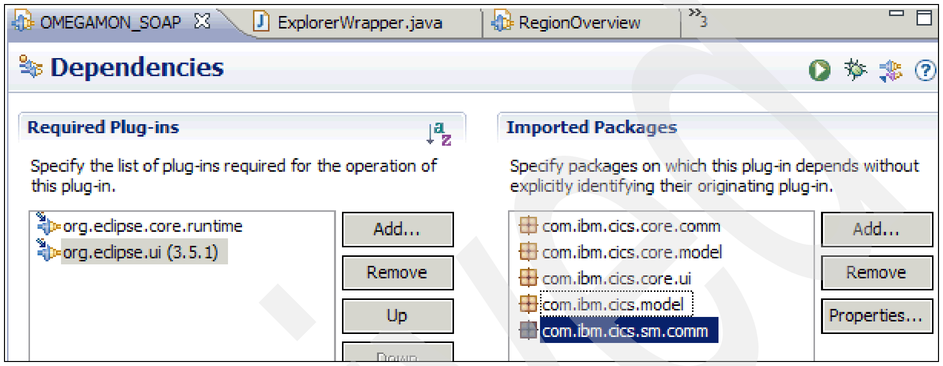
We can now move on to the challenge of handling events from the navigation tree, the IManagedRegion events. The IManagedRegion does not have a reference to a jobname, so we have to go through several other steps.

To get the jobname, we have to get an instance of an IRegion object for ourselves by driving the CICS Explorer's API. The IManagedRegion object has a reference to the name that CICS Explorer uses to identify the CICS region. This is the main thing that we need when we want to get an IRegion instance. Because we expect to use this several plug-ins, we place this code in the OMEGAMON_SOAP plug-in. The code we use is based on the code introduced in 5.1.3, "Using our sample code" on page 98, but with a few differences. For simplicity, we have placed all this code in static methods in a new class called ExplorerWrapper in the OMEGAMON_SOAP plug-in. This new class is shown in Example 6-29 on page 215. We have included the import statements for clarity.

Example 6-29 ExplorerWrapper.java

```
package omegamon_soap;

import com.ibm.cics.core.model.CICSTypes;
import com.ibm.cics.core.model.ICPSM;
import com.ibm.cics.core.model.IResourcesModel;
import com.ibm.cics.core.model.ScopedContext;
import com.ibm.cics.core.ui.UIPlugin;
import com.ibm.cics.model.ICICSplex;
import com.ibm.cics.model.IManagedRegion;
import com.ibm.cics.model.IRegion;

public class ExplorerWrapper {
   public static String getJobName(IManagedRegion managedRegion) {
      ICPSM cpsm = (ICPSM)
         UIPlugin.getDefault().getResourceManager(
         UIPlugin.SYSTEM_MANAGER_CONNECTION_ID);
      if (cpsm != null) {
         // we need to find the context for the passed managed region
         String context = getContext(cpsm, managedRegion);
         if (context != null) {
            ScopedContext scopedContext =
               new ScopedContext(context, managedRegion.getName());
            IResourcesModel regionModel =
                  cpsm.getModel(CICSTypes.Region, scopedContext);
            if (regionModel != null) {
               regionModel.activate();
               regionModel.maybeFetch(0,-1);
               if (regionModel.size() > 0) {
                  regionModel.maybeFetch(0,regionModel.size());
                  IRegion region = (IRegion) regionModel.get(0);
                  String jobname = region.getJobName();
                  return jobname;
               }
            }
         }
      }
      // failed to find the jobname
      return null;
   }

   private static String getContext(ICPSM cpsm, IManagedRegion
managedRegion) {
      // find the plex containing the passed managed region
      ICICSplex[] plexes = cpsm.getCICSplexes();
      if (plexes != null) {
         if (plexes.length == 1) {
            // only one plex - must be the right one
            return plexes[0].getName();
         }
         else {
            // test each plex in turn
            String searchRegionName = managedRegion.getName();
            for (ICICSplex plex : plexes) {
               IManagedRegion[] regions = cpsm.getManagedRegions(plex);
               for (IManagedRegion region : regions) {
                  if (region.getName().equals(searchRegionName)) {
                     // found the owning cmas
                     return plex.getName();
                  }
               }
            }
         }
      }
      // we have failed to find the owning cmas
      return null;
   }
}
```

When you create the ExplorerWrapper class as shown in Example 6-29 on page 215, you have multiple compile errors due to the plug-in not having dependencies on the necessary CICS Explorer plug-ins. By a process of searching for the classes referenced in the errors using the technique described in 4.3.3, “Adding CICS Explorer code to the Eclipse Template code” on page 82, we found that the required dependencies that have to be added to OMEGAMON_SOAP are as shown in Figure 6-27.



Specify the list of plug-ins required for the operation of this plug-in.

- org.eclipse.core.runtime
- org.eclipse.ui (3.5.1)

Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.

- com.ibm.cics.core.comm
- com.ibm.cics.core.model
- com.ibm.cics.core.ui
- com.ibm.cics.model
- com.ibm.cics.sm.comm

Figure 6-27 OMEGAMON_SOAP Explorer dependencies

The code in Example 6-29 on page 215 deserves discussion. The only public method is the `getJobName(…)` method. It is passed an IManagedRegion instance, and it returns the jobname for that CICS region. It starts by getting a connection to the host through the CICS Explorer API (the cpsm reference). After we have this reference, we need to find the context for the passed IManagedRegion. This is achieved by the `getContext` method, which queries every CICS region name on every plex until it finds a matching region name. It can then return the owning CMAS.

Note: The ManagedRegion object actually has a reference to its context, but this is not currently available through the IManagedRegion interface. As the CICS Explorer SDK is still evolving, this reference might become available in the future, which removes the need for the `getContext(…)` method.

After we have the context, we can query the IManagedRegion for its name, which is the scope for the query we are about to issue. Given the context and scope, we can now request the CICSTypes.Region (IRegion) that we require. When we receive this object we cast it into an IRegion and extract the jobname that we return.

This code can be easily modified to retrieve a jobname based on other objects received from the CICS Explorer.

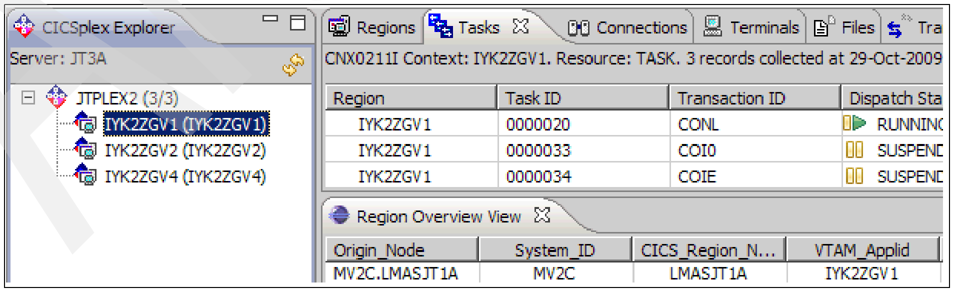
The final piece of code is another clause in the if statement in our selection listener in RegionOverviewView, as shown in Example 6-30.

Example 6-30 IManagedRegion clause in selection listener

```
if(!selection.isEmpty()) {
        Object firstElement =
((StructuredSelection)selection).getFirstElement();
        if (firstElement instanceof String[]) {
                .
                .
                .
        }
        else if (firstElement instanceof IRegion) {
                .
                .
                .
        }
        else if (firstElement instanceof IManagedRegion) {
                IManagedRegion managedRegion = (IManagedRegion) firstElement;
                String jobname = ExplorerWrapper.getJobName(managedRegion);
                String originNode = SoapInterface.getOriginNode(jobname);
                if (originNode != null) {
                        setOriginNode(originNode);
                }
        }
}
```

This code drives the `getJobName(…)` method we described and then drives the OMEGAMON code to fetch the region overview for the selected CICS region.

If we run the code now, we can see the OMEGAMON Region Overview for any CICS region selected in the tree, without having to select that CICS region in the CICS Explorer Regions view, as shown in Figure 6-28.



CNX0211I Context: IYK2ZGV1. Resource: TASK. 3 records collected at 29-Oct-2009

Region	Task ID	Transaction ID	Dispatch Sta
IYK2ZGV1	0000020	CONL	RUNNINC
IYK2ZGV1	0000033	COI0	SUSPENC
IYK2ZGV1	0000034	COIE	SUSPENC

Origin_Node	System_ID	CICS_Region_N...	VTAM_Applid
MV2C.LMASJT1A	MV2C	LMASJT1A	IYK2ZGV1

Figure 6-28 CICS Explorer tree selection and OMEGAMON

We now have a plug-in that displays OMEGAMON data related to the currently selected CICS region in CICS Explorer. It is a simple matter to modify this code to display other OMEGAMON reports based on other selections in the CICS Explorer.

6.9 Summary

In this chapter we have discussed the SOAP interface into the Tivoli Enterprise Management Server (TEMS) that allows access to a variety of OMEGAMON data. We have used this interface to provide information for several plug-ins that are useful in their own right, but that also provide examples from which to create your own reports displaying other OMEGAMON data.

We have also demonstrated how to drive OMEGAMON reports based on selections in the CICS Explorer. This code can be used as a basis for other reports that you might want to create.

The code and techniques described here can also be used when writing other extensions to the CICS Explorer for data sources other than OMEGAMON. Additionally, these examples can be used to integrate OMEGAMON data with products other than CICS Explorer.

Other possible extensions to the plug-ins created here include other representations of the data, such as graphs or a hierarchical view of the managed system list that is similar to the to the physical navigation tree view in the Tivoli Enterprise Portal (TEP) client. That requires interesting code to convert the data into a tree structure, but after this had been done, it provides a useful plug-in from which to navigate through the available data that can be retrieved from the SOAP interface.

6.10 References and further reading

The information and code presented in this chapter gives you a good starting point when writing your own Eclipse plug-ins that integrate OMEGAMON data and features with the CICS Explorer. The following is a list of several other sources of information that might prove useful.

6.10.1 IBM Tivoli Monitoring Information Center

This is the definitive guide to the components that make up the ITM architecture, and includes extensive details about the configuration and usage of the TEMS Web services interface. This includes a discussion of the other commands available in the SOAP interface that have not been discussed in this chapter.

6.10.2 OMEGAMON XE for CICS Information Center

This provides detailed information about the features available in OMEGAMON XE for CICS, and provides detailed information about the available reports that you can exploit in your plug-ins.

6.10.3 IBM Open Process Automation Library (OPAL)

This site brings together a selection of documents, best practices and tools for products across the range of Tivoli software:

http://www-01.ibm.com/software/brandcatalog/portal/opal/

6.10.4 IBM Tivoli Monitoring Eclipse plug-in

This plug-in makes use of many of the concepts demonstrated in this chapter and provides access to data from CICS and CICS Transaction Gateway:

http://www-01.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10OM1D

6.10.5 Using IBM Tivoli Monitoring V6.1 SOAP Services

This document provides an excellent overview of the TEMS Web services interface and its usage:

http://www-01.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10TM4M

7

Setting CICS Trace Levels through CICS Explorer

CICS tracing is a great tool for understanding how and why a system or application is experiencing a problem. It traces the flow of execution through CICS code and through user applications. You can see what functions are being performed, which parameters are being passed, and the values of important data fields at the time trace calls are made.

At defined trace points, CICS writes variable length tracing entries into the CICS trace table. Trace points are used to make trace entries when exception conditions occur. These trace points are always captured. Other trace points are used to trace the mainline execution of CICS code. These trace points are each associated with a level attribute. CICS allows you to control how much CICS system tracing is to be done based on the trace level setting for each trace component.

This chapter describes the necessary steps to create an CICS Explorer plug-in that allows you to set CICS trace levels dynamically for each trace component, similar to the functionality provided by the CICS CETR transaction.

7.1 The Trace Component Plug-in

In this example you create an Eclipse JFace viewer to display the CICS trace components and accept the users input of which components to turn on or off and what levels to set. It is important to understand three main concepts of viewers when dealing with JFace viewers:

- Viewer: The user interface (for example, the tableviewer in this example)
- Content provider: Provides the model objects to the viewer
- Label provider: Defines how the model objects are displayed

This project consists of three main components:

- A model that represents the data for the plug-in
- A view that displays the data
- A pop-up display that allows the view to be displayed

Begin by creating a blank plug-in and add the necessary extension points, packages, classes, and methods as you go along. This plug-in requires two images that must be located in the icons folder for the project. After you create your project, make sure you have the following two images in the icons folder:

- `checked.gif`
- `unchecked.gif`

You can find the images for these two icons in the additional materials for this IBM Redbooks publication.

7.2 Creating the project

Start off by creating a project to contain all the classes for your plug-in. Switch to the Plug-in Development perspective and select **File** → **New** → **Project** → **Plug-in project** and click **Next**. The first page of the wizard displays. Enter the project name as `cics.set.trace` and click **Next**.

On the second page of the wizard, set the default properties. Click **Next**.

Note: You are going to create a blank plug-in without using a template. However, if you want to create this plug-in using a template as in prior examples, in the third page of the wizard, select **Custom plug-in wizard** and click **Next.** Then, in the Template Selection page select **Popup Menu** and **View** and click **Next**. You need to add appropriate names for the view and pop-up menu settings. See 7.4.1, “Adding extension points and packages” on page 227 for naming conventions for this example.

In the third page of the wizard, clear the **Create a plug-in using one of the templates** check box and click **Finish**. Your plug-in project has now been created along with the default manifest files and the activator class. Because you did not use a template, the view class was not generated. You create this class in a later step.

An activator class controls the lifecycle of a plug-in as well as any images used in the plug-in that are managed by the ImageRegistry. If the project has a class that extends AbstractUIPlugin (for example, Activator.java), there must be a `getImageDescriptor()` method to access ImageDescriptors given a path to the image. In this example, you are using two check box images, so you must add this method to the Activator.java class.

Note: If you used a template to create this plug-in, the `getImageDescriptor()` method was generated automatically when the activator class was created.

Open the Activator.java class in the “set.cics.trace” package and add the code in Example 7-1.

Example 7-1 getImageDescriptor method

```
public static ImageDescriptor getImageDescriptor(String path) {
   return imageDescriptorFromPlugin(PLUGIN_ID, path);
}
```

You get an error because the ImageDescriptor class must be imported. Add the following import

```
import org.eclipse.jface.resource.ImageDescriptor;
```

Save and close the activator class. Next, you create the model structure for your project.

7.3 Creating the model

This project uses two simple classes that represents the data model in your design. The `Component` class is a simple object representing a trace component with getters and setters for the following properties:

```
private String compId;
private String stdTraceLevel;
private String spcTraceLevel;
private boolean selected;
```

The `ComponentModel` class is used to hold a collection of `Component` instances. So, let's create the model.

Within your project, create the package cics.set.trace.model (right-click the project, select **New** → **Package**). Create the `Component` class (Example 7-2) in this package.

Example 7-2 Component class

```
package cics.set.trace.model;

import java.beans.*;

public class Component {
   private String compId;
   private String stdTraceLevel;
   private String spcTraceLevel;
   private boolean selected;

   private PropertyChangeSupport propertyChangeSupport =
      new PropertyChangeSupport(this);

   public Component() {  }

   public Component(boolean selected, String compId,
         String stdTraceLevel, String spcTraceLevel) {
      super();
      this.selected = selected;
      this.compId = compId;
      this.stdTraceLevel = stdTraceLevel;
      this.spcTraceLevel = spcTraceLevel;
   }

   public void addPropertyChangeListener(
      String propertyName,PropertyChangeListener listener) {
         propertyChangeSupport
            .addPropertyChangeListener(propertyName, listener);
}

   public void removePropertyChangeListener(
      PropertyChangeListener listener) {
           propertyChangeSupport
                 .removePropertyChangeListener(listener);
   }

   public String getCompId() { return compId; }
   public String getSpcTraceLevel() { return spcTraceLevel; }
   public String getStdTraceLevel() { return stdTraceLevel; }
   public boolean isSelected() { return selected; }
   public void setCompId(String compId) {
      this.compId = compId;
   }

   public void setSpcTraceLevel(String spcTraceLevel) {
      propertyChangeSupport.firePropertyChange("spcTraceLevel",
         this.spcTraceLevel,
         this.spcTraceLevel = spcTraceLevel);
   }

   public void setStdTraceLevel(String stdTraceLevel) {
      propertyChangeSupport.firePropertyChange("stdTraceLevel",
         this.stdTraceLevel,
         this.stdTraceLevel = stdTraceLevel);
    }

   public void setSelected(Boolean isSelected) {
      propertyChangeSupport.firePropertyChange("selected",
         this.selected, this.selected = isSelected);
   }

   public String toString() {
      String text = compId + "1=";

      if (stdTraceLevel.equals("1")) text += "1";
      else if (stdTraceLevel.equals("1-2")) text += "2";
      else if (stdTraceLevel.equals("OFF")) text += "0";
      else text += stdTraceLevel;

      text += "&" + compId + "2=";
      if (spcTraceLevel.equals("1-2")) text += "2";
      else if (spcTraceLevel.equals("OFF")) text += "0";
      else text += spcTraceLevel;

      return text;
   }
}
```

As you review this class, notice there is one additional parameter specified, properChangeSupport. This parameter is used to add a PropertyChangeListener to each property. A Property Change Event occurs when a property value has changed. In this plug-in you have three properties that can change:

- selected
- stdTraceLevel
- spcTraceLevel

You add a PropertyChangeListener to the setter methods for each property to listen for any changes.

Next, create the `ComponentModel` class in this same package (Example 7-3). This module initializes each of the trace components levels for both standard and special and provides a collection of all the trace components. It sets all standard tracing to level 1 and all special tracing to level 1–2, unless that component only has level 1 tracing.

Example 7-3 ComponentModel class

```
package cics.set.trace.model;

import java.util.*;

public class ComponentModel {
   private static ComponentModel content;
   private List<Component> components;
   // there are 65 trace components total
   public String[] compId = {
           "AP","BA","BM","BR","CP","DC","DD","DH","DM","DP",
           "DS","DU","EI","EJ","EM","EP","FC","GC","IC","IE",
           "II","IS","KC","KE","LC","LD","LG","LM","ME","ML",
           "MN","NQ","OT","PA","PC","PG","PI","PT","RA","RI",
           "RL","RM","RS","RX","RZ","SC","SH","SJ","SM","SO",
           "ST","SZ","TC","TD","TI","TR","TS","UE","US","WB",
           "WU","W2","XM","XS" };
   // these components only have Level 1 tracing
   private String[] spcL1Only = {
           "BM","DC","KC","PC","SC","SZ","TD","UE" };

   public ComponentModel() {
      components = new ArrayList<Component>();
      // initialize each entry
      for (int i=0; i<compId.length; i++) {
         Component component = new Component();
         component.setSelected(false);
         component.setCompId(compId[i]);
         component.setStdTraceLevel("1");

         if (Arrays.asList(spcL1Only).contains(compId[i]))
            component.setSpcTraceLevel("1");
         else
            component.setSpcTraceLevel("1-2");

         components.add(component);
      }
   }

   public static synchronized ComponentModel getInstance() {
      if (content != null)
         return content;
```

```
        content = new ComponentModel();
        return content;
    }

    public List<Component> getComponents() {
        return components;
    }
}
```

7.4 Creating the view

Now that you have the data model structure complete, you can create the view. Because you are working with a blank plug-in, you need to add the extension points and import any packages required for this plug-in.

Note: If you used a template in the earlier step, some of these extension points might already be added to your plug-in project.

7.4.1 Adding extension points and packages

First, add the extension points. In the manifests files, click the Extensions tab and click **Add** under All Extensions. On the "Extension Point Selection" page, select **org.eclipse.ui.views** under "Extension Points" and select **Sample View** under "Available templates". Click **Next**. On the next page, specify the following values:

- View Class Name: **TraceComponentView**
- View Name: **Trace Component View**
- View Category Name: **CICS Category**

You are using a table viewer for this view, so ensure it is selected. Click **Finish** to add this extension point and create the plug-in class. Notice that three extensions have been added to the Extensions tab for your plug-in.

You just added the view template to your project. Because this plug-in uses a pop-up menu as well, you need to add one more extension for the pop-up menu extension point. Click **Add** again and select **org.eclipse.ui.popupMenus** under "Extension Points" and select **Popup Menu** under "Available templates" and click **Next**. On the next page add this pop-up menu to the Regions resource, so specify the following values

- Target Object's Class: **com.ibm.cics.model.IRegion**
- Submenu Name: **CICS Trace**
- Action Label: **Set Components**

- Action Class: **SetComponents**

Click **Finish**. You might get a pop-up window asking to save the changes made to the plug-in. Click **Yes.** Notice that following extensions have been added to the Extensions tab for the plug-in:

- `org.eclipse.ui.views`
- `org.eclipse.ui.perspectiveExtensions`
- `org.eclipse.help.contexts`
- `org.eclipse.ui.popupMenus`

Next, add the CICS packages this plug-in requires. Because it is activated when a CICS region is selected, you are using the Region resource. Add that package and a few others. In the manifests files, click the Dependencies tab and click **Add** under “Required Plug-ins”. Add the following packages for the plug-in:

- `com.ibm.cics.core.comm`
- `com.ibm.cics.core.model`
- `com.ibm.cics.core.ui`

Note: If you do not know what packages are required, you can add them at a later time.

You are now ready to code the view and pop-up classes.

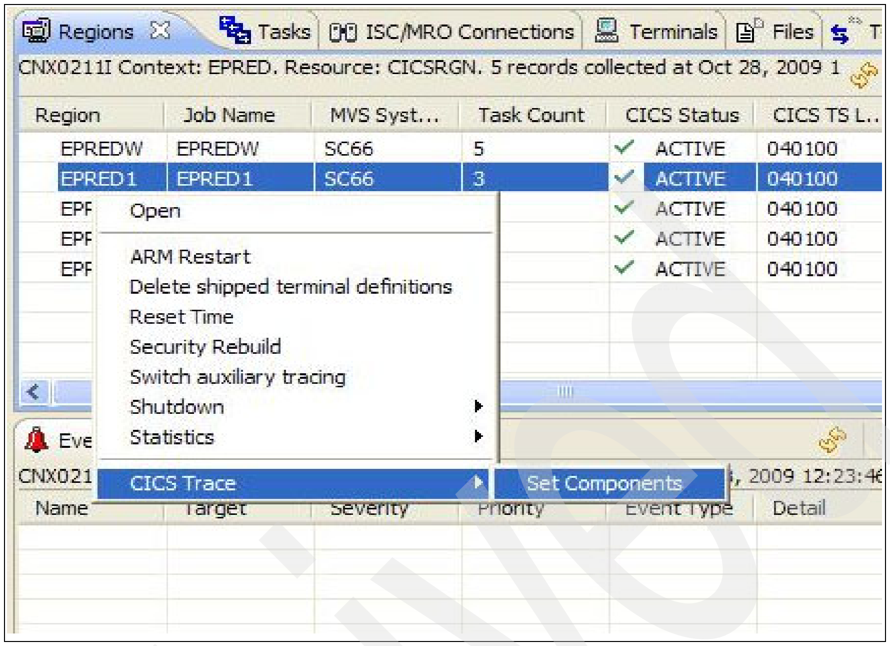
7.4.2 Modifying the pop-up action class

This view is opened when you right-click a CICS region and select **CICS Trace** → **Set Components**. You now add the code to make that happen. Open the pop-up class, SetComponents.java, in the cics.set.trace.pop-up.actions package. In this class you need to declare a global variable, region, and alter the `selectionChanged()` and `run()` methods. Modify this class to look like Example 7-4.

Example 7-4 SetComponents class

```
package cics.set.trace.popup.actions;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.*;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.*;

import com.ibm.cics.model.IRegion;

public class SetComponents implements IObjectActionDelegate {
   @SuppressWarnings("unused")
   private Shell shell;
   @SuppressWarnings("unused")
   private IRegion region;

   public SetTrace() {
      super();
   }

   public void setActivePart(IAction action,
                  IWorkbenchPart targetPart) {
      shell = targetPart.getSite().getShell();
   }

   public void run(IAction action) {
      final IWorkbenchPage page =
            PlatformUI.getWorkbench().getActiveWorkbenchWindow()
                 .getActivePage();
      try {
         @SuppressWarnings("unused")
         IViewPart view = page
            .showView("cics.set.trace.views.TraceComponentView");
      } catch (PartInitException e) {
         e.printStackTrace();
      }
   }
   public void selectionChanged(IAction action,
                  ISelection selection) {
      region = (IRegion)((StructuredSelection)selection)
                    .getFirstElement();
   }
}
```

Test your pop-up menu. Run the plug-in as an Eclipse application and connect to a WUI region. Right click a region in the Region view. See Figure 7-1.



CNX0211I Context: EPRED. Resource: CICSRGN. 5 records collected at Oct 28, 2009 1

Region	Job Name	MVS Syst...	Task Count	CICS Status	CICS TS L..
EPREDW	EPREDW	SC66	5	✓ ACTIVE	040100
EPRED1	EPRED1	SC66	3	✓ ACTIVE	040100
EPF				✓ ACTIVE	040100
EPF				✓ ACTIVE	040100
EPF				✓ ACTIVE	040100

CNX021 , 2009 12:23:4

Name	Target	Severity	Priority	Event Type	Detail

Figure 7-1 Set Components pop-up menu option

If you select **Set Components**, the view looks like Figure 7-2.

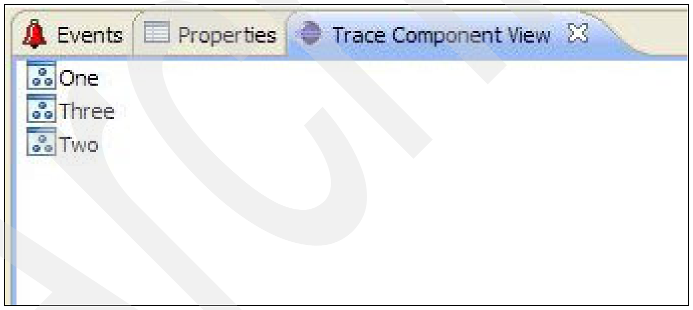


Figure 7-2 Trace Component View: Default

This shows that the pop-up is working as it opens the view. You might notice there is not much to this view, as this is the default view.

7.4.3 Modifying the view class

It is time to add the code for how the view looks. Expand the cics.set.trace.views package and open the TraceComponentView.java class. There are several default methods that were generated in this class. You modify a few of these classes and add a few more to create your view, but there are some you do not use. All other methods except for those in the following list might be deleted, as they are not used by this plug-in:

- NameSorter()
- makeActions()
- contributeToActionBars()
- fillLocalToolBar()
- setFocus()
- showMessage()

You also need to alter the `contributeToActionBars()` method and remove the reference to `fillLocalPullDown()`. Also alter the `createPartcontrol()` method and remove `hookContextMenu()` and `hookDoubleClickAction()`.

Begin coding. First, import your data model.

```
import java.io.*;
import java.net.*;
import java.util.List;
import cics.set.trace.model.*;
import com.ibm.cics.model.*;
import com.ibm.cics.core.model.*;
import com.ibm.cics.core.ui.*;
```

This class is defined with four variables already. You are using them all but you rename a few. Define the following global variables for your view:

```
private Table table;
private ICPSM cpsm;
private IRegion region;
private Context context;
private int portNum;
private String host;
List<Component> componentList =
        ComponentModel.getInstance().getComponents();
```

Then, perform the following steps:

1. Rename all references to `action1` to `selectAll`.
2. Rename all references to `action2` to `deselectAll`.
3. Rename all references to `doubleClickAction` to `submit`.

These parameters are used as you build the necessary methods. If you have all the dependencies and imported packages set up correctly, you do not receive any errors. Now, create the table by adding the following method, `createTable()`, to the class. This method creates a table of four columns to hold your data. See Example 7-5.

Example 7-5 createTable method

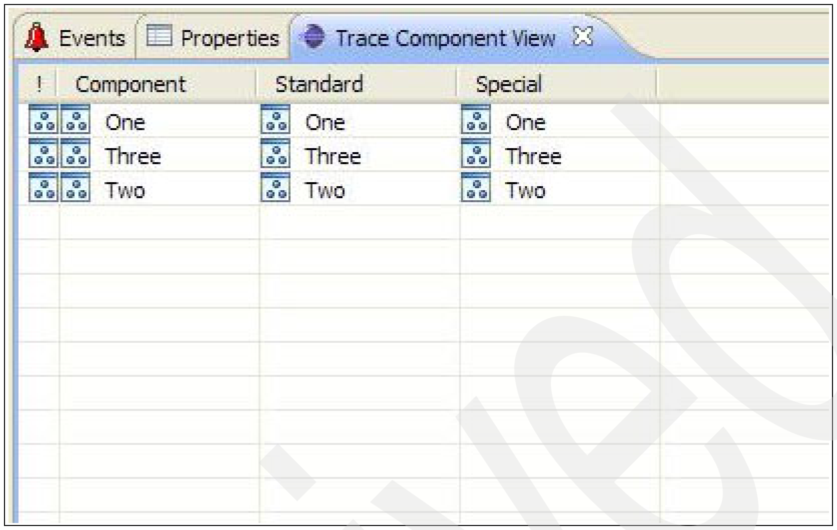
```
public void createTable(Composite parent) {
   String[] titles = { "!", "Component", "Standard", "Special" };
   int[] bounds = { 20, 100, 100, 100 };

   table = viewer.getTable();
   table.setHeaderVisible(true);
   table.setLinesVisible(true);

   for (int i = 0; i < titles.length; i++) {
      TableViewerColumn column =
               new TableViewerColumn(viewer, SWT.NONE);
      column.getColumn().setText(titles[i]);
      column.getColumn().setWidth(bounds[i]);
      column.getColumn().setResizable(true);
   }
}
```

You need to add a call to the `createTable()` method in the `createPartControl()` method for the table to be set up. The modified version of this class looks like Example 7-6.

Example 7-6 Modified createPartcontrol method

```
public void createPartControl(Composite parent) {
   System.out.println("In createPartControl...");
   viewer =
         new TableViewer(parent, SWT.MULTI | SWT.V_SCROLL
                | SWT.FULL_SELECTION);

   createTable(parent);

   viewer.setContentProvider(new ViewContentProvider());
   viewer.setLabelProvider(new ViewLabelProvider());
   viewer.setSorter(new NameSorter());
   viewer.setInput(getViewSite());

   // Create the help context id for the viewer's control
   PlatformUI.getWorkbench().getHelpSystem()
         .setHelp(viewer.getControl(), "cics.set.trace.viewer");
   makeActions();
   contributeToActionBars();
}
```

If you run the example, the view contains a table with four columns, as shown in Figure 7-3.



!	Component	Standard	Special
	One	One	One
	Three	Three	Three
	Two	Two	Two

Figure 7-3 Trace Component View: Table with four columns

Next, alter the ContentProvider to indicate what data objects to use and the LabelProvider to define how the data is displayed. Instead of creating your own classes, alter the `ViewContentProvider()` and `ViewLabelProvider()` methods. You need to import the Activator, so add the following import statement:

```
import cics.set.trace.Activator;
```

The updated ViewContentProvider and ViewLabelProvider are shown in Example 7-7.

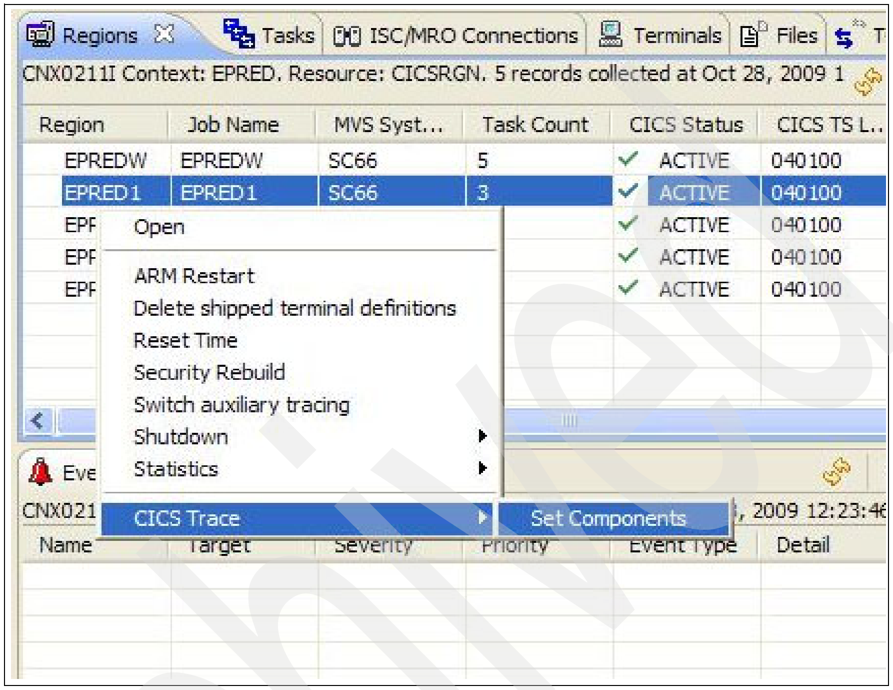
Example 7-7 Modified ViewContentProvider and VIewLabelProvider methods

```
class ViewContentProvider implements IStructuredContentProvider {
   @SuppressWarnings("unchecked")
   public Object[] getElements(Object inputElement) {
      List<Component> components = (List<Component>) inputElement;
      return components.toArray();
   }
   public void dispose() {
   }
   public void inputChanged(Viewer viewer, Object oldInput,
            Object newInput) {
   }
}

class ViewLabelProvider extends LabelProvider
```

```
                        implements ITableLabelProvider {
   private final Image CHECKED = Activator.getImageDescriptor(
         "icons/checked.gif").createImage();
   private final Image UNCHECKED = Activator.getImageDescriptor(
         "icons/unchecked.gif").createImage();

   public Image getColumnImage(Object element, int columnIndex) {
      // In case you don't like image just return null here
      if (columnIndex == 0) {
         if (((Component) element).isSelected()) {
            return CHECKED;
         } else {
            return UNCHECKED;
         }
      }
      return null;
   }
   public String getColumnText(Object element, int columnIndex) {
      Component component = (Component) element;
      switch (columnIndex) {
         case 0: return String.valueOf(component.isSelected());
         case 1: return component.getCompId();
         case 2: return component.getStdTraceLevel();
         case 3: return component.getSpcTraceLevel();
         default: throw new RuntimeException("Should not happen");
      }
   }
}
```

Before you can test the plug-in, verify the two icon files are in the icons folder for your project. You must alter the createPartControl() method to set the input. In createPartControl() change the statement `viewer.setInput(getViewSite())` to `viewer.setInput(componentList);`

Run the plug-in as an Eclipse application. In the new Eclipse window that opens, connect to the host and right-click a region in the Regions tab as shown in Figure 7-4.



CNX0211I Context: EPRED. Resource: CICSRGN. 5 records collected at Oct 28, 2009 1

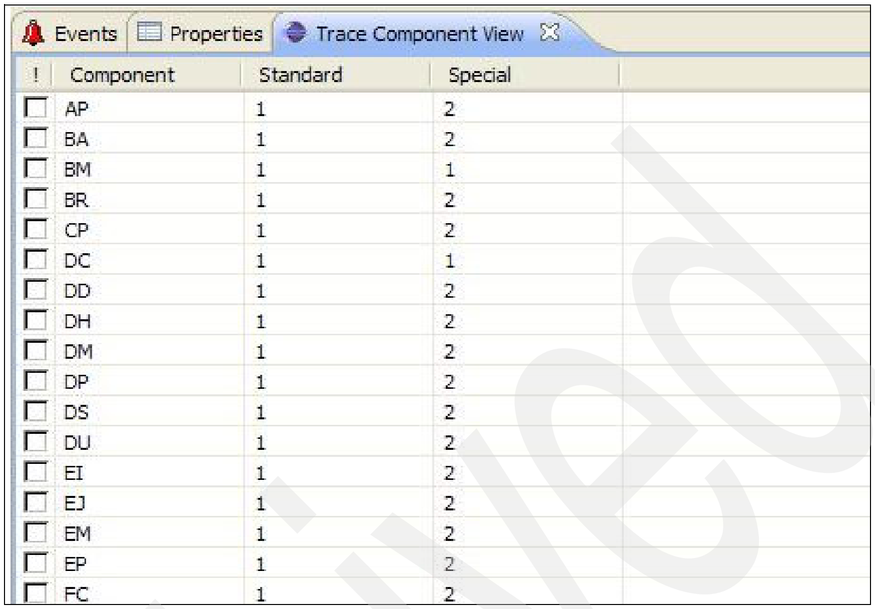
Region	Job Name	MVS Syst...	Task Count	CICS Status	CICS TS L..
EPREDW	EPREDW	SC66	5	✓ ACTIVE	040100
EPRED1	EPRED1	SC66	3	✓ ACTIVE	040100
EPF				✓ ACTIVE	040100
EPF				✓ ACTIVE	040100
EPF				✓ ACTIVE	040100

CNX021 ..., 2009 12:23:4

Name	Target	Severity	Priority	Event Type	Detail

Figure 7-4 Set Components pop-up menu option

Select **CICS Trace** → **Set Components**. The view shown in Figure 7-5 displays.



Events | Properties | Trace Component View

!	Component	Standard	Special
☐	AP	1	2
☐	BA	1	2
☐	BM	1	1
☐	BR	1	2
☐	CP	1	2
☐	DC	1	1
☐	DD	1	2
☐	DH	1	2
☐	DM	1	2
☐	DP	1	2
☐	DS	1	2
☐	DU	1	2
☐	EI	1	2
☐	EJ	1	2
☐	EM	1	2
☐	EP	1	2
☐	FC	1	2

Figure 7-5 Trace Component View - using data model

At this point, you can scroll up and down but none of the fields are yet editable. Add this part next so that you can select which components to set and what level to which to set them. Use the JFace CellEditor class to specify how the user can edit each cell in the table. Each column contains the following items:

- Column 1: A selectable check box
- Column 2: An uneditable text field
- Column 3 and 4: Drop-down list boxes displaying possible options to set.

Create a new class, TraceEditor, which extends the EditingSupport class in the `cics.set.trace.models` package to define what each column is and what data it displays. Create the class shown in Example 7-8.

Example 7-8 TraceEditor class

```
package cics.set.trace.model;

import java.util.Arrays;

import org.eclipse.jface.viewers.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.*;

public class TraceEditor extends EditingSupport {
```

```
private CellEditor editor;
 private Composite parent;
private int column;
// arrays for the various levels of tracing
 private String[] level1= { "OFF", "1" };
 private String[] level2 = { "OFF", "1", "1-2" };
 private String[] level3 = { "OFF", "1", "1-2", "3" };
 // arrays for components with level 1 tracing only
private String[] l1Only = {
        "BM","DC","KC","PC","SC","SZ","TD","UE" };
 // arrays for components with level 1 and 2 tracing only
 private String[] l2Only = {
        "AP","BR","CP","EC","EI","FC","IC","RA",
        "RI","SJ","TC","WB","WU" };

 public TraceEditor(ColumnViewer viewer, int column) {
   super(viewer);
   parent = ((TableViewer) viewer).getTable();
   this.column = column;
}

protected boolean canEdit(Object element) {
   return true;
}

protected CellEditor getCellEditor(Object element) {
   Component c = (Component) element;
   // create the correct editor based on the column index
   switch (this.column) {
   case 0:
      editor = new CheckboxCellEditor(null, SWT.CHECK |
                          SWT.READ_ONLY);
      break;
   case 1:
      editor = new TextCellEditor(parent, SWT.READ_ONLY);
      break;
   case 2:
      if (Arrays.asList(l1Only).contains(c.getCompId()))
         editor = new ComboBoxCellEditor(parent, level1);
      else if (Arrays.asList(l2Only).contains(c.getCompId()))
         editor = new ComboBoxCellEditor(parent, level2);
      else
         editor = new ComboBoxCellEditor(parent, level3);
      break;
   case 3:
      if (Arrays.asList(l1Only).contains(c.getCompId()))
         editor = new ComboBoxCellEditor(parent, level1);
      else if (Arrays.asList(l2Only).contains(c.getCompId()))
         editor = new ComboBoxCellEditor(parent, level2);
      else
         editor = new ComboBoxCellEditor(parent, level3);
      break;
   default:break;
   }
   return editor;
}

protected Object getValue(Object element) {
   Component c = (Component) element;
   switch (this.column) {
   case 0:return c.isSelected();
   case 1:return c.getCompId();
   case 2:if (c.getStdTraceLevel().equals("OFF"))
               return 0;
            else if (c.getStdTraceLevel().equals("1"))
               return 1;
            else if (c.getStdTraceLevel().equals("1-2"))
               return 2;
            else if (c.getStdTraceLevel().equals("3"))
               return 3;
   case 3:if (c.getSpcTraceLevel().equals("OFF"))
               return 0;
            else if (c.getSpcTraceLevel().equals("1"))
               return 1;
            else if (c.getSpcTraceLevel().equals("1-2"))
               return 2;
            else if (c.getSpcTraceLevel().equals("3"))
               return 3;
   default:break;
   }
   return null;
}

protected void setValue(Object element, Object value) {
   Component c = (Component) element;
   switch (this.column) {
   case 0:c.setSelected((Boolean) value);
            break;
   case 1:c.setCompId(String.valueOf(value));
            break;
   case 2:if (((Integer) value) == 1) {
               c.setStdTraceLevel("1");
               c.setSelected(true);
            } else if (((Integer) value) == 2) {
               c.setStdTraceLevel("1-2");
               c.setSelected(true);
            } else if (((Integer) value) == 3) {
               c.setStdTraceLevel("3");
```

```
                c.setSelected(true);
             } else {
                c.setStdTraceLevel("OFF");
                c.setSelected(true);
             }
             break;
      case 3:if (((Integer) value) == 1) {
                c.setSpcTraceLevel("1");
                c.setSelected(true);
             } else if (((Integer) value) == 2) {
                c.setSpcTraceLevel("1-2");
                c.setSelected(true);
             } else if (((Integer) value) == 3) {
                c.setSpcTraceLevel("3");
                c.setSelected(true);
             } else {
                c.setSpcTraceLevel("OFF");
                c.setSelected(true);
             }
             break;
      default:break;
      }
      getViewer().update(element, null);
   }
}
```

The main methods within this class do the following:

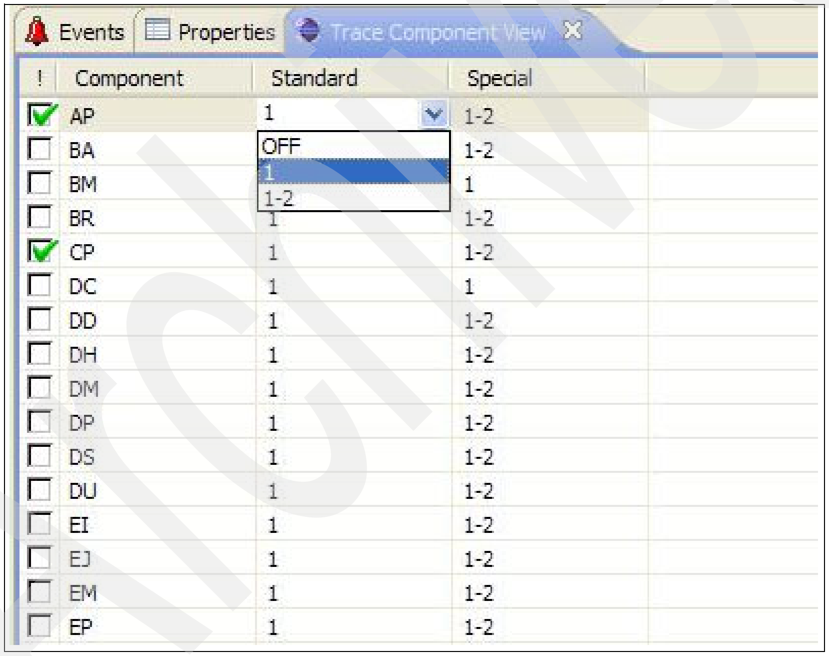
- The `getCellEditor` method returns the celleditor you want to use. This is implemented using a switch statement and is based on the column.
- The `setValue` method receives a new value from the user input and sets the new value to the object.
- The `getValue` method receives the object that was changed and returns the value for the table.

Next, assign the editors to the table columns. This is done in the `createTable()` method in the TraceComponentView.java class. Modify this method as shown in Example 7-9.

Example 7-9 Modified createTable method

```
public void createTable(Composite parent) {
   System.out.println("In createTable...");
   String[] titles = { "!", "Component", "Standard", "Special" };
   int[] bounds = { 20, 100, 100, 100 };

   table = viewer.getTable();
   table.setHeaderVisible(true);
   table.setLinesVisible(true);

   for (int i = 0; i < titles.length; i++) {
     TableViewerColumn column = new TableViewerColumn(viewer,
             SWT.NONE);
     column.getColumn().setText(titles[i]);
     column.getColumn().setWidth(bounds[i]);
     column.getColumn().setResizable(true);
     column.setEditingSupport(new TraceEditor(viewer, i));
   }
}
```

Run the plug-in. You are now able to edit the columns as shown in Figure 7-6. The check boxes are now selectable and the fields in the Standard and Special columns are now drop-down lists displaying all the available trace settings for each given component.



!	Component	Standard	Special
✓	AP	1	1-2
☐	BA	OFF	1-2
☐	BM	1	1
☐	BR	1-2	1-2
✓	CP	1	1-2
☐	DC	1	1
☐	DD	1	1-2
☐	DH	1	1-2
☐	DM	1	1-2
☐	DP	1	1-2
☐	DS	1	1-2
☐	DU	1	1-2
☐	EI	1	1-2
☐	EJ	1	1-2
☐	EM	1	1-2
☐	EP	1	1-2

Figure 7-6 Plug-in with editable fields

You now have your view the way you want it. The next step is to add functionality so that you can submit the changes to the table to the host system, setting the actual trace components to the specified level. You start with the three action buttons you renamed in an earlier step (that is, selectAll, deselectAll, and submit). Edit the `fillLocalToolBar()` method as shown in Example 5-10 to add each of these buttons to the local toolbar for your view.

Example 7-10 Modified fillLocalToolBar methods

```
// add the 3 buttons to the local tool bar
private void fillLocalToolBar(IToolBarManager manager) {
   manager.add(selectAll);
   manager.add(deselectAll);
   manager.add(submit);
}
```

Edit the `makeActions()` method as shown in Example 5-11 to specify what each button does when it is selected.

Example 7-11 Modified makeActions method

```
// specify the action for each button
private void makeActions() {
   selectAll = new Action() {
      public void run() {
         for (Component c: componentList) {c.setSelected(true);}
         viewer.refresh();
      }
   };
   selectAll.setText("Select All");

   deselectAll = new Action() {
      public void run() {
         for (Component c: componentList) {c.setSelected(false);}
         viewer.refresh();
      }
   };
   deselectAll.setText("Deselect All");

   submit = new Action() {
      private String text;

      public void run() {
         text = "";
         for (Component c : componentList) {
            if (c.isSelected()) { text += c + "&"; }
         }
         // submit request if port available
         if (portNum > 0) {
            text = text.substring(0, text.length() - 1);
            setTracing(portNum, text);
         } else {
             showMessage("You must first define a TCPIPService"
                + "number to submit this request.");
         }
      }
   };
   submit.setText("Submit");
}
```

The **selectAll** and **deSelectAll** buttons toggle the check box on and off for all components. The main button here is the **Submit** button, which calls the `setTracing()` method. When this button is clicked, a URL request is built based on the selected trace components. For this example to work, a TCPIPService definition with a valid port must be available for the selected CICS region (for example, HTTPNSSL). Before you take a look at the `setTracing()` method that creates the URL and sends the request to CICS, create an `init()` method that obtains the region name and available port number each time a region is selected. Add the method shown in Example 7-12 to the TraceComponentView class.

Example 7-12 init method

```
public void init(IViewSite site) throws PartInitException {
   super.init(site);
   // add selection listener to look for when users click a
   // Region entry in the Region view to acquire region name
   site.getPage().addSelectionListener(new ISelectionListener() {
      public void selectionChanged(IWorkbenchPart arg0,
                  ISelection selection) {
         if (!selection.isEmpty()) {
            Object firstElement =
               ((StructuredSelection)selection).getFirstElement();
            if (firstElement instanceof IRegion) {
               region = (IRegion) firstElement;
               portNum = getTcpipPort(region);
            }
         }
      }
   });
}
```

As you can see, this method calls the `getTcpipPort()` method to obtain the available port number. Add the method in Example 7-13 to the class.

Example 7-13 getTcpipPort method

```
public int getTcpipPort(IRegion region) {
   int port = 0;
```

```
    cpsm = UIPlugin.getDefault().getCPSM();
    ICICSplex[] cicsplexes = cpsm.getCICSplexes();
    //set scopedContext(CPSM,CPSM)
    host = cpsm.getHost();
    ScopedContext scopedContext = new ScopedContext
          (cicsplexes[0].getName(),cicsplexes[0].getName());
    plexContext = new Context(scopedContext.getContext());
    ITCPIPService[] tcpipServices = null;
    IResourcesModel model = cpsm.getModel(CICSTypes.TCPIPService,
       plexContext);
    model.activate(); // Initiate the model
    if (model.size() > 0) { // Check there are objects to get
       tcpipServices = new ITCPIPService[model.size()];
       model.maybeFetch(0, model.size()); // Fetch the results
       for (int i = 0; i < model.size(); i++)
          tcpipServices[i] = (ITCPIPService) model.get(i);
    }
    for (ITCPIPService tcpipService : tcpipServices) {
       if ("HTTPNSSL".equals(tcpipService.getName())) {
          if(tcpipService.getRegionName().equals(region.getName()))
             port = tcpipService.getPort().intValue();
          else {
                port = 0;
                showMessage("There is currently no port available "
                      + "in region " + region.getName()
                      + ". You need to define a TCPIPService
                      + "definition for HTTPNSSL");
          }     }
    }
    return port;
}
```

The `getTcpipPort()` method does the following:

- Obtain the host name of the connected CICSPlex SM
- Obtain the name of the Context for the selected CICSPlex
- Obtain a list of all TCPIPService definitions within the sysplex
- Search this list for a TCPIPService named HTTPNSSL
- Check this HTTPNSSL definition is defined in the selected CICS region

  If yes, get the port number. Otherwise, put out a message that a TCPIPService definition is required for the selected region.

Return the port number and then call the `setTracing()` method to build the URL request and send it to the host system. Next, create the `setTracing()` method (Example 7-14).

Example 7-14 setTracing method

```
private void setTracing(int port, final String t) {
   HttpURLConnection conn = null;
   BufferedReader rd  = null;
   StringBuilder sb = null;
   String line = null;
   String url = null;
   URL serverAddress = null;

   try {
      url = "http://"+host+":"+port+"/CICS/CWBA/COBSETTR?"+t;
      serverAddress = new URL(url);
      //set up out communications stuff
      conn = null;
      //Set up the initial connection
      conn = (HttpURLConnection)serverAddress.openConnection();
      conn.setRequestMethod("GET");
      conn.setDoOutput(true);
      conn.setReadTimeout(10000);
      conn.connect();

      //read the result from the server
      rd  = new BufferedReader(new
           InputStreamReader(conn.getInputStream()));
      sb = new StringBuilder();

      while ((line = rd.readLine()) != null) {
         sb.append(line + '\n');
      }
         MessageDialog.openInformation(
            viewer.getControl().getShell(),
            "Result",
            sb.toString());
   } catch (MalformedURLException e) {
      e.printStackTrace();
   } catch (ProtocolException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   } finally {
      //close the connection, set all objects to null
      conn.disconnect();
      rd = null;
      sb = null;
      conn = null;
   }
}
```

7.5 CICS TS Application

You have now completed all the coding necessary on the front-end. The last piece required is the back-end program that gets called (COBSETTR). This program is a CICS Web Support SPI program written in COBOL which performs the following tasks:

- Executes an EXEC CICS INQUIRE TRACETYPE (STANDARD|SPECIAL) for each component to inquire on the current value
- Parses the HTML data sent by CICS Explorer to extract each component name and level to set for both standard and special tracing
- Executes an EXEC CICS SETTRACETYPE (STANDARD|SPECIAL) for each component to set the new value
- Returns an HTTP response to indicate the success of the request to CICS Explorer

The COBSETTR program is supplied in the additional materials for this book. You need to make sure you have a TCPIPService definition called HTTPNSSL set with CWBA in the host system.

After you have the back-end program in place and loaded into the load library, execute this plug-in and the following actions occur:

- When you click the **Select All** button, all components are selected.
- When you click the **De-select All** button, all components are unselected.
- When you click the **Submit** button, the URL request is sent to the host system kicking off the COBSETTR API program, the trace components are set for the selected region, and the message in Figure 7-7 is displayed upon completion.

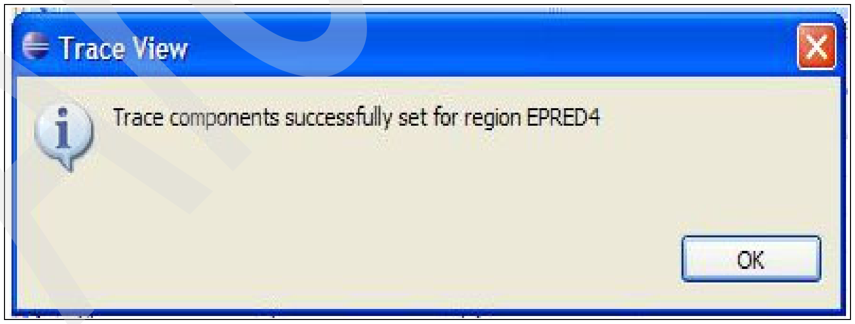

Trace components successfully set for region EPRED4

Figure 7-7 Message box indicating trace successfully set on host

Press the **Enter** key after making your last selection before clicking the **Submit** button in order for all changes to take affect.

Note: There are few features that could be added to this plug-in if time allowed. They are listed here for future reference:

- The plug-in does not poll the host system to query what the current settings for CICS trace components are. Therefore, it does not recall what was set or is set in the region.
- An option to turn all trace components off by the click of a button.

8

Adding a sticky note plug-in to CICS Explorer

In this chapter we document a plug-in that implements a sticky note memo function in your CICS Explorer. You can use this facility to annotate the resources in your CICS Explorer workspace, and to view and manage the sticky notes that have been created.

From the point of view of writing plug-ins for CICS Explorer, what is interesting about this plug-in is that it demonstrates how you can provide additional input data that is useful to you, but which is not related to mainline CICS Explorer function. This pluf, and have that data saved and managed by a plug-in beyond the standard provide input date and data this wtin

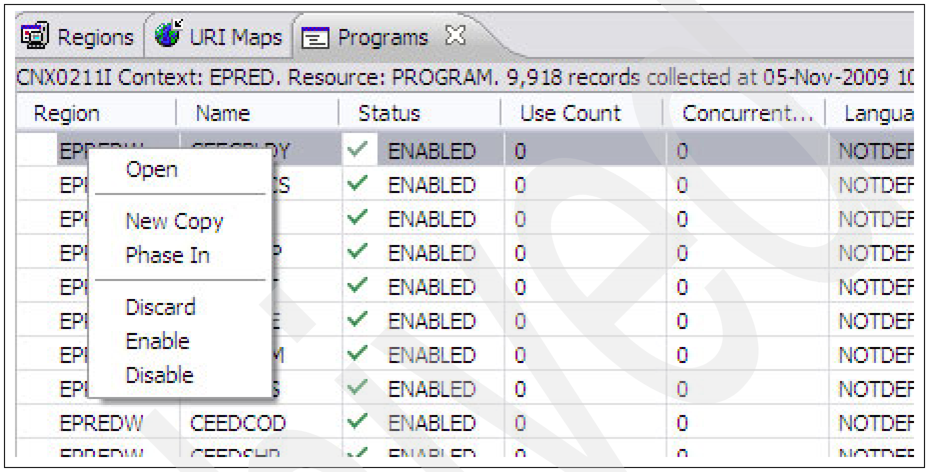
The plug-in implements a pop-up that can be opened from any CICS Explorer resource or definition view.

In this chapter we describe:

- A specification for the memo plug-in
- The code created to implement the plug-in
- How to use the plug-in

8.1 The specification for your sticky note plug-in

Start your CICS Explorer, and connect to the target CICS WUI region, then left-click the Operations tab in the action bar, and select any of the resources. You get a view like that shown in Figure 8-1, showing all the programs installed in the CICSPlex scope within which you are working.



CNX0211I Context: EPRED. Resource: PROGRAM. 9,918 records collected at 05-Nov-2009 1(

Region	Name	Status	Use Count	Concurrent...	Langua
EP	DY	✓ ENABLED	0	0	NOTDEF
EPI	S	✓ ENABLED	0	0	NOTDEF
EPI		✓ ENABLED	0	0	NOTDEF
EPI	P	✓ ENABLED	0	0	NOTDEF
EPI	T	✓ ENABLED	0	0	NOTDEF
EPI	E	✓ ENABLED	0	0	NOTDEF
EPI	M	✓ ENABLED	0	0	NOTDEF
EPI	S	✓ ENABLED	0	0	NOTDEF
EPREDW	CEEDCOD	✓ ENABLED	0	0	NOTDEF

Figure 8-1 CICS Explorer operations view

If you right-click one of the programs, CICS Explorer presents you with a pop-up menu offering a number of actions that you can perform on that program (for example, **Disable**, **Enable**, or **New Copy**).

This plug-in adds a new action, Add Sticky Notes, to the pop-up menu presented when you right-click any CICS Explorer resource, allowing you to associate a piece of text with that resource.

The plug-in allows you to view and manage the sticky notes that have been created within your CICS Explorer.

In this example you create an Eclipse JFace viewer to display the sticky notes that have been created. you also create a pop-up to allow users to create new sticky notes.

We are working with three main concepts:

- Viewer

  The user interface (the tableviewer in this example)
- Content provider

  Provides the model objects (sticky notes) to the viewer
- Label provider

  Defines how the sticky notes are displayed

This project consists of three main components:

- A model

  Represents the data for the plug-in
- A view

  Displays the data
- A pop-up menu

  Allows the view to be displayed

We begin by creating a blank plug-in and then add the necessary extension points, packages, classes, and methods as we go along.

8.2 Overview of components

The diagram in Figure 8-2 gives an overview of the components created and modified to produce this plug-in. The white boxes are the Java packages created either by the Eclipse wizards or by you. The colored boxes are the different classes either modified or created for this project.

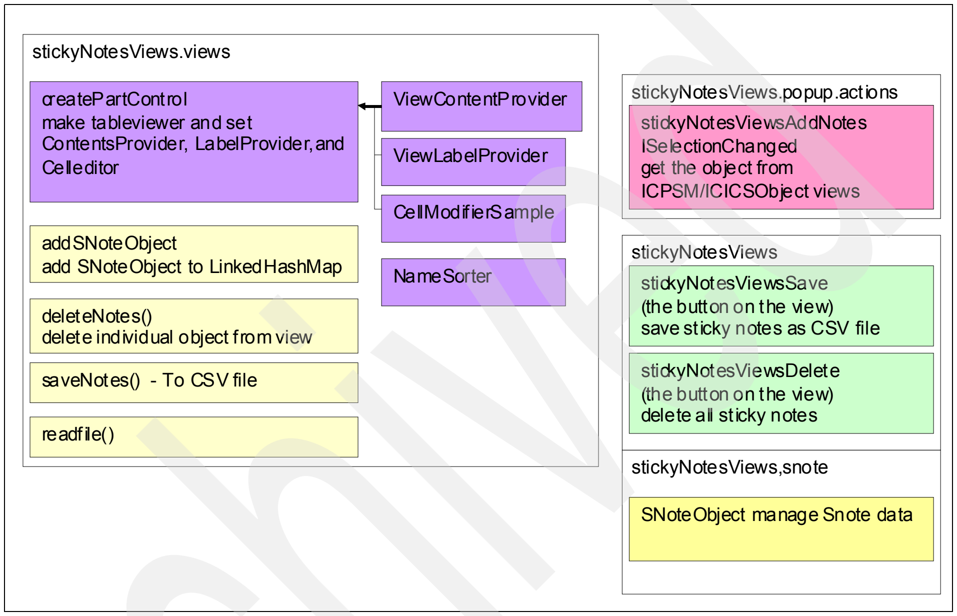


Figure 8-2 Overview of sticky notes Packages

The `createPartControl()`, `ViewContentProvider()`, `ViewLabelProvider()` and `CellModifierSample()` methods are Eclipse-plug-in extension methods that have been modified with custom code for our purposes. The `addSnoteObject()` `saveNotes()`, and `deleteNotes()` methods are methods we added to the StickyNotesView package generated by Eclipse for our new view.

`stickyNotesViews.snote()` is a new standalone object containing the code which manages the internals of the sticky notes object.

The stickyNotesViews package (which in retrospect we should have named stickyNotesViewsActions) contains the actions associated with our sticky notes view. You can click actions in the toolbar of the sticky notes views to save or delete the sticky notes (`StickyNotesViewsSave()` and `StickyNotesViewsDelete()`).

The stickyNotesViews.pop-up.actions package is generated by Eclipse tooling to contain the code associated with the actions you can perform when you right-click a CICS object and select sticky notes. You can add a note through the pop-up menu (`stickyNotesViewAddNotes()`).

8.3 Using a wizard to create your new plug-in

Start your Eclipse development environment, and select **File** → **New** →**Project**. Select **Plug-in Project**, then click **Next**. The first page of the wizard displays. Enter the project name as `StickyNotes` and click **Next**.

In the second page of the wizard, set the default properties. Click **Next**.

Note: We are going to create a blank plug-in without using a template. However, if you want to create this plug-in using a template as in prior examples, on the third page of the wizard, select **Custom plug-in wizard** and click **Next.** In the Template Selection page, select **Popup Menu** and **View** and click **Next**. You need to add appropriate names for the view and popup settings. See 5.4.1, "Specification of new textbox and button" on page 125 for naming conventions for this example.

In the third page of the wizard, clear the **Create a plug-in using one of the templates** check box and the Activator class, then click **Finish**. Your plug-in project has been created with the default manifest files and no activator class. Because you did not use a template, no view or pop-up class was generated. You create this class in a later step.

You are now ready to create the model structure for our project.

8.4 Creating the model

This project uses a single class that represents the data model in our design. The `SNoteObject` class is a simple object representing a sticky note component with getters and setters for the following properties:

```
private String SnoteName;
private String SnoteType;
private String SNoteComment;
```

Within your project, create the package `stickynotes.snote` (right-click the project, select **New** → **Package**). Create the `SNoteObject` class in this package, and copy in the code shown in Example 8-1.

Example 8-1 SNoteObject

```
package stickyNotesViews.snote;

public class SNoteObject{
      String SNoteName;
      String SNoteType;
      String SNoteComment;
      public SNoteObject(String snotename,String snotetype,String
snotecomment){
         SNoteName = snotename;
         SNoteType = snotetype;
         SNoteComment = snotecomment;
      }
      public String getSNoteName(){return SNoteName;}
      public String getSNoteType(){return SNoteType;}
      public String getSNoteComment(){return SNoteComment;}

      public String setSNoteName(String string){
         SNoteName = string;
         return null;}
      public String setSNoteType(String string){
         SNoteType = string;
         return null;}
      public String setSNoteComment(String string){
         SNoteComment = string;
         return null;}

   }
```

8.5 Creating the pop-up

Now that the data model structure is complete for your sticky notes object, you can create the pop-up which is going to allow users to create a sticky note. Because you are working with a blank plug-in, you need to add the extension points and import any packages required for this plug-in.

Note: If you used a template in the earlier step, some of these extension points might already be added to your plug-in project.

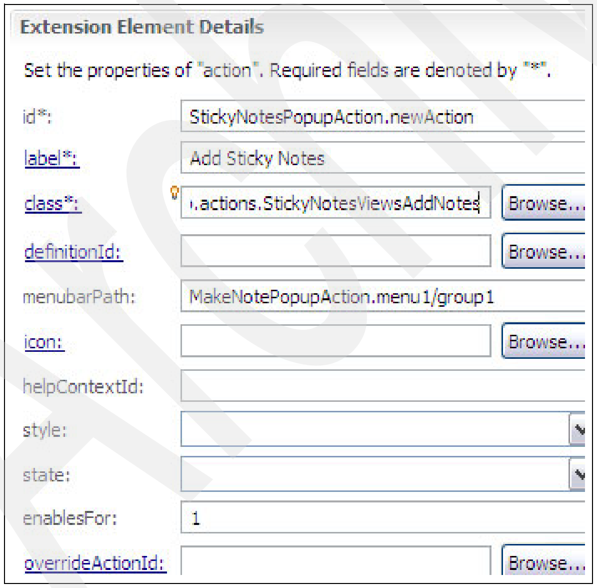
8.5.1 Adding extension points and packages for pop-up

You need to add an extension for the pop-up menu extension point. Open the manifest editor for your StickyNotes project by double-clicking the MANIFEST.MF object. Click **Add** and select `org.eclipse.ui.pop-upMenus` under **Extension Points**. Select pop-up Menu under **Available templates** and click **Next**.

As you are going to make sticky notes available to be used with any CICS resource, you need to associate your new function with the generic CICS Explorer object representing a CICS resource or definition. On the next page, add this pop-up menu to the ICICSObject resource by specifying the following values:

- Target Object's Class: `com.ibm.cics.model.ICICSObject`
- Submenu Name: `sticky notes`
- Action Label: `Add Sticky Notes`
- Action Class: `StickyNotesViewsAddNotes`

Click **Finish**. You might get a pop-up window asking to save the changes made to the plug-in. Click **Yes.** When you have saved your new `AddStickyNotes` action, click it. See Figure 8-3.


Extension Element Details

Set the properties of "action". Required fields are denoted by "*".

id*:	StickyNotesPopupAction.newAction
label*:	Add Sticky Notes
class*:	ı.actions.StickyNotesViewsAddNotes
definitionId:	
menubarPath:	MakeNotePopupAction.menu1/group1
icon:	
helpContextId:	
style:	
state:	
enablesFor:	1
overrideActionId:	

Figure 8-3 Add Sticky Notes Extension details

Add global variables to the class as shown in Example 8-2. (The imports generated by Eclipse have been suppressed in this figure.)

Example 8-2 .StickyNotes pop-up globals

```
package sticky.notes.popup.actions;

import org.eclipse.jface.action.IAction;
..
import sticky.notes.snote.SNoteObject;

import com.ibm.cics.model.ICICSObject;

public class StickyNotesViewsAddNotes implements IObjectActionDelegate
{

   private Shell shell;
   public ICICSObject icicsobject;
   private String notetext;
```

You need to update the `run()` method of the `StickyNotesViewsAddNotes_object`. This is the code invoked when the user clicks this action in the pop-up menu. When the user selects **Add Sticky Note**, the updated `run()` method (Example 8-3) invokes the `addSNoteobject()` method of the sticky notes view to update the list object containing the list of sticky notes.

Example 8-3 run() method of StickyNotesViewsAddNotes

```
public void run(IAction action) {
   String dialogTitle =
icicsobject.getCICSType().getInterfaceType().getName();

   dialogTitle = dialogTitle.substring(dialogTitle.lastIndexOf(".") +
2);
   String dialogMessage = "Resource name : " + icicsobject.getName();
   String initialValue = "input text notes";

   InputDialog inputdialog = new InputDialog(shell, dialogTitle,
dialogMessage, initialValue,
   new IInputValidator() {
      @Override
   public String isValid(String newText) {
   if (newText.length() == 0) {
         return "input text !";
      } else {
         notetext = newText;
```

```
            return null;
         }
         }
      });

      if (inputdialog.open() == Dialog.OK) {
         IWorkbench workbench =  PlatformUI.getWorkbench();
         IWorkbenchWindow window = workbench.getActiveWorkbenchWindow();
         IWorkbenchPage page = window.getActivePage();

      try {
         ViewPart sampleView = (ViewPart) page
      .findView("stickyNotesViews.views.StickyNotesView");

      if (sampleView == null) {
         sampleView = (ViewPart) page
               .showView("stickyNotesViews.views.StickyNotesView");

      }
      SNoteObject snoteobject = new SNoteObject(
            icicsobject.getName(), dialogTitle, notetext);
      ((StickyNotesView) sampleView).addSNoteObject(snoteobject,
            sampleView);
   } catch (PartInitException e) {
      e.printStackTrace();
   }

   }

   }
```

At this point, you might have an error flagged in Eclipse because you have not yet defined your sticky notes view.

You also need to modify the `Selectionchanged()` method so that when the user clicks a CICS resource to add a sticky note, information about that resource is saved. See Example 8-4.

Example 8-4 SelectionChanged() method of tickyNotesViewsAddNotes

```
      public void selectionChanged(IAction action, ISelection selection) {
         icicsobject = (ICICSObject) ((StructuredSelection) selection)
               .getFirstElement();

      }
```

Next, you have to write the code to manage the new sticky notes view.

8.5.2 Adding extension points and packages for view

To add the extension points, open the manifest file for your plug-in, click the Extensions tab, and perform the following steps

1. Click **All Extensions** → **Add**.
2. In the Extension Point Selection panel, select **Extension Points** → **org.eclipse.ui.views**, and **Available templates** → **Sample View**. Click **Next.**
3. Specify the following values
   - View Class Name: `StickyNotesView`
   - View Name: `Sticky Notes View`
   - View Category Name: `Sticky Notes`
4. You are using a table viewer for this view so ensure it is selected. Click **Finish** to add this extension point and create the plug-in class.

Notice that three extensions have been added to the Extensions tab for our plug-in: one for the eclipse view plug-in, one for the eclipse perspective extension plug-in, and one for the eclipse help context plug-in. You have added the view template to your project and Eclipse has created a StickyNotesView.java file for you. You are ready to insert the code to implement the sticky notes view. In Example 8-5 you can see the global variables that you need to add.

- Initialize the table columns for the table in our view
- Create a hashmap `item` to manage your sticky notes objects.
- Use the Java Memento object to create a sticky note

Example 8-5 Global variables for StickyNotesView.java

```
public class StickyNotesView extends ViewPart {
   /** The ID of the view as specified by the extension. **/
   public static final String ID =
"sticky_notes.views.StickyNotesView";
   private static final String[] columnNames = { "Name", "Type",
"Comment" };
   private static final int[] columnWidths = { 100, 120, 300 };
   private TableViewer viewer;
   private Action action1;
   private TableColumn column;
   final String DIRECTORY_PATH =
ResourcesPlugin.getWorkspace().getRoot().getLocation().toOSString();
   final String TEMP_CSVFILE = "stickyNotesTemp";
   final LinkedHashMap<SNoteObject, ViewPart> items = new
LinkedHashMap<SNoteObject,
ViewPart>();
```

In Example 8-6 you see the changes you need to make to the ViewContentProvider class in the template generated by Eclipse. Change the getElements() method of this class to return an array of sticky notes objects from the entries in the items hash table.

Example 8-6 Changed code to return array of sticky notes objects

```
   class ViewContentProvider implements IStructuredContentProvider {
      public void inputChanged(Viewer v, Object oldInput, Object
newInput) {}
      public void dispose() {}
      public Object[] getElements(Object parent) {
         return items.keySet().toArray();}
}
```

In Example 8-7 you see the changes to the ViewLabelProvider class. It is called to populate the table that is being created. Depending on the column that is being called to populate, it returns the appropriate sticky note name, type or, text.

Example 8-7 Changes to ViewLabelProvider

```
class ViewLabelProvider extends LabelProvider implements
         ITableLabelProvider {
      public String getColumnText(Object obj, int index) {
         SNoteObject snoteobject = (SNoteObject) obj;
         switch (index) {
         case 0:return snoteobject.getSNoteName();
         case 1:return snoteobject.getSNoteType();
         case 2:return snoteobject.getSNoteComment();}
         return null;}
      public Image getColumnImage(Object obj, int index) {return null;}
      public Image getImage(Object obj) {
         return PlatformUI.getWorkbench().getSharedImages().getImage(
               ISharedImages.IMG_OBJ_ELEMENT);}}
```

Modify the CellModifierSample class as shown in Example 8-8, to use the SNoteObject.

Example 8-8 CellModifierSample

```
   public class CellModifierSample implements ICellModifier {
      private TableViewer viewer;
      public CellModifierSample(TableViewer viewer) {
         this.viewer = viewer;}
      @Override
      public boolean canModify(Object element, String property) {
         return true;}
      @Override
      public Object getValue(Object element, String propety) {
         SNoteObject snoteobject = (SNoteObject) element;
         Object snote = null;
         if (propety.equals("Comment")) {
            snote = snoteobject.getSNoteComment();}
         return snote;}
      @Override
      public void modify(Object element, String propety, Object value)
{
         TableItem tableItem = (TableItem) element;
         SNoteObject snoteobject = (SNoteObject) tableItem.getData();
         if (propety.equals("Comment")) {
            snoteobject.setSNoteComment((String) value);}
         viewer.update(snoteobject, null);}}
```

It is the `createPartControl()` method, as shown in Example 8-9 that builds the view. In our case, it builds a view suited to our sticky notes display. It invokes a series of methods to set up the action bars, context help, and so forth. Because sticky notes are saved across restarts of Eclipse, this method invokes the `readfile()` method to retrieve the sticky notes saved the last time Eclipse was run.

Example 8-9 createPartControl Method

```
public void createPartControl(Composite parent) {
      viewer = new TableViewer(parent, SWT.MULTI | SWT.H_SCROLL
            | SWT.V_SCROLL | SWT.FULL_SELECTION);
      viewer.setContentProvider(new ViewContentProvider());
      viewer.setLabelProvider(new ViewLabelProvider());
      viewer.setInput(getViewSite());
      // Create the help context id for the viewer's control

PlatformUI.getWorkbench().getHelpSystem().setHelp(viewer.getControl(),
            "Sticky_Notes.viewer");
      makeActions();
      hookContextMenu();
      contributeToActionBars();
      Table table = viewer.getTable();
      table.setHeaderVisible(true);
      table.setLinesVisible(true);
      // Set the column headers
      for (int i = 0; i < columnNames.length; i++) {
         column = new TableColumn(table, SWT.LEFT);
```

```
            column.setText(columnNames[i]);
            column.setWidth(columnWidths[i]);}
      CellEditor[] celleditor = new CellEditor[] { new
TextCellEditor(table),
            new TextCellEditor(table), new TextCellEditor(table) };
      viewer.setColumnProperties(columnNames);
      viewer.setCellEditors(celleditor);
      viewer.setCellModifier(new CellModifierSample(viewer));
      readFile(TEMP_CSVFILE);}
```

We are going to add an action which allows the user to delete the selected sticky note from the list. Example 8-10 shows the `makeActions()` method that implements this action. You can see that it invokes the `remove()` method of the items object to remove the selected object. It then refreshes the view, with the object removed.

Example 8-10 makeActions() method of StickyNotesView

```
private void makeActions() {
   action1 = new Action() {
      public void run() {
         final StructuredSelection selection = (StructuredSelection)
viewer
               .getSelection();
         for (Object obj : selection.toArray())
               items.remove(obj);
         viewer.refresh();
         try {saveNotes(TEMP_CSVFILE);
         } catch (UnsupportedEncodingException e) {
            e.printStackTrace();}}
   };
   action1.setText("Delete Sticky Note");
   action1.setToolTipText("Remove this Sticky Note");

action1.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages()
         .getImageDescriptor(ISharedImages.IMG_OBJS_INFO_TSK));
      ;}
```

In Example 8-11 on page 260, we see the `addSnoteObject()` method that is invoked to add a new StickyNote into the table. We then refresh the view. This code is invoked by the pop-up action code to add a sticky note to our list, and then refreshes the view to pick up the new sticky note. You can see that each time a sticky note is added to the list, we save the entire list to a temporary .csv file.

Example 8-11 addSNoteObject() method

```
public void addSNoteObject(SNoteObject snoteobject, ViewPart 
sampleView) {
      items.put(snoteobject, sampleView);
      viewer.refresh();
      try {saveNotes(TEMP_CSVFILE);
      } catch (UnsupportedEncodingException e) {}}
```

In Example 8-12 we see the deleteNotes() method of our StickyNotes view. When invoked, this method deletes all the StickyNotes in the list.

Example 8-12 deleteNotes() method

```
   public void deleteNotes() {
      items.clear();
      viewer.refresh();
      try {saveNotes(TEMP_CSVFILE);
      } catch (UnsupportedEncodingException e) {}}
```

The code that manages the saving of the sticky notes to a .csv file is in the saveNotes() method is shown in Example 8-13.

Example 8-13 saveNotes() method

```
public void saveNotes(String fileName) throws 
UnsupportedEncodingException {
   String filePath = DIRECTORY_PATH + "\\" + fileName + ".csv";
   try {FileOutputStream fos = new FileOutputStream(filePath, false);
      OutputStreamWriter osw = new OutputStreamWriter(fos, "UTF-8");
      BufferedWriter bw = new BufferedWriter(osw);
      Table table = viewer.getTable();
      for (int i = 0; i < table.getItemCount(); i++) {
         TableItem items = table.getItem(i);
         bw.write(items.getText(0) + "," + items.getText(1) + ","
               + items.getText(2));
         bw.newLine();}
      bw.flush();
      bw.close();
      osw.close();
      fos.close();
   } catch (FileNotFoundException e) {e.printStackTrace();
   } catch (UnsupportedEncodingException e) {e.printStackTrace();
   } catch (IOException e) {e.printStackTrace();}}
```

The routine that reads saved sticky notes from the .csv file can be found in Example 8-14. The .csv file is created within the Eclipse workspace in the home directory of the Eclipse being used, with the name `TEMP_CSVFILE.csv`. If Eclipse is closed down and restarted, the data is retrieved by the `createPartControl` method when the sticky notes view is opened.

Example 8-14 readFile() method

```
public void readFile(String file_name) {
   String filePath = DIRECTORY_PATH + "\\" + TEMP_CSVFILE + ".csv";
   try {File file = new File(filePath);
      BufferedReader br = new BufferedReader(new FileReader(file));
      String str;
      while ((str = br.readLine()) != null) {
         String[] strArray = str.split(",");
         SNoteObject snoteobject = new SNoteObject(strArray[0],
               strArray[1], strArray[2]);
         items.put(snoteobject, null);
         viewer.refresh();}
      br.close();
   } catch (FileNotFoundException e) {
   } catch (IOException e) {}}
```

Although we have added code to the view to handle the saving and deletion of our sticky notes, we have not yet added extensions to the view to create the actions on the toolbar that you can click to perform the save and the delete.

8.5.3 Adding extension points and code for Delete View Action

We need to create the extensions and code that deletes a sticky notes list, or save a sticky notes list to a .csv file so that it can be saved across starts of CICS Explorer.

If it is not already open, double-click the MANIFEST.MF object in the package explorer, and select the Extensions tab in the view. We are going to add an exention to the org.eclipse.ui.vewActions plug-in. Click **Add**, and then select the org.eclipse.ui.vewActions plug-in. Now we can add our new View Actions.

Right-click the org.eclipse.ui.vewActions object in the extensions list, and select **New** → **viewContribution**. See Figure 8-4.

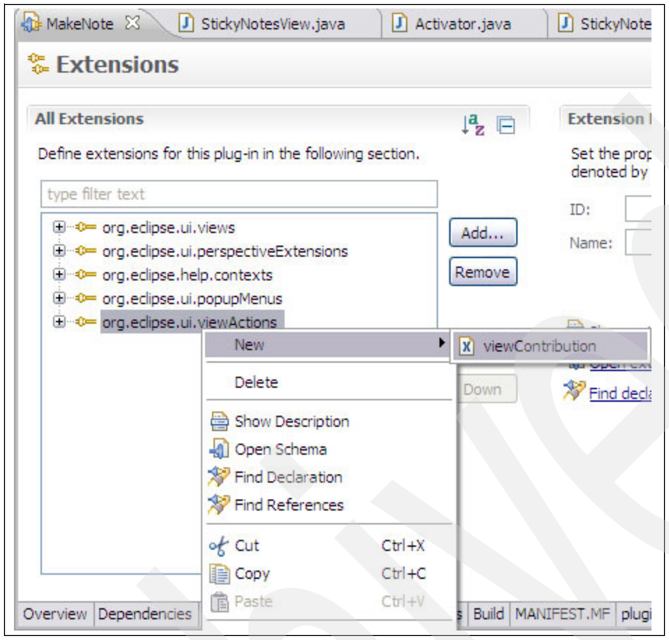


Figure 8-4 Add viewContirbution

Call the new viewContribution StickyNotes, and give it a target ID of `StickyNotesView`.

Right-click your new StickyNotesviewContribution1, and select **New**, then an action. Give this action an ID of `StickyNotes.action1`. You are prompted for the name of the class that is going to execute the new action. Click **class** to create the new class in the stikyNotesView package, and call it `stickyNotesViews.StickyNotesViewsDelete`. (We create this class and add it to the stickyNotesViews package shortly).

Next, we are going to add a graphic to the toolbar, which the user can click to delete all the sticky notes. Right-click the StickyNotesviewContribution1 again but this time select **New** → **Menu**. Give the new menu a name of `StickyNotes.menu1`. In the path box, enter `additions`, and in the icon box, add `icons/delete_config.gif`. (You need to have copied the red “x” graphic into the icons directory of StickyNotesView project before you try to run the plug-in.)

Now you are ready to put the delete code into the new class file you have created. See Example 8-15.

This new class identifies the current view, and issues the `deleteNotes()` call to delete all sticky notes assciated with it.

Example 8-15 StickyNotesViewDelete class

```
package stickyNotesViews;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IViewActionDelegate;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PlatformUI;

import stickyNotesViews.views.StickyNotesView;

public class StickyNotesViewsDelete implements IViewActionDelegate {

   private Shell shell;
   @Override
   public void init(IViewPart view) {}
   @Override
   public void run(IAction action) {
      if (MessageDialog
            .openConfirm(shell, "Comfirm Delete",
                  "Are you sure you want to permanently delete all
sticky notes?") == true) {
         IWorkbench workbench = PlatformUI.getWorkbench();
         IWorkbenchWindow window = workbench.getActiveWorkbenchWindow();
         IWorkbenchPage page = window.getActivePage();
         IViewPart sampleView = (IViewPart) page
               .findView("stickyNotesViews.views.StickyNotesView");
         ((StickyNotesView) sampleView).deleteNotes();
      }
   }
   @Override
   public void selectionChanged(IAction action, ISelection selection) {}
}
```

8.5.4 Adding extension points and code for Save View Action

We need to create the extensions and code that saves a sticky notes list to a .csv file so that it can be saved across starts of CICS Explorer.

If it is not already open, double-click the MANIFEST.MF object in the package explorer, and select the Extensions tab in the view. We are going to add an exention to the org.eclipse.ui.vewActions plug-in. Click **Add**, and select the org.eclipse.ui.vewActions plug-in. Now we can add our new View Actions.

Right-click the org.eclipse.ui.vewActions object in the extensions list, and select **New** → **viewContribution** (Ssee Figure 8-4 on page 262).

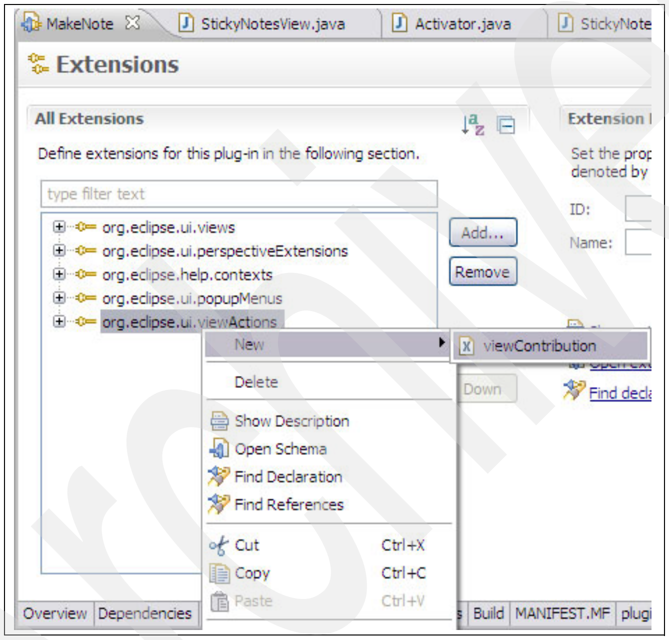


Figure 8-5 Add viewContribution

Call the new viewContribution StickyNotes, and give it a target ID of `StickyNotesView`.

Right-click your new StickyNotesviewContribution2, and select **New** → **action**. Give this action an ID of `StickyNotes.action2`. You are prompted for the name of the class that is going to execute the new action. Click **class** to create the new class in the stikyNotesView package, and call it `stickyNotesViews.StickyNotesViewsSave`. (We create this class and add it to the `stickyNotesViews` package shortly.)

Next, we are going to add a graphic to the toolbar, which the user can click to delete all the sticky notes. Rright-click the StickyNotesviewContribution2 again, but this time select **New** → **Menu**. Give the new menu a name of `StickyNotes.menu2`. In the path box, enter `additions`, and in the icon box, add `icons/save.gif` (you need to have copied the diskette save graphic into the icons directory of StickyNotesView project before you try to run the plug-in).

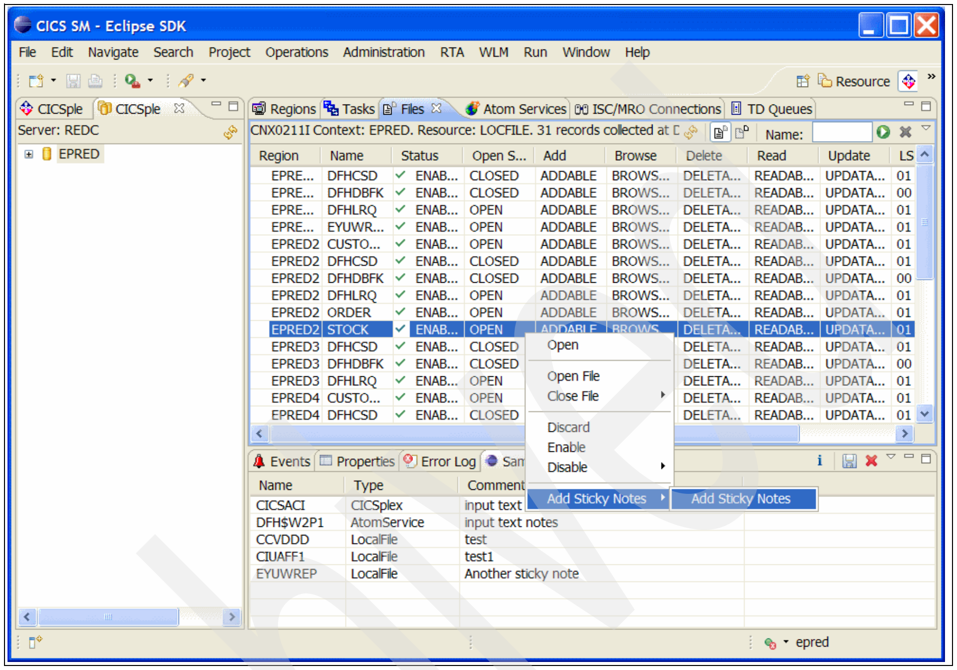
You are now ready to put the save code into the new class file you have just created. See Example 8-16.

This new class identifies the current view, and then issues the `saveNotes()` call to save all sticky notes associated with it to a .csv file in the Eclipse workspace.

Example 8-16 StickyNotesViewSave class

```
package stickyNotesViews;

import java.io.UnsupportedEncodingException;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.jface.dialogs.IInputValidator;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IViewActionDelegate;
import org.eclipse.ui.IViewPart;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;

import stickyNotesViews.views.StickyNotesView;

public class StickyNotesViewsSave implements IViewActionDelegate {

   private Shell shell;
   private String notetext;
   @Override
   public void init(IViewPart view) {}
   @Override
   public void run(IAction action) {
      String dialogTitle = "Save Sticky Notes";
      String dialogMessage = "input CSV FILE name (Extention(.csv) is
automatically added.) ";
      String initialValue = "";
      InputDialog inputdialog = new InputDialog(shell, dialogTitle,
            dialogMessage, initialValue, new IInputValidator() {
               @Override
               public String isValid(String newText) {
                  if (newText.length() == 0) {
                     return "input filename.";
                  } else {
                     notetext = newText;
                     return null;
                  }
               }
            });

      if (inputdialog.open() == Dialog.OK) {
         IWorkbench workbench = PlatformUI.getWorkbench();
         IWorkbenchWindow window =
workbench.getActiveWorkbenchWindow();
         IWorkbenchPage page = window.getActivePage();
         try {
            ViewPart sampleView = (ViewPart) page
                  .findView("stickyNotesViews.views.StickyNotesView");
            if (sampleView == null) {
               sampleView = (ViewPart) page

.showView("stickyNotesViews.views.StickyNotesView");
            }
            ((StickyNotesView) sampleView).saveNotes(notetext);


         } catch (PartInitException e) {e.printStackTrace();
         } catch (UnsupportedEncodingException e) {
e.printStackTrace();}
      }

   }
   @Override
   public void selectionChanged(IAction action, ISelection selection)
{}
}
```

If you have copied the images into the icon directory, we are now ready to test our new plug-in. Right-click the the resource to which you want to add a sticky note. For this test we are using a file. See Figure 8-6.



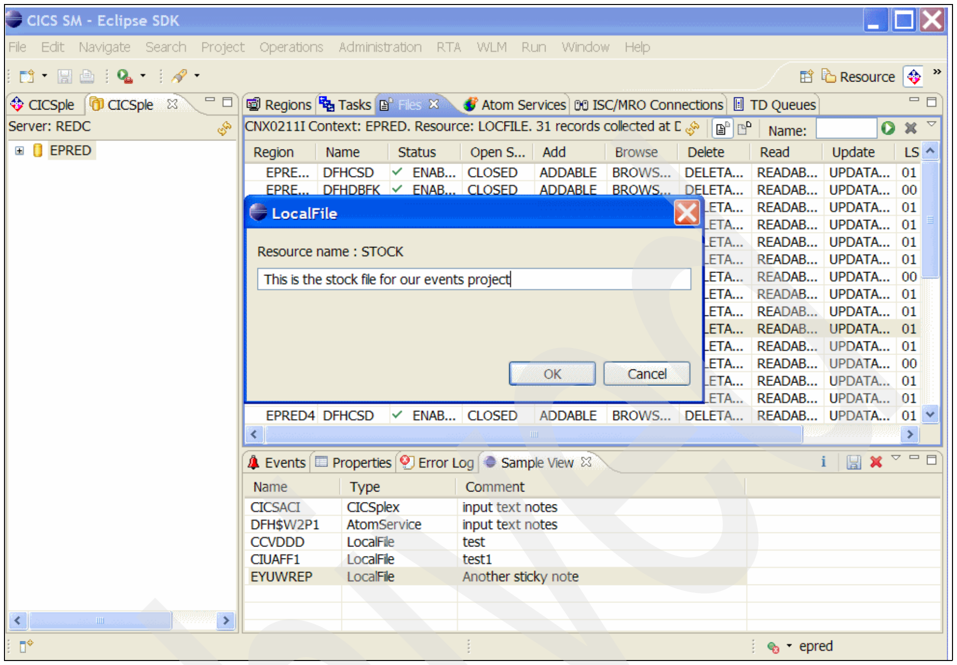
CNX0211I Context: EPRED. Resource: LOCFILE. 31 records collected at D

Region	Name	Status	Open S...	Add	Browse	Delete	Read	Update	LS
EPRE...	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRE...	DFHLRQ	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	EYUWR...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	CUSTO...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRED2	DFHLRQ	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	ORDER	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	STOCK	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED3	DFHCSD	✓ ENAB...	CLOSED			DELETA...	READAB...	UPDATA...	01
EPRED3	DFHDBFK	✓ ENAB...	CLOSED			DELETA...	READAB...	UPDATA...	00
EPRED3	DFHLRQ	✓ ENAB...	OPEN			DELETA...	READAB...	UPDATA...	01
EPRED4	CUSTO...	✓ ENAB...	OPEN			DELETA...	READAB...	UPDATA...	01
EPRED4	DFHCSD	✓ ENAB...	CLOSED			DELETA...	READAB...	UPDATA...	01

Name	Type	Comment
CICSACI	CICSplex	input text
DFH$W2P1	AtomService	input text notes
CCVDDD	LocalFile	test
CIUAFF1	LocalFile	test1
EYUWREP	LocalFile	Another sticky note

Figure 8-6 Add sticky note to file

This opens a box to add the description for the sticky note. See Figure 8-7.


CNX0211I Context: EPRED. Resource: LOCFILE. 31 records collected at D

Region	Name	Status	Open S...	Add	Browse	Delete	Read	Update	LS
EPRE...	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	00
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	00
						ETA...	READAB...	UPDATA...	01
						ETA...	READAB...	UPDATA...	01
EPRED4	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01

LocalFile

Resource name : STOCK

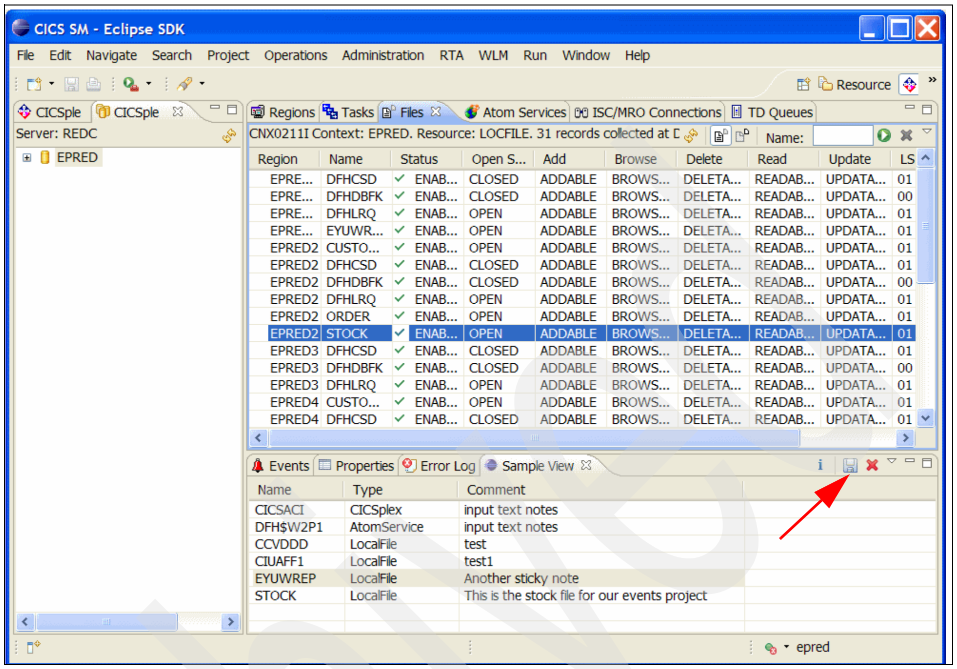
This is the stock file for our events project

Name	Type	Comment
CICSACI	CICSplex	input text notes
DFH$W2P1	AtomService	input text notes
CCVDDD	LocalFile	test
CIUAFF1	LocalFile	test1
EYUWREP	LocalFile	Another sticky note

CICS SM - Eclipse SDK
File Edit Navigate Search Project Operations Administration RTA WLM Run Window Help
Resource
CICSple CICSple
Server: REDC
EPRED
Regions Tasks Files Atom Services ISC/MRO Connections TD Queues
Name:
OK Cancel
Events Properties Error Log Sample View
epred

Figure 8-7 Sticky note description

You see the note display in the sample view shown in Figure 8-8.


CICS SM - Eclipse SDK
File Edit Navigate Search Project Operations Administration RTA WLM Run Window Help
Resource
CICSple CICSple
Server: REDC
EPRED
Regions Tasks Files Atom Services ISC/MRO Connections TD Queues
Name:
Events Properties Error Log Sample View
epred

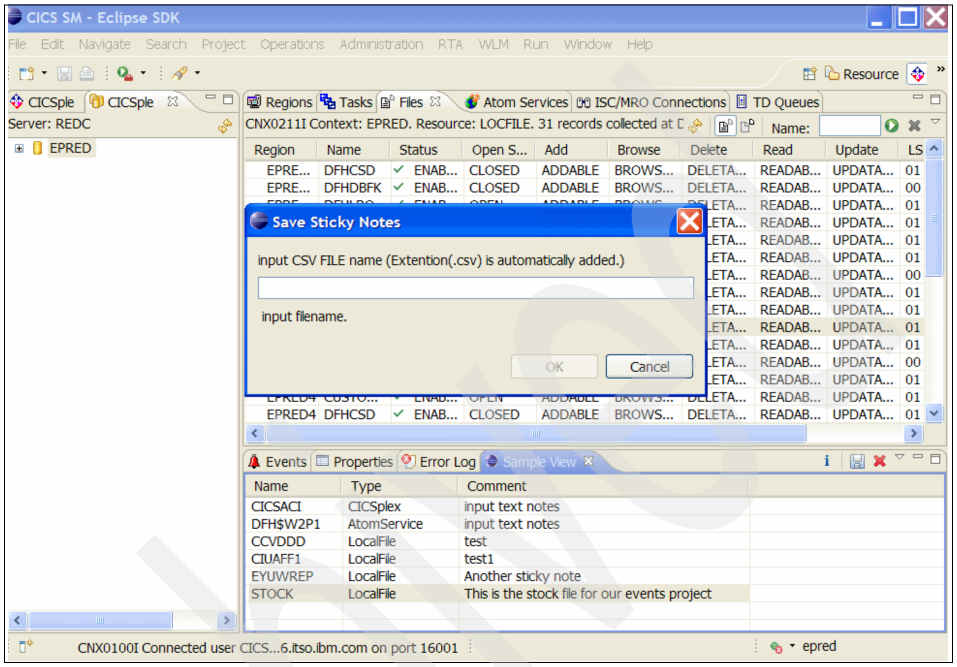
CNX0211I Context: EPRED. Resource: LOCFILE. 31 records collected at D

Region	Name	Status	Open S...	Add	Browse	Delete	Read	Update	LS
EPRE...	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRE...	DFHLRQ	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	EYUWR...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	CUSTO...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRED2	DFHLRQ	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	ORDER	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED2	STOCK	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED3	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED3	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRED3	DFHLRQ	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED4	CUSTO...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED4	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01

Name	Type	Comment
CICSACI	CICSplex	input text notes
DFH$W2P1	AtomService	input text notes
CCVDDD	LocalFile	test
CIUAFF1	LocalFile	test1
EYUWREP	LocalFile	Another sticky note
STOCK	LocalFile	This is the stock file for our events project

Figure 8-8 New sticky note added

Save this note to a CSV file. Click the disk on the window (denoted by the red arrow in the bottom right of the figure) as shown in Figure 8-8 on page 269. When clicked, you get the text box shown in Figure 8-9.


CICS SM - Eclipse SDK
File Edit Navigate Search Project Operations Administration RTA WLM Run Window Help
Resource
CICSple CICSple
Server: REDC
EPRED
Regions Tasks Files Atom Services ISC/MRO Connections TD Queues
Name:
Events Properties Error Log Sample View
Save Sticky Notes
OK Cancel
epred

CNX0211I Context: EPRED. Resource: LOCFILE. 31 records collected at D

Region	Name	Status	Open S...	Add	Browse	Delete	Read	Update	LS
EPRE...	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRED4	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01

input CSV FILE name (Extention(.csv) is automatically added.)

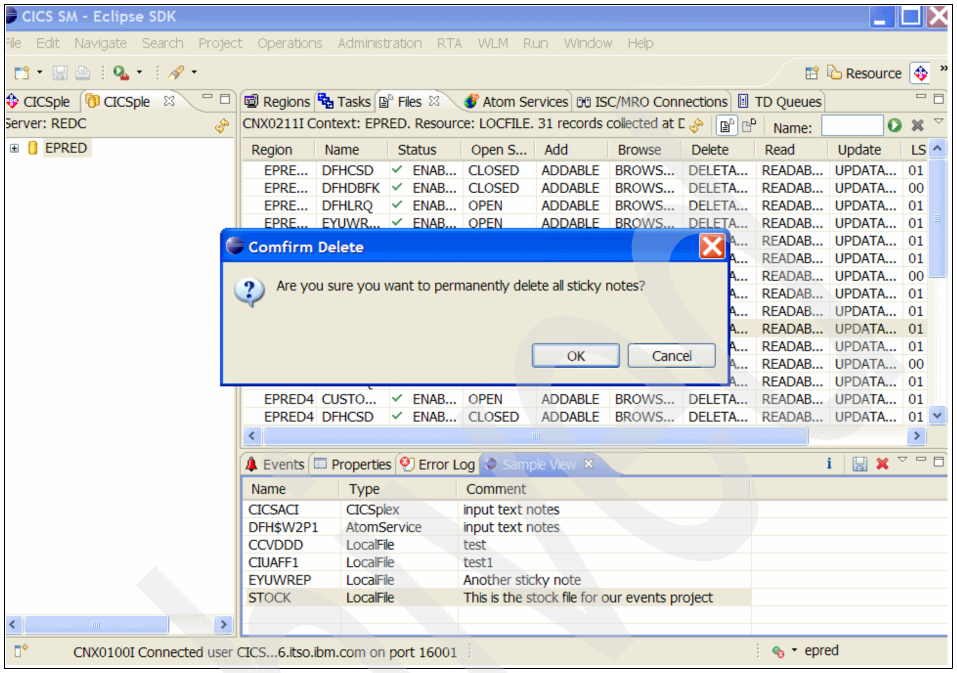
input filename.

Name	Type	Comment
CICSACI	CICSplex	input text notes
DFH$W2P1	AtomService	input text notes
CCVDDD	LocalFile	test
CIUAFF1	LocalFile	test1
EYUWREP	LocalFile	Another sticky note
STOCK	LocalFile	This is the stock file for our events project

CNX0100I Connected user CICS...6.itso.ibm.com on port 16001

Figure 8-9 Save Sticky note as CSV file

You can also delete the sticky if you want. To do so, click the red X next to the save sticky note to CSV file, you get the message shown in Figure 8-10.



CICS SM - Eclipse SDK

CNX0211I Context: EPRED. Resource: LOCFILE. 31 records collected at [

Region	Name	Status	Open S...	Add	Browse	Delete	Read	Update	LS
EPRE...	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	DFHDBFK	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	00
EPRE...	DFHLRQ	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRE...	EYUWR...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	00
						A...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	01
						A...	READAB...	UPDATA...	00
						A...	READAB...	UPDATA...	01
EPRED4	CUSTO...	✓ ENAB...	OPEN	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01
EPRED4	DFHCSD	✓ ENAB...	CLOSED	ADDABLE	BROWS...	DELETA...	READAB...	UPDATA...	01

Comfirm Delete

Are you sure you want to permanently delete all sticky notes?

OK Cancel

Name	Type	Comment
CICSACI	CICSplex	input text notes
DFH$W2P1	AtomService	input text notes
CCVDDD	LocalFile	test
CIUAFF1	LocalFile	test1
EYUWREP	LocalFile	Another sticky note
STOCK	LocalFile	This is the stock file for our events project

CNX0100I Connected user CICS...6.itso.ibm.com on port 16001

Figure 8-10 Delete sticky note

If you select **OK** the sticky note is deleted.

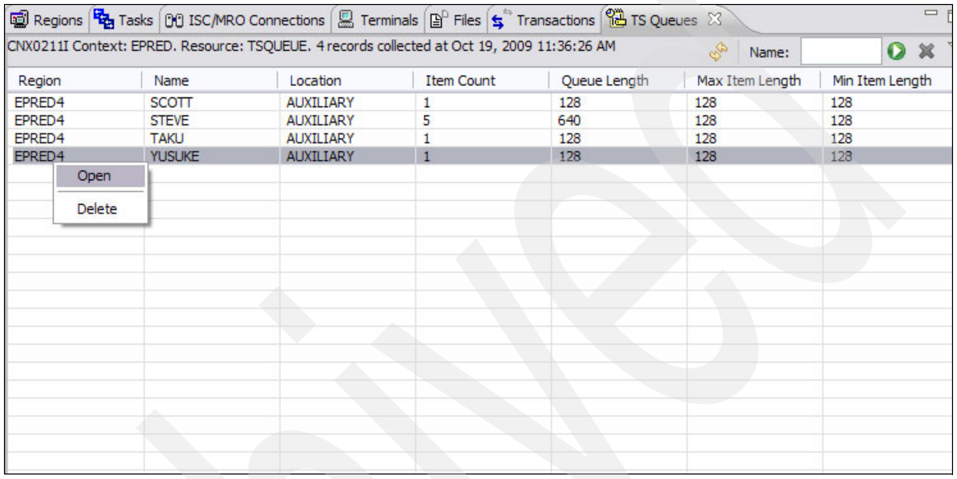
Implementing a CEBR view in CICS Explorer

In this chapter we look at how you can extend the functionality provided by the Operations view of the CICS Explorer, to implement a CEBR-like interface which, when you have right-clicked a TS queue, allows you to view the contents of that queue.

- The skill profile of the person who implemented this sample is a CICS Systems Programmer who knows only what is in Chapter 5 of this book (And a bit of Chapter 4), with no real experience with Java other than a bit of playing around with Java primer material and the samples you find earlier in this book.
- The author found that with the help of searching on the Web, and timely interventions from a colleague familiar with Java who was also working on this Redbooks publication, the author managed to implement the sample.
- The author adopted a top-down approach to this sample. The starting point was a new interface the author wanted to achieve using CICS Explorer. The author designed the user interface, and then looked at the user interface code the author had to write in Eclipse. The author then looked at the code required in our new plug-in to connect to CICS and retrieve the TS Queue. Finally, the author wrote the backend code CICS TS code to retrieve the TS queue records and return them to his new plug-in.

9.1 The specification for your CEBR view

If you start your CICS Explorer, and connect to the target CICS WUI region, then left-click the Operations tab in the action bar, and select **Queues** → **TS Queues**, you get a view like Figure 9-1, showing all the TS queues known in the CICSPlex scope within which you are working.



CNX0211I Context: EPRED. Resource: TSQUEUE. 4 records collected at Oct 19, 2009 11:36:26 AM

Region	Name	Location	Item Count	Queue Length	Max Item Length	Min Item Length
EPRED4	SCOTT	AUXILIARY	1	128	128	128
EPRED4	STEVE	AUXILIARY	5	640	128	128
EPRED4	TAKU	AUXILIARY	1	128	128	128
EPRED4	YUSUKE	AUXILIARY	1	128	128	128

Figure 9-1 CICS Explorer view of TS queues

If you right-click one of the TS queues shown in the list, CICS Explorer presents you with two possible actions: **Open** or **Delete**. Wouldn't it be nice if there were ? To achieve a third option, **Browse**, which allows you to browse the contents of the queue , you are must create a new Eclipse plug-in. This plug-in has two Eclipse user interface objects:

- View

  This object displays the records in your TS queue

- Pop-up menu

  This object gives the option of displaying the new TS queue browse view

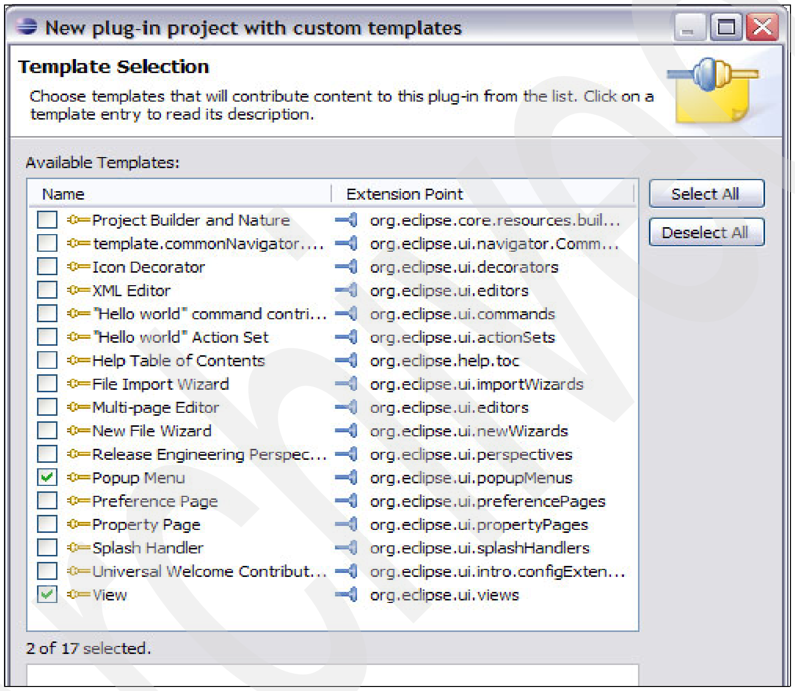
Use the wizard provided by Eclipse to create a template for your new plug-in. After you have created the template, add the code required to:

- Retrieve the name of the queue to be browsed.
- Build an HTTP request to invoke a CICS TS Cobol program that retrieves the contents of the requested TS queue and handle the HTTP response containing the contents of TS queue.
- Display the contents of the TS queue in a new CEBR view.

9.2 Using the wizard to create your new plug-in

Start your Eclipse development environment, and select **File** → **New** → **Project**. Select **Plug-in Project**, and click **Next**.

Name your new project `BrowseTSQueue` and click **Next**. You are prompted to select a template to use for this project (Figure 9-2). This time you are going to be more specific about what kind of plug-in you want. Select **Custom Plug-in wizard** and click **Next**.



Template Selection

Choose templates that will contribute content to this plug-in from the list. Click on a template entry to read its description.

Available Templates:

	Name	Extension Point
☐	Project Builder and Nature	org.eclipse.core.resources.buil...
☐	template.commonNavigator....	org.eclipse.ui.navigator.Comm...
☐	Icon Decorator	org.eclipse.ui.decorators
☐	XML Editor	org.eclipse.ui.editors
☐	"Hello world" command contri...	org.eclipse.ui.commands
☐	"Hello world" Action Set	org.eclipse.ui.actionSets
☐	Help Table of Contents	org.eclipse.help.toc
☐	File Import Wizard	org.eclipse.ui.importWizards
☐	Multi-page Editor	org.eclipse.ui.editors
☐	New File Wizard	org.eclipse.ui.newWizards
☐	Release Engineering Perspec...	org.eclipse.ui.perspectives
✓	Popup Menu	org.eclipse.ui.popupMenus
☐	Preference Page	org.eclipse.ui.preferencePages
☐	Property Page	org.eclipse.ui.propertyPages
☐	Splash Handler	org.eclipse.ui.splashHandlers
☐	Universal Welcome Contribut...	org.eclipse.ui.intro.configExten...
✓	View	org.eclipse.ui.views

2 of 17 selected.

Figure 9-2 Create custom plug-in template

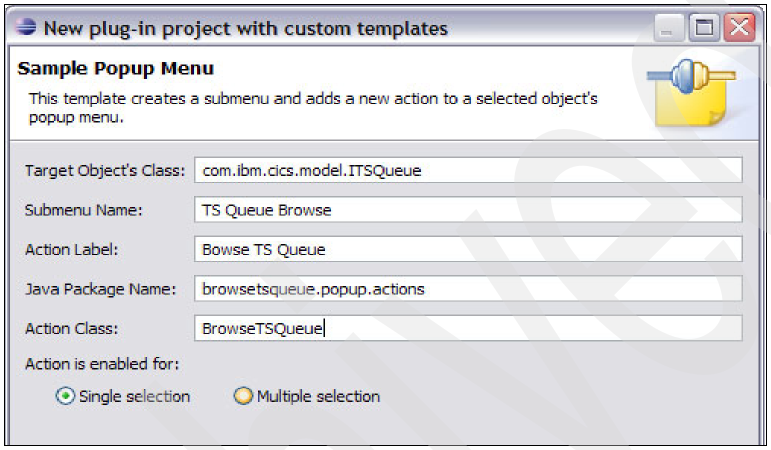
Your plug-in provides the following advantages:

- Manage a pop-up menu that allows a TS queue to be selected for browsing
- Create the view in which the contents of the TS queue are shown.

Check the **View** and **Pop-up Menu** boxes and click **Next**.

The wizard presents you with the sample pop-up menu to ask for more information about the pop-up menu. It needs to know the class that defines the object to be managed by your pop-up menu. In this case, the Target Object's class is the Explorer class used to manage TS queues: `com.ibm.cics.model.ITSQueue`.

Name your new class `BrowseTSQueue` and give suitable names for the submenu name and action label, as shown in Figure 9-3, and click **Next**.



Sample Popup Menu

This template creates a submenu and adds a new action to a selected object's popup menu.

Target Object's Class:	com.ibm.cics.model.ITSQueue
Submenu Name:	TS Queue Browse
Action Label:	Bowse TS Queue
Java Package Name:	browsetsqueue.popup.actions
Action Class:	BrowseTSQueue

Action is enabled for:

[✓ Single selection] Multiple selection

Figure 9-3 TS Queue Browse sample pop-up menu

The wizard presents you with the Main View Settings panel. Select appropriate names (it is suggested to specify what kind of class you are creating in the name, so call this Action class `BrowseTSQueueAction`), and click **Finish**.

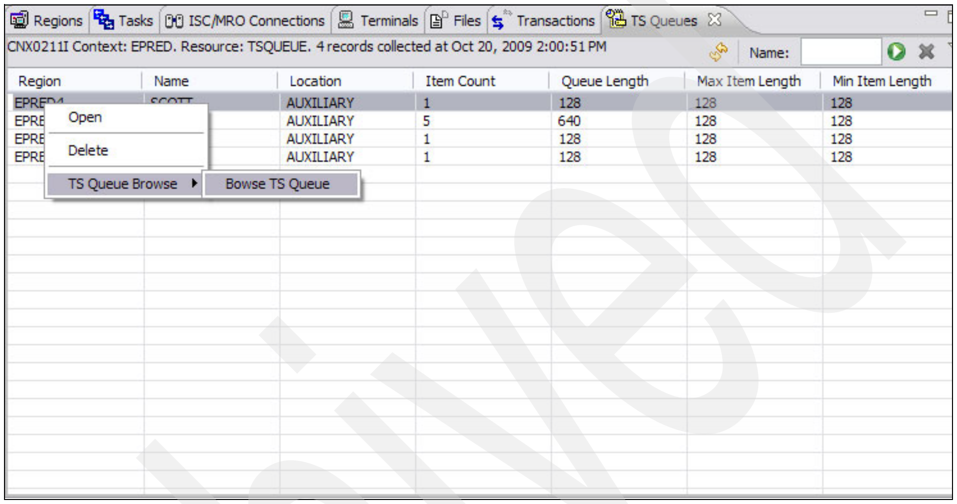
Note: On the subject of appropriate names, when you have more than one class in your plug-in (here we have an Action and a View), make sure the name you choose for your class uniquely identifies that class within your plug-in. Otherwise, you might have to manage two different classes with the same name that reference each other, which creates complications.

The wizard creates the plug-in and presents us with the overview editor view for our new plug-in `BrowseTSQueue`.

At this point, although you have not written any new code yet, run your `BrowseTSQueue` plug-in to see what it does.

In the Package Explorer view on the left side of the workspace, right-click the `BrowseTSQueue` package and select **Run as → Eclipse Application**.

A new instance of Eclipse starts with the new plug-in installed. If it has not already started with the CICSPlex SM perspective, click **Window** → **Show View** → **CICS Explorer**. Then choose your scope by selecting the CICSPlex or CICS region you want to look at in the left pane, and select **Operations** → **Queues** → **TS Queues**. Right-clicking one of the TS Queues in this list displays a new option in the pop-up menu, as shown in Figure 9-4.

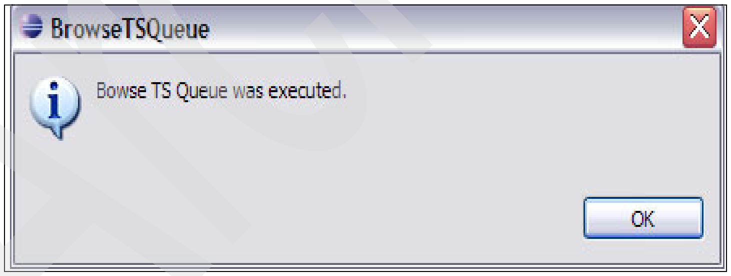


CNX0211I Context: EPRED. Resource: TSQUEUE. 4 records collected at Oct 20, 2009 2:00:51 PM

Region	Name	Location	Item Count	Queue Length	Max Item Length	Min Item Length
EPRED4	SCOTT	AUXILIARY	1	128	128	128
EPRE		AUXILIARY	5	640	128	128
EPRE		AUXILIARY	1	128	128	128
EPRE		AUXILIARY	1	128	128	128

Figure 9-4 Browse TS Queue option on pop-up menu

You have created your desired GUI interface without the need to write a single line of code. Select **TSQueueBrowse** → **Browse TS Queue** to see Figure 9-5.



Bowse TS Queue was executed.

Figure 9-5 Dummy action for TSQueue Browse plug-in

You now have to write the code to turn this dummy action into your desired function.

9.3 Coding your TSQueue Browse plug-in

Now comes the tricky part. You need to modify the code generated by Eclipse to implement your TS Queue Browse function. Remember, there are two objects you are working with:

- TS Queue Browse view
- TS Queue Browse pop-up action

9.3.1 Adding code to the TSQueue Browse pop-up menu

You need to modify the pop-up menu to do two things:

- Remember which TS queue the user has clicked so that we can use this information to retrieve the contents of that queue.
- Create the TS queue Browse view that displays the contents of the TS queue

Browse TS Queue pop-up menu function

In the Explorer package, expand the src folder and double-click the java source file `BrowseTSQueueAction.java` in the browsetsqueue.popup.actions package. The Java editor view displays. Double-click the BrowseTSQueueAction.java tab to get a full window edit session.

Change the selectionChanged() method

The method you modify to remember which TS queue you are working with is the `selectionChanged()` method. This method is invoked each time the user right-clicks a TS queue in the Explorer TS Queue view. Replace the no-op method that is generated by Eclipse with the code in Example 9-1.

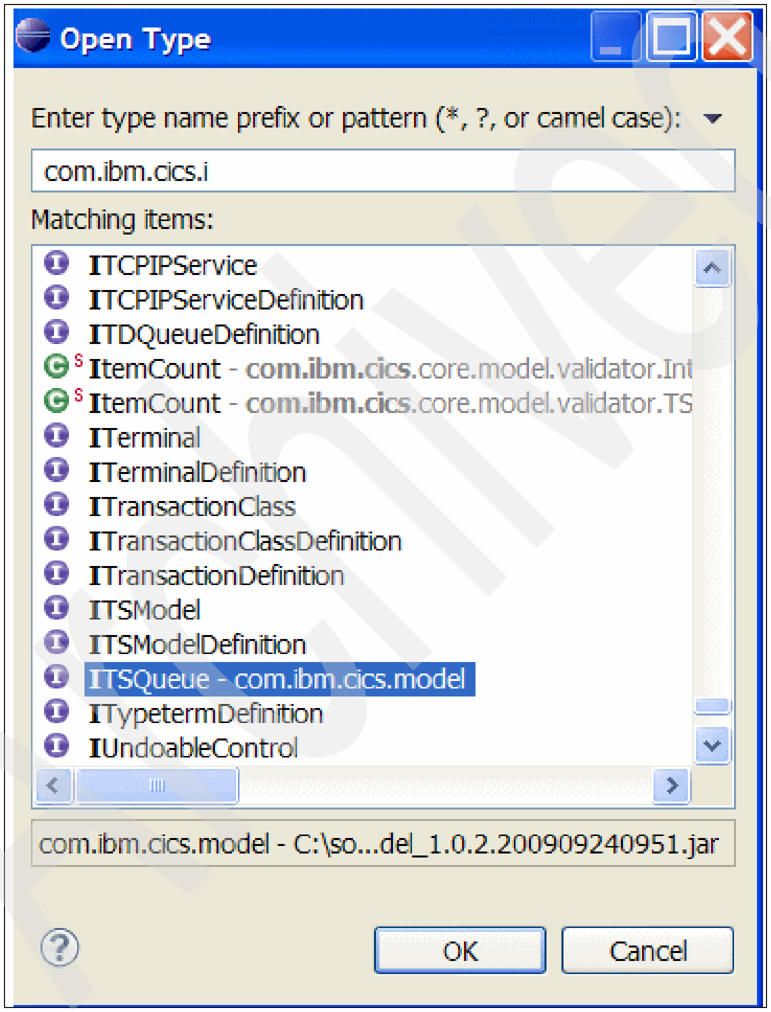
Example 9-1 Browse TSQueue pop-up menu selectionChanged method

```
public void selectionChanged(IAction action, ISelection selection)
   {
      tsqueue = (ITSQueue) ((StructuredSelection)
selection).getFirstElement();
   }
```

Eclipse flags errors when you put this code in. This is because you need to declare this plug-in’s dependency on the relevant CICS Explorer object (in this case the ITSQueue object).

If you are unsure about the name of the CICS Explorer class you need to use to manipulate your CICS object, use the Navigator tab on the workspace menu bar to help locate it.

Click the Navigator tab in the workspace action bar, and select **Open Type**. In the pop-up menu that displays, enter `string com.ibm.cics`. Eclipse presents you with a list of all the Java classes it has found in packages beginning with that name. When you are working with classes that manipulate CICS Explorer objects, you usually are working with the interface for those objects, so it is a good idea to enter the string `string com.ibm.cics.i` into the search field. You get a list that probably contains the object you want to manipulate. In our case, we are going to be working with the ITSQueue interface, as in Figure 9-6.


Open Type

Enter type name prefix or pattern (*, ?, or camel case):

com.ibm.cics.i

Matching items:

```
ITCPIPService
ITCPIPServiceDefinition
ITDQueueDefinition
ItemCount - com.ibm.cics.core.model.validator.Int
ItemCount - com.ibm.cics.core.model.validator.TS
ITerminal
ITerminalDefinition
ITransactionClass
ITransactionClassDefinition
ITransactionDefinition
ITSModel
ITSModelDefinition
ITSQueue - com.ibm.cics.model
ITypetermDefinition
IUndoableControl
```

com.ibm.cics.model - C:\so...del_1.0.2.200909240951.jar

Figure 9-6 Finding the right TS Queue object using the Eclipse Navigator

We have now identified the object we want to work with. We can start modifying the copied code to work with TS queues.

In the Package Explorer view, double-click the `plugin.xml` file of the BrowseTSQueue project to open the overview editor, and click the Dependencies tab. Click **Add** if the Required plug-ins pane, and enter the name of the CICS Explorer plug-in which contains the ITSQueue object `com.ibm.cics.model`

Press Ctrl+S to save the change, and double-click **BrowseTSQueue.java** view to return there. If you pass the cursor across the ITSQueue object, Eclipse offers to generate the import statement for you. Click this option, and the error associated with ITSQueue disappears. If you pass the cursor across the StructuredSelection object, you see that you have a similar problem. Eclipse offers to add the import statement because it already knows about the package that contains this class. Ask Eclipse to do the necessary import.

Finally, note that the tsqueue variable has not been previously declared. Because you need to access this variable across several different methods, it needs to be declared at the object level, and not at the method level. Add the declaration for tsqueue just after the class definition as shown in Example 9-2.

Example 9-2 Declaration for global variable tsqueue

```
public class BrowseTSQueue implements IObjectActionDelegate {

   private Shell shell;
   private ITSQueue tsqueue;
```

Change the run() method

The run method of your BrowseTSQueue pop-up action object is invoked when the user has clicked the Browse TS queue action in the pop-up menu. Replace the default code (which outputs the text shown in Figure 9-5 on page 277), with code that displays the Browse TS Queue view. Replace the default code with the code shown in Example 9-3 on page 281.

You see that there are errors associated with various objects, which Eclipse can resolve if you place the cursor over the error and select the appropriate import option. The only error that you cannot resolve in this way is the error associated with the `populateInformation()` method of the BrowseTSQueue view object. You have not yet defined this method, so this is normal (you define this method shortly).

Example 9-3 run method of BrowseTSQueue pop-up action object

```
   public void run(IAction action) {

      final IWorkbenchPage page =
PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage();

      try
      {
         IViewPart view =
page.showView("browsetsqueue.views.BrowseTSQueue");
         ((BrowseTSQueueView) view).populateInformation(tsqueue);
      }
      catch (PartInitException e) { e.printStackTrace(); }
   }
```

Press Ctrl+S to save your changes, and close the pop-up action package `BrowseTSQueueAction.java`.

You have now made all the modifications to the Browse TSQueue pop-up action menu object. The next step is to add the corresponding code in the Browse TSQueue view object.

Add code to the Browse TSQueue view object

Double-click the `BrowseTSQueueView.java` object in the browsetsqueue.views package to open an edit session. You are going to add code to do the following tasks:

- Ask to always be notified when an action is performed
- Populate the view with the contents of the requested TS Queue
- Show the populated view

Update ViewContentProvider() method

This method is responsible for providing objects to the associated view. In our case, the TS queue data is held in an array of strings, which are accessible globally throughout the class. The `populateInformation()` method calls CICS and puts the TS queue records into a global string array variable `tsrecord`, which is accessed through this method. Example 9-4 on page 282 shows the string array `tsrecord` being returned to the caller by this method.

Example 9-4 ViewContentProvider() method of BrowseTSQueueView class

```
class ViewContentProvider implements IStructuredContentProvider {
      public void inputChanged(Viewer v, Object oldInput, Object 
newInput) {
      }
      public void dispose() {
      }
      public Object[] getElements(Object parent) {
         return tsrecord;
      }
   }
```

Add init() method

You are going to add code to your view object to ensure that whenever an action is performed on a TS queue, you update the TS queue contents so that they are current. This is done by creating a Listener at the time your view object is created.

By default, Eclipse does not generate an `init` method for a view object, as it is implicit. However, in this case, you are adding function to be executed after the implicit `init` (what is termed a `super.init` method). You add code to create a SelectionListener for your view.

Each time a TS queue browse view is opened, Eclipse adds a selection listener to that view, to wait for selection events. When an object is selected in the TS queue browse view, the inline `selectionChanged()` method coded in Example 9-5 is invoked.

Example 9-5 BrowseTSQueue view object super.init method

```
@Override
public void init(IViewSite site) throws PartInitException
{
      super.init(site);

// Add a selection listener to look for when people click TSQueue 
entries
// in the main TSQueue view and then populate this view with their 
contents
      site.getPage().addSelectionListener(new ISelectionListener()
      {
         @Override
         public void selectionChanged(IWorkbenchPart arg0,ISelection 
selection)
         {
            if (!selection.isEmpty())
            {
               Object firstElement = ((StructuredSelection)
selection).getFirstElement();

               if (firstElement instanceof ITSQueue)
               {
                   final ITSQueue tsqueue = (ITSQueue) firstElement;

                   populateInformation(tsqueue);
               }
            }
         }
         });
      }
```

Note: This `selectionChanged()` method is unrelated to the pop-up action `selectionChanged()` method discussed in "Change the selectionChanged() method" on page 278. The methods have the same name but they are for different objects.

Because this `selectionChanged()` method is driven for any action requested in your view, it checks to see if the requested action is for a TS queue object, and if so, refreshes the TS queue contents by invoking the `populateInformation()` method described in the following section.

Add populateInformation() method

This is the key method of your new implementation. It is here that you call CICS TS to retrieve the contents of the TS queue being processed. This method performs the following tasks:

- Build an HTTP Request to send to CICS TS
- Process the HTTP response containing the TS queue records
- Make the processed records available to the other methods of the Browse TS queue view object.

Because the call to CICS TS might take time to complete, it is executed asynchronously, on a background thread. This method is shown in Example 9-6.

Example 9-6 populateInformation() method

```
public void populateInformation(final ITSQueue tsqueue) {
        Job job = new Job("MyJob") {
                @Override
                protected IStatus run(final IProgressMonitor monitor) {

                   monitor.beginTask("myJob", IProgressMonitor.UNKNOWN);

                   IO(monitor,tsqueue);

                   Display.getDefault().asyncExec(new Runnable()
                   {
                      public void run()
                      {
                         // Do your GUI updates here
                         GUI();
                      }
                   });
                                                                monitor.done();

                   return Status.OK_STATUS;
                }
        };
        job.schedule();
```

To make it easier to split out the code that needs to execute asynchronously, the function that actually makes the call to CICS has been separated into a routine called `IO()` and a code that communicates with the user put into a method called `GUI()`. These two method calls are bracketed by calls associated with a `job` object (which separates them out into a separate job for execution on a separate thread) and a `monitor` object used to track the progress of the job (which generates a little progress bar at the foot of the workspace). See Figure 9-7

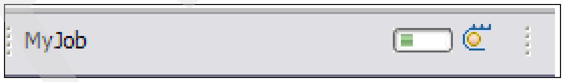


Figure 9-7 Eclipse Progress Bar

This separation of IO and GUI routines is suggested when working with CICS Explorer.

Note: In our testing, the access to the CICS back-end application was so fast that the progress bar associated with the monitor object never got the chance to appear unless we trapped the alias transaction with CEDF.

In Example 9-7 we can see the `IO()` method, which is invoked by `populateInformation()`. In Example 9-7 we see the call to CICS over an HTTP connection using the classes supplied in java.net.

Example 9-7 IO() method called by populateInformation()

```
void IO(final IProgressMonitor monitor,final ITSQueue tsqueue)
   {
      int length = tsqueue.getItemCount().intValue();
      tsrecord = new String[length+1];
      URL url = null;
      String tsname = tsqueue.getName();

      //Identify the HTTP connection to the target CICS Web Support 
application
      //with the URL expected by CICS Web Support
      try {
            url = new URL("http://9.12.4.74:2404/CICS/CWBA/CEBRWEB");
      } catch (MalformedURLException e1) {
         // TODO Auto-generated catch block
         System.out.println("Entered populateInformation" + 
e1.getMessage());
      }

      // Set up connection to the target CICS Web Support application
      // this particular application expects a method of POST.
      HttpURLConnection urlc = null;
      try {
         urlc = (HttpURLConnection) url.openConnection();
         try {
            urlc.setRequestMethod("POST");
         } catch(ProtocolException e)
         { throw new Exception("Shouldn't happen - doesn't support POST 
???",e);
         }

         // Build the body of the HTTP POST request.
         // this particular application expects one input field 
TSQUEUE=tsqname
         urlc.setDoOutput(true);
```

```
        urlc.setDoInput(true);
        urlc.setUseCaches(false);
        urlc.setAllowUserInteraction(false);
        OutputStream out = urlc.getOutputStream();

        // Send the HTTP Request to CICS
        try {
           String outputstring = "TSQUEUE=" + tsname;
           out.write(outputstring.getBytes());
           out.flush();
           } catch(IOException e) {
           throw new Exception("IOException while posting data");
        } finally {
           if (out!=null)
           out.close();
        }

        // Read the response returned by CICS
        InputStream in = urlc.getInputStream();
        BufferedReader rd = new BufferedReader(new
InputStreamReader(in));
        String line;
        int j = 0;
        while ((line = rd.readLine()) != null) {
           tsrecord[j] = line;
           j+=1;
        }
        rd.close();
     } catch(Exception e) {
        System.out.println("Exception here!!");
        e.printStackTrace();
     }
  }
```

The code handling the HTTP request and response is standard Java for manipulating HTTP.

The HTTP request sent to CICS by our plug-in is shown in Example 9-8.

Example 9-8 HTTP POST request to invoke program CEBRWEB

```
POST /CICS/CWBA/CEBRWEB HTTP/1.1
Cache-Control: no-cache..Pragma: no-cache
User-Agent: Java/1.6.0
Host: 9.12.4.74:2404
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive.
Content-type: application/x-www-form-urlencoded
Content-Length: 13

TSQUEUE=STEVE...........
```

The HTTP response returned by CICS looks like Example 9-9.

Example 9-9 HTTP response

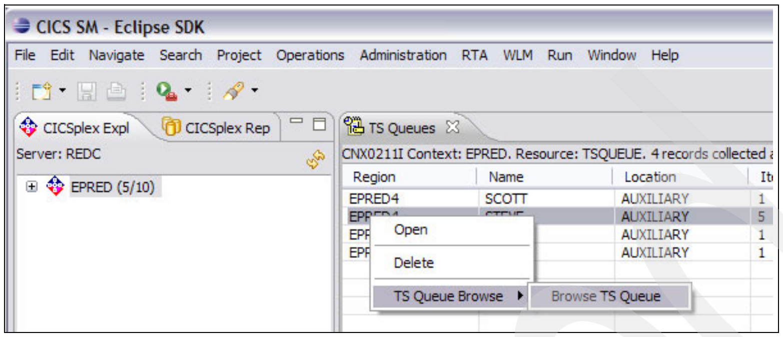
```
HTTP/1.1 200 OK..Date: Thu, 22 Oct 2009 19:17:38 GMT
Server: IBM_CICS_Transaction_Server/4.1.0(zOS)
Content-Length: 000000000000412
Connection: Keep-Alive

ABCD
EFGH
IJKL
MNOP
QRST
```

The TS queue records are delimited by a CRLF sequence at the end of each record (as it stands, this sample only works for TS queues that contain text).

The loop at the end of the `IO()` method loops through the body of the HTTP response, and writes each line (TS queue record) out to a separate string in our `tsrecord` string array.

The TS Browse view itself is only updated when the `GUI()` method is invoked to perform a `viewer.refresh()` call.

In Figure 9-8 we have selected **Operations** → **Queues** → **TS Queues**. When we click the CICSPlex in the left pane to set the context, CICS Explorer populates the view with information about TS queues in the CICSPlex.

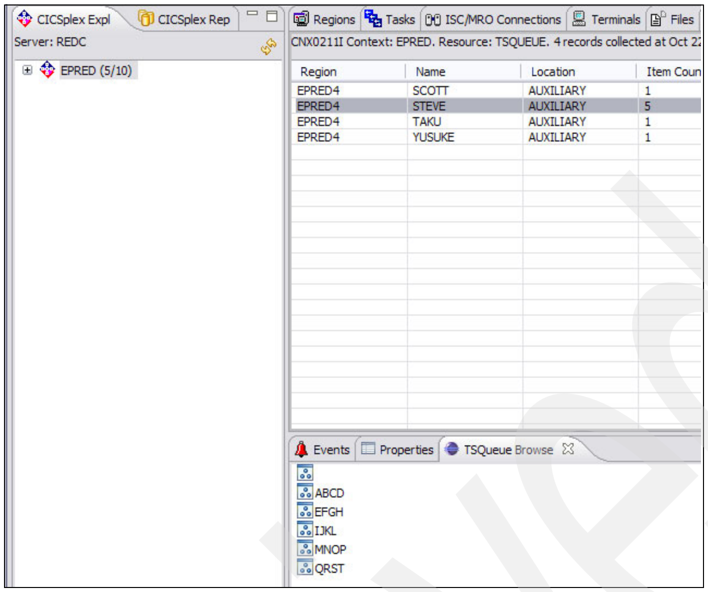


CNX0211I Context: EPRED. Resource: TSQUEUE. 4 records collected

Region	Name	Location	It
EPRED4	SCOTT	AUXILIARY	1
EPRED4	STEVE	AUXILIARY	5
EPR		AUXILIARY	1
EPR		AUXILIARY	1

Figure 9-8 Select a TS queue to be browsed

When you click the CICSplex name on the left, CICS Explorer populated the upper TS Queue operations view with the details of the TS queues that exist in CICSPlex EPRED. The Browse TS queue pane is still empty because we have not yet selected a TS queue to browse. If we now right-click one of the TS queues in the upper-center pane, and choose the option **TS Queue Browse** → **Browse TS Queue**, the lower-center pane is populated with the records of the selected TS queue (Figure 9-9 on page 289). After the Browse TS queue is displayed left-click the TS queue you are interested in to see what it contains.



CNX0211I Context: EPRED. Resource: TSQUEUE. 4 records collected at Oct 2

Region	Name	Location	Item Coun
EPRED4	SCOTT	AUXILIARY	1
EPRED4	STEVE	AUXILIARY	5
EPRED4	TAKU	AUXILIARY	1
EPRED4	YUSUKE	AUXILIARY	1

Figure 9-9 Populated Browse TS Queue Panel

9.4 CICS TS Application Specification

The back-end application that is invoked by the CICS Explorer Browse TS Queue plug-in is a CICS Web Support COBOL SPI program that does the following:

- Parses the HTML forms data sent by CICS Explorer to extract the name of the TS Queue to be returned
- Performs an EXEC CICS INQUIRE TS QUEUE to retrieve details about the requested TS queue
- Executes a loop to retrieve the items in the TS queue and adds them to the body of the HTTP response returned to CICS Explorer
- Returns an HTTP response containing the TS Queue contents to CICS Explorer

The CEBRWEB program is supplied with the additional materials for this book.

Note: We have a couple of “To Do” items for this sample, which we did not have time to implement

- Support for exchanging and displaying binary data (between the CEBRWEB program and the BrowseTSQueue plug-in)
- Ability to rewrite one of the records in the queue

If you want to implement any of them, you can download the Eclipse workspace containing the existing code and add to the plug-in, and perhaps share your new plug-in with the CICS Explorer community through the CICS Explorer forum.

9.5 Extending the connection preferences panel

In this section, we extend the CICS Explorer connection preferences panel to input connection details for the CICS service. This replaces the existing hard coded values in the example code.

9.5.1 Creating the extension points

Starting with the BrowseTSQueueView workspace, double-click **plugin.xml** and select the Extensions tab. Clear the **Show only extension points from the required plug-ins** check box, select **com.ibm.cics.core.comm**, and click **Finish**.

When prompted with the message “Do you want to add plug-in com.ibm.cics.core.comm, declaring the connections extension point, to the list of plug-in dependencies?” select **Yes**.

Repeat the same for extension com.ibm.cics.core.ui.connectionCustomizers.

Click the plugin.xml tab and observe that the extensions have been added to the bottom of the XML.

Replace the content of the com.ibm.cics.core.comm extension with Example 9-10 on page 291.

Example 9-10 The connections extension point

```
<extension point="com.ibm.cics.core.comm.connections">
   <category abbreviatedName="CEBR"
   connectionType="browsetsqueue.connection.IWebConnection"
   id="browsetsqueue.connection.category"
   name="CEBR Category">
   </category>
   <connection category="browsetsqueue.connection.category"
   class="browsetsqueue.connection.WebConnection"
   id="browsetsqueue.connection.connection"
   name="CEBR Web Connection">
   </connection>
</extension>
```

Replace the content of the com.ibm.cics.core.ui.connectionCustomizers extension with Example 9-11.

Example 9-11 The connectionCustomizer extension point

```
<extension point="com.ibm.cics.core.ui.connectionCustomizers">
   <customizer
   class="browsetsqueue.connection.ConnectionCustomizer"
   connectionId="browsetsqueue.connection.connection"
   id="browsetsqueue.connection.customizer"
   name="CEBR Web Customizer">
   </customizer>
   </extension>
```

Press CTRL+S to save the updates. You have now done enough for your new connection type to be shown in the CICS Explorer connection preferences. You can observe this by running the BrowseTSQueueView application and selecting **Windows** → **Preferences** → **Connections**. From the Connection Type drop down mneu select **CEBR Web Connection**. Input the correct values for your connection and click **Apply**. The connection information is now saved.

Click the **Connect** button and you see that nothing happens. This is because you have not yet created the classes that handles the connection activity. This is what we do next.

9.5.2 Creating the connection classes

Next, we can create the classes that provide a connection to he back-end service. We need:

- An interface that is referenced by the CICS Explorer classes when handling the connection
- A class that connects directly with the back-end service, and implement the interface above
- A class that exposes the connection details on the connection preferences panel
- A class that provides a bridge between CICS Explorer and the class that connects to the back end service above.

IWebConnection interface

Back in your Eclipse development environment create a new interface with the package `browsetsqueue.connection` and name `IWebConnection`. The interface needs to extend com.ibm.cics.core.comm.IConnection. This interface matches up with the connectionType entry in the category element of your `plugin.xml`.

You do not need to specify any methods in the interface. The interface is used by proxies under the covers by the CICS Explorer and Java proxies only work with interfaces. The IWebConnection interface is shown in Example 9-12.

Example 9-12 The IWebConnection interface

```
package browsetsqueue.connection;

import com.ibm.cics.core.comm.IConnection;

public interface IWebConnection extends IConnection {}
```

Important: In order for the CICS Explorer to work with the IWebConnection interface, you must make its package visible from the plug-in. To do this double-click `plugin.xml` and select the Runtime tab. Click the **Add** button next to Exported Packages and select **browsetsqueue.connection** from the list. Press CTRL+S to save the changes.

The package is now added to the export list and is visible to any Eclipse plug-in that chooses to work with it.

WebConnection class

A class is needed to directly deal with the connection to the back-end service. In the case of BrowseTSQueueView all that is needed is the ability to store and retrieve the host name and port number of the CICS TCPIPService to which the view connects. Requests to the service are only made when needed and no permanent connection state is maintained. Because of this, the class you use to deal with the connection is simple.

In the same Java package create a class called `WebConnection` that extends com.ibm.cics.core.comm.AbstractConnection and implements browsetsqueue.connection.IWebConnection. Eclipse prompts you to add in the required unimplemented methods. Move the mouse cursor of the WebConnection class name and select **Add unimplemented methods**. The finished version of `WebConnection` is shown in Example 9-13.

Example 9-13 The WebConnection class

```
package browsetsqueue.connection;

import com.ibm.cics.core.comm.AbstractConnection;
import com.ibm.cics.core.comm.ConnectionException;

public class WebConnection extends AbstractConnection
               implements IWebConnection
{
   private boolean connected = false;

   @Override
   public void connect() throws ConnectionException
   {
      connected = true;
   }

   @Override
   public void disconnect() throws ConnectionException
   {
      connected = false;
   }

   @Override
   public boolean isConnected()
   {
      return connected;
   }
}
```

ConnectionCustomizer class

Looking at plugin.xml, the connectionCustomizer extension references a class under the customizer element. This class deals with modifications to the connection preferences panel specific to your connection type. Even if you choose to use the default entry fields on the connection preferences panel you must still use the connectionCustomizer extension because this causes your connection to get instantiated.

Alongside the above classes create one called ConnectionCustomizer that implements the interface com.ibm.cics.core.ui.IConnectionCustomizer. Make sure you add the necessary methods that the IConnectionCustomizer interface requires you to implement.

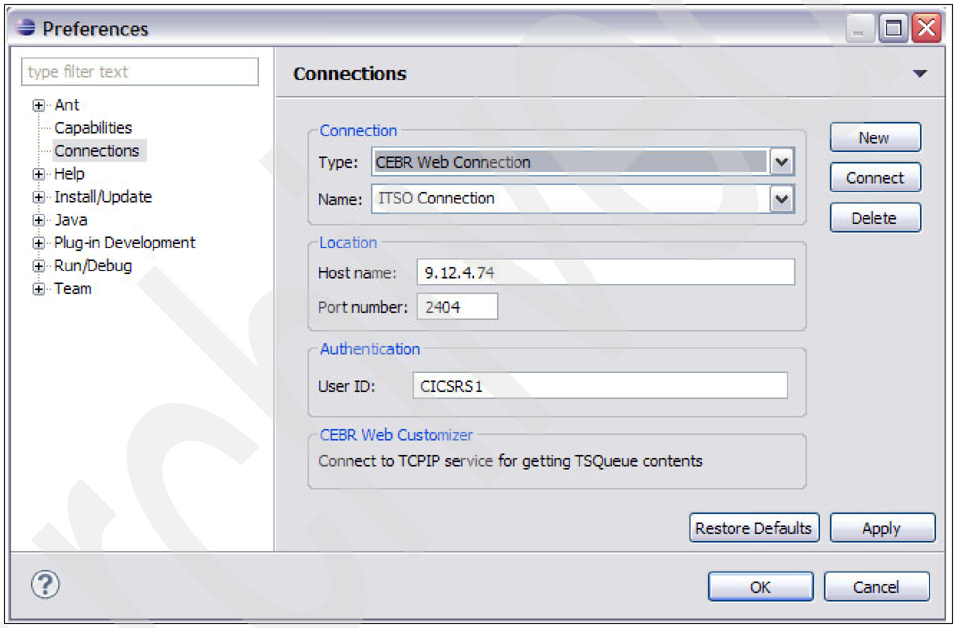
The createControl() method provides an opportunity to extend the connection preferences panel with additional widgets. For the CEBR Web connection we add a Label widget that explains the purpose of this connection type. Example 9-14 shows the finished version of the ConnectionCustomizer class.

Example 9-14 The ConnectionCustomizer class

```
package browsetsqueue.connection;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IConfigurationElement;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import com.ibm.cics.core.comm.ConnectionConfiguration;
import com.ibm.cics.core.ui.IConnectionCustomizer;

public class ConnectionCustomizer implements IConnectionCustomizer
{
   @Override
   public void createControl(Composite arg0)
   {
      Label label = new Label(arg0, SWT.NONE);
      label.setText("Connect to TCPIP service for " +
            "getting TSQueue contents");
   }

   @Override
   public boolean performOk() { return true; }

   @Override
   public void setConfiguration(ConnectionConfiguration arg0)
   {}
```

```
    @Override
    public void setDirty(boolean arg0) {}

    @Override
    public void setInitializationData(IConfigurationElement config,
String propertyName, Object data) throws CoreException {}
}
```

Running the example and opening the connection preferences panel gives the CEBR Web information, as shown in Figure 9-10.



Connection

Type: CEBR Web Connection

Name: ITSO Connection

Location

Host name: 9.12.4.74

Port number: 2404

Authentication

User ID: CICSRS1

CEBR Web Customizer

Connect to TCPIP service for getting TSQueue contents

Figure 9-10 Connection preferences panel with CEBR Web Connection details

If you fill in the fields and click **Connect**, you see that nothing happens. This is due to the way the CICS Explorer interacts with connection objects. To get make this function work, further objects need to be created.

WebConnectable class

Create a new Java class called `WebConnectable`. The new class must implement the interface com.ibm.cics.core.comm.IConnectable and provide implementations of all of its methods. The `WebConnectable` class provides a bridging mechanism for the CICS Explorer to work with the WebConnection object. Only one instance of it can exist in the system, so the relevant code must be in place to enforce this. The technique for creating a singleton of an object is as follows:

1. Create a static instance of the object inside its own class.
2. Define a private constructor such that no one other than the class itself can instantiate it.
3. Create a public static method called `getDefault()` that returns the static instance of the class defined in step 1.

Example 9-15 shows the finished version of the `WebConnectable` class.

Example 9-15 The WebConnectable class

```
package browsetsqueue.connection;

import com.ibm.cics.core.comm.ConnectionException;
import com.ibm.cics.core.comm.IConnectable;
import com.ibm.cics.core.comm.IConnectableListener;
import com.ibm.cics.core.comm.IConnection;

public class WebConnectable implements IConnectable
{
   // Singleton instance of this class
   private static WebConnectable webConnectable
               = new WebConnectable();

   // Restrict others from instantiating this class
   private WebConnectable() {}

   // Return the singleton instance
   public static WebConnectable getDefault()
   { return webConnectable; }

   private IConnection webConnection;

   @Override
   public void addListener(IConnectableListener arg0) {}

   @Override
   public void disconnect()
   {
      try { webConnection.disconnect(); }
      catch (ConnectionException e)
      { e.printStackTrace(); }
   }

   @Override
   public IConnection getConnection()
   { return webConnection; }

   @Override
   public Class<IWebConnection> getConnectionType()
   { return IWebConnection.class; }

   @Override
   public boolean isConnected()
   { return webConnection.isConnected(); }

   @Override
   public void setConnection(IConnection arg0)
   { webConnection = arg0; }
}
```

The final step is to register the `WebConnectable` class with the UIPlugin resource manager. This enables the connection preferences panel to work with the object when the **Connect** button is pressed.

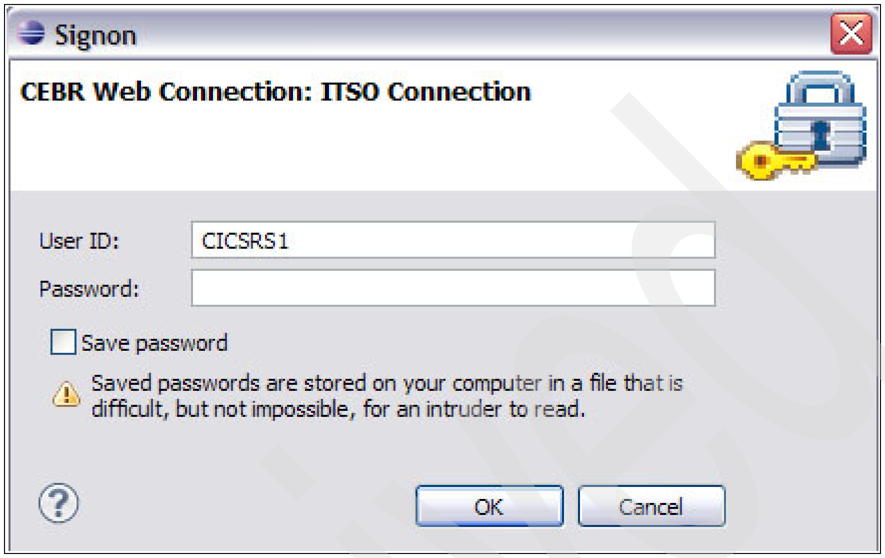
9.5.3 Register WebConnectable with the resource manager

The `WebConnectable` class needs to be registered prior to the connection preferences panel being shown. One place to do this is the Activator class of your plug-in. If you do not have an Activator class, you can do it at static initialization of the `ConnectionCustomizer` class. Example 9-16 shows the additional code that is added to ConnectionCustomizer to perform the registration.

Example 9-16 Registering the WebConnectable class with the resource manager

```
public class ConnectionCustomizer implements IConnectionCustomizer
{
   static
   {
   UIPlugin.getDefault()
   .setResourceManager("browsetsqueue.connection.category",
   WebConnectable.getDefault());
   }
...
```

Next, run the plug-in and click **Connect**. You see the password dialog box shown in Figure 9-11.


Signon

CEBR Web Connection: ITSO Connection

User ID: CICSRS1

Password:

Save password

Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

Figure 9-11 The sign-on dialog for your connection

Note: Because we are not using authentication in this example, the password field can be left blank. If you choose to use HTTP basic authentication, this is where the appropriate login information is entered.

Click **OK** in the dialog box to return to the connection preferences panel. Even though it is not easy to see it at this point, you are signed in.

9.5.4 Viewing sign-on status in the trim bar

The CICS Explorer has a trim widget at the bottom right of the workspace for showing signing status. Figure 9-12 shows the CICS Explorer trim widget.

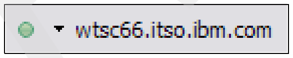


Figure 9-12 The CICS Explorer trim widget

The CEBR Web Connection can make use of this trim whenever the CEBR view is selected. This is done by overriding the `getPartProperty()` method of your view. Add the code shown in Example 9-17 on page 299 to `BrowseTSQueueView` to make use of the trim widget.

Example 9-17 Override getPartProperty() to make use of the trim widget

```
public String getPartProperty(String key)
{
   if (IConnectionCategory.class.getName().equals(key))
   return "browsetsqueue.connection.category";
   else return super.getPartProperty(key);
}
```

Next, run the plug-in, sign-in, and open the TSQueue Browse view. The trim widget is updated to show your new connection. Figure 9-13 shows the trim widget with a connection to the ITSO systems. The green icon indicates that the connection is active.

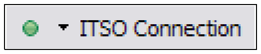


Figure 9-13 The CEBR Web Connection widget

9.5.5 Using the CEBR Web Connection

At this point we have created the object for managing the connection to the CEBR service in CICS. We next update the TSQueue Browse view to make use of the connection.

The TSQueue Browse view must signify its interest in the CEBR Web Connection to work with it. This is done by creating an class of type `IResourceManagerListener` and registering it with the resource manager. After the listener is registered it receives notifications about the connection status. Add the code in Example 9-18 to `BrowseTSQueueView` as an inner class for receiving notifications.

Example 9-18 WebConnectionListener inner class

```
private class WebConnectionListener
         implements IResourceManagerListener
{
   @Override
   public void connected(IConnectable arg0)
   { connection = arg0.getConnection(); }

   @Override
   public void connecting(IConnectable arg0) {}

   @Override
   public void disconnected(IConnectable arg0)
```

```
    { connection = null; }

    @Override
    public boolean disconnecting(IConnectable arg0)
    { return false; }

    @Override
    public void exception(IConnectable arg0,
                Exception arg1) {}
}
```

Add an `IConnection` variable to `BrowseTSQueueView`, as shown in Example 9-19. This holds a reference to the connection object that is given to the `connected()` method of the inner class.

Example 9-19 IConnection instance for holding the connection object

```
private IConnection connection;
```

You must now register an instance of the `WebConnectionListener` with the resource manager. This is done by adding the code in Example 9-20 to the end of BrowseTSQueueView's `init()` method.

Example 9-20 Registering the WebConnectionListener with the resource manager

```
UIPlugin.getDefault()
.addResourceManagerListener("browsetsqueue.connection.category",
new WebConnectionListener());
```

Whenever the connection status is modified, the `WebConnectionListener` instance is notified.

Note: The `WebConnectionListener` only gets added when the TSQueue Browse view is first opened. This implies that if you have signed on before the view is opened then you miss the call to the `connected()`. This is not the case however, because the `connect()` method of the inner class always gets called by the resource manager when the view is opened, even if it 10 minutes after you have signed on.

The final stage is to make use of the connection details when retrieving the TSQueue contents across HTTP.

9.5.6 Updating BrowseTSQueueView to use connection details

Currently the host name and port number of the CICS service to connect to are hard coded in the `IO()` method of BrowseTSQueueView. Now that a connection is available, the host name and port number need to be replaced with the values given to the connection preferences panel. Do this by updating the URL object creation in the `IO()` method so it looks like the code shown in Example 9-21.

Example 9-21 Creating a URL using the connection object information

```
url = new URL("http://" + connection.getHost() + ":" +
        connection.getPort() + "/CICS/CWBA/CEBRWEB");
```

If the connection object does not exist (that is, you have not yet signed on to the CEBR Web Connection), a NullPointerException occurs when the preceding code is run. To prevent this from happening make the call to `IO()` conditional on the connection object existing and having a valid connection. Do this by modifying the call to `IO()` in the `populateInformation()` method, as shown in Example 9-22.

Example 9-22 Conditionally calling IO() depending on the connection status

```
if (connection != null && connection.isConnected())
      IO(monitor, tsqueue);
```

Next, run the plug-in code, sign in, and connect. You can see that the TSQueue information is being retrieved as before, but this time it is using the connection information you specified in the connection preferences panel.

Note: The code in the preceding example shows a valid connection even if an invalid IP address is entered. An enhancement to this code is to implement a `ping()` method that validates the host name and port information.

The finished code for this section is in the workspace titled BrowseTSQueueWithConnectionPanel.

A

Reference list of CICS SDK elements

This appendix provides the listing of available CICS types, system manager actions, definition builders and mutable objects.

CICS types

The following sections are a reference list of CICS SDK CICS types.

CICS Resources

- AtomService
- Bundle
- BundlePart
- CaptureSpecification
- CompletedTask
- Connection
- DB2Connection
- DB2Entry
- DB2Transaction
- DBCTLSubsystem
- DocumentTemplate
- EventBinding
- EventProcessing
- ExtrapartitionTDQueue
- GlobalDynamicStorageArea
- IndirectTDQueue
- IntrapartitionTDQueue
- IntervalControlRequest
- IPICConnection
- JVMClassCache
- JVMStatus
- JVMPool
- JVMProfile
- JVMServer
- Library
- LibraryDSName
- LocalFile
- LocalTransaction
- Pipeline
- ProcessType
- Program
- Region
- RemoteFile
- RemoteTDQueue
- RemoteTransaction
- RPLList
- Task

- TCPIPService
- Terminal
- TransactionClass
- TSModel
- TSQueue
- UnitOfWorkEnqueue
- URIMap
- WebService
- WMQConnection
- WMQConnectionStatistics
- WMQInitiationQueue
- XMLTransform

CICS Definitions

- AtomServiceDefinition
- BundleDefinition
- ConnectionDefinition
- CorbaServerDefinition
- DB2ConnectionDefinition
- DB2EntryDefinition
- DB2TransactionDefinition
- DocumentTemplateDefinition
- DeployedJARFileDefinition
- EnqueueModelDefinition
- FileDefinition
- IPICConnectionDefinition
- JournalModelDefinition
- JVMServerDefinition
- LibraryDefinition
- LSRPoolDefinition
- MapSetDefinition
- PartnerDefinition
- PipelineDefinition
- ProcessTypeDefinition
- ProfileDefinition
- ProgramDefinition
- PartitionSetDefinition
- RequestModelDefinition
- SessionDefinition
- TCPIPServiceDefinition
- TDQueueDefinition
- TerminalDefinition
- TransactionDefinition

- TransactionClassDefinition
- TSModelDefinition
- TypetermDefinition
- URIMapDefinition
- WebServiceDefinition
- WMQConnectionDefinition

CPSM Definitions

- CICSRegionDefinition
- CICSRegionGroupDefinition
- ResourceDescriptionDefinition
- ResourceGroupDefinition

CPSM Managers

- BatchedRepositoryUpdateRequest
- CICSplex
- CMASDetails
- Event
- ManagedRegion
- WLMActiveWorkload

CSD Definitions

- CSDGroupDefinition
- CSDListDefinition

System manager actions

- Acquire
- AddToGroup
- AllStatistics
- ARMRestart
- Backout
- Cancel
- Close
- CloseForce
- CloseImmediate
- CloseNoWait
- CloseWait

- Commit
- Connect
- Delete
- DeleteShipped
- Deregister
- Disable
- DisableForce
- DisableNoWait
- DisableWait
- Discard
- DisconnectForce
- DisconnectNoWait
- DisconnectWait
- Drain
- Enable
- EndAffinity
- EventProcessing
- Force
- ForceCancel
- ForcePurge
- ForcePurgeJVMClassCacheAutoStart
- ForcePurgeJVMClassCacheNoAutoStart
- ForcePurgeTask
- Kill
- KillTask
- Inservice
- Install
- Lock
- NewCopy
- NoRecoveryData
- NotPending
- Open
- Outservice
- PhaseIn
- PhaseOut
- PhaseOutJVMClassCacheAutoStart
- PhaseOutJVMClassCacheNoAutoStart
- Purge
- PurgeJVMClassCacheAutoStart
- PurgeJVMClassCacheNoAutoStart
- PurgeTask
- Rebuild
- Release
- RemoveFromGroup
- ResetTime

- Resynchronize
- Scan
- SecurityRebuild
- ShutdownImmediate
- ShutdownNormal
- ShutdownTakeover
- Start
- Stop
- Switch
- Unlock

Definition builders

Here is the list of all available definition builders. All entries that end with the word **Gen** are abstract and cannot be instantiated. Instead you must use one of its subclasses.

- AtomServiceDefinitionBuilderGen
  - AtomServiceDefinitionBuilder
- BuilderHelper
- BundleDefinitionBuilder
- ConnectionDefinitionBuilder
- CorbaServerDefinitionBuilder
- DB2ConnectionDefinitionBuilder
- DB2EntryDefinitionBuilder
- DB2TransactionDefinitionBuilder
- DeployedJARFileDefinitionBuilder
- DocumentTemplateDefinitionBuilderGen
  - DocumentTemplateDefinitionBuilder
- EnqueueModelDefinitionBuilder
- FileDefinitionBuilder
- IPICConnectionDefinitionBuilder
- JournalModelDefinitionBuilder
- JVMServerDefinitionBuilder
- LibraryDefinitionBuilder
- LSRPoolDefinitionBuilder
- MapSetDefinitionBuilder
- PartitionSetDefinitionBuilder
- PartnerDefinitionBuilder
- PipelineDefinitionBuilder
- ProcessTypeDefinitionBuilder
- ProfileDefinitionBuilder

- ProgramDefinitionBuilder
- RequestModelDefinitionBuilder
- ResourceDescriptionDefinitionBuilder
- ResourceGroupDefinitionBuilder
- SessionDefinitionBuilder
- TCPIPServiceDefinitionBuilder
- TDQueueDefinitionBuilderGen
  - ExtrapartitionTDQueueBuilder
  - IndirectTDQueueBuilder
  - IntrapartitionTDQueueBuilder
  - RemoteTDQueueBuilder
- TerminalDefinitionBuilder
- TransactionClassDefinitionBuilder
- TransactionDefinitionBuilder
- TSModelDefinitionBuilder
- TypetermDefinitionBuilder
- URIMapDefinitionBuilderGen
  - AtomURIMapDefinitionBuilder
  - ClientURIMapDefinitionBuilder
  - PipelineURIMapDefinitionBuilder
  - ServerURIMapDefinitionBuilder
- WebServiceDefinitionBuilder
- WMQConnectionDefinitionBuilder

Mutable objects

- IMutableAtomService
- IMutableAtomServiceDefinition
- IMutableBundle
- IMutableBundleDefinition
- IMutableBundlePart
- IMutableCaptureSpecification
- IMutableCICSDefinition
- IMutableCICSResource
- IMutableConnection
- IMutableConnectionDefinition
- IMutableCorbaServerDefinition
- IMutableCPSMDefinition
- IMutableDB2Connection
- IMutableDB2ConnectionDefinition
- IMutableDB2Entry
- IMutableDB2EntryDefinition
- IMutableDB2Transaction

- IMutableDB2TransactionDefinition
- IMutableDBCTLSubsystem
- IMutableDeployedJARFileDefinition
- IMutableDocumentTemplate
- IMutableDocumentTemplateDefinition
- IMutableEnqueueModelDefinition
- IMutableEventBinding
- IMutableEventProcessing
- IMutableExtrapartitionTDQueue
- IMutableFileDefinition
- IMutableGlobalDynamicStorageArea
- IMutableIndirectTDQueue
- IMutableIntervalControlRequest
- IMutableIntrapartitionTDQueue
- IMutableIPICConnection
- IMutableIPICConnectionDefinition
- IMutableJournalModelDefinition
- IMutableJVMClassCache
- IMutableJVMPool
- IMutableJVMProfile
- IMutableJVMServer
- IMutableJVMServerDefinition
- IMutableJVMStatus
- IMutableLibrary
- IMutableLibraryDefinition
- IMutableLibraryDSName
- IMutableLocalFile
- IMutableLocalTransaction
- IMutableLSRPoolDefinition
- IMutableMapSetDefinition
- IMutablePartitionSetDefinition
- IMutablePartnerDefinition
- IMutablePipeline
- IMutablePipelineDefinition
- IMutableProcessType
- IMutableProcessTypeDefinition
- IMutableProfileDefinition
- IMutableProgram
- IMutableProgramDefinition
- IMutableRegion
- IMutableRemoteFile
- IMutableRemoteTDQueue
- IMutableRemoteTransaction
- IMutableRequestModelDefinition
- IMutableResourceDescriptionDefinition

- IMutableResourceGroupDefinition
- IMutableRPLList
- IMutableSessionDefinition
- IMutableTask
- IMutableTCPIPService
- IMutableTCPIPServiceDefinition
- IMutableTDQueueDefinition
- IMutableTerminal
- IMutableTerminalDefinition
- IMutableTransactionClass
- IMutableTransactionClassDefinition
- IMutableTransactionDefinition
- IMutableTSModel
- IMutableTSModelDefinition
- IMutableTSQueue
- IMutableTypetermDefinition
- IMutableURIMap
- IMutableURIMapDefinition
- IMutableWebService
- IMutableWebServiceDefinition
- IMutableWMQConnection
- IMutableWMQConnectionDefinition
- IMutableWMQConnectionStatistics
- IMutableWMQInitiationQueue
- IMutableXMLTransfor

View IDs

Resource views

- com.ibm.cics.core.ui.view.cicsPlexRepositories
- com.ibm.cics.core.ui.view.cicsplexes
- com.ibm.cics.core.ui.view.connections
- com.ibm.cics.core.ui.view.eventbindings
- com.ibm.cics.core.ui.view.events
- com.ibm.cics.core.ui.view.files
- com.ibm.cics.core.ui.view.regions
- com.ibm.cics.core.ui.view.tasks
- com.ibm.cics.core.ui.view.tdqueues
- com.ibm.cics.core.ui.view.terminals
- com.ibm.cics.core.ui.view.transactions
- com.ibm.cics.core.ui.view.tsqueues

- com.ibm.cics.sm.ui.views.atomServices
- com.ibm.cics.sm.ui.views.batchedRepositoryUpdateRequests
- com.ibm.cics.sm.ui.views.bundleParts
- com.ibm.cics.sm.ui.views.bundles
- com.ibm.cics.sm.ui.views.captureSpecifications
- com.ibm.cics.sm.ui.views.cicsstors
- com.ibm.cics.sm.ui.views.cmasDetails
- com.ibm.cics.sm.ui.views.completedTasks
- com.ibm.cics.sm.ui.views.db2Connections
- com.ibm.cics.sm.ui.views.db2Entries
- com.ibm.cics.sm.ui.views.db2Transactions
- com.ibm.cics.sm.ui.views.dbctlsss
- com.ibm.cics.sm.ui.views.documentTemplates
- com.ibm.cics.sm.ui.views.eventProcessing
- com.ibm.cics.sm.ui.views.intervalControlRequests
- com.ibm.cics.sm.ui.views.ipicConnections
- com.ibm.cics.sm.ui.views.jvmClassCaches
- com.ibm.cics.sm.ui.views.jvmPools
- com.ibm.cics.sm.ui.views.jvmProfiles
- com.ibm.cics.sm.ui.views.jvmServers
- com.ibm.cics.sm.ui.views.jvmStatus
- com.ibm.cics.sm.ui.views.libraries
- com.ibm.cics.sm.ui.views.libraryDSNames
- com.ibm.cics.sm.ui.views.pipelines
- com.ibm.cics.sm.ui.views.processTypes

- com.ibm.cics.sm.ui.views.programs
- com.ibm.cics.sm.ui.views.rplList
- com.ibm.cics.sm.ui.views.tcpipServices
- com.ibm.cics.sm.ui.views.transactionClasses
- com.ibm.cics.sm.ui.views.tsModels
- com.ibm.cics.sm.ui.views.unitOfWorkEnqueues
- com.ibm.cics.sm.ui.views.uriMaps
- com.ibm.cics.sm.ui.views.webServices
- com.ibm.cics.sm.ui.views.wlmActiveWorkloads
- com.ibm.cics.sm.ui.views.wmqConnectionStatistics
- com.ibm.cics.sm.ui.views.wmqConnections
- com.ibm.cics.sm.ui.views.wmqInitiationQueues
- com.ibm.cics.sm.ui.views.xmlTransforms

Definition views

- com.ibm.cics.core.ui.view.transactionDefinitions

- com.ibm.cics.sm.ui.views.atomServiceDefinitions
- com.ibm.cics.sm.ui.views.bundleDefinitions
- com.ibm.cics.sm.ui.views.connectionDefinitions
- com.ibm.cics.sm.ui.views.corbaServerDefinitions
- com.ibm.cics.sm.ui.views.db2ConnectionDefinitions
- com.ibm.cics.sm.ui.views.db2EntryDefinitions
- com.ibm.cics.sm.ui.views.db2TransactionDefinitions
- com.ibm.cics.sm.ui.views.deployedJARFileDefinitions
- com.ibm.cics.sm.ui.views.documentTemplateDefinitions
- com.ibm.cics.sm.ui.views.enqueueModelDefinitions
- com.ibm.cics.sm.ui.views.fileDefinitions
- com.ibm.cics.sm.ui.views.ipicConnectionDefinitions
- com.ibm.cics.sm.ui.views.journalModelDefinitions
- com.ibm.cics.sm.ui.views.jvmServerDefinitions
- com.ibm.cics.sm.ui.views.libraryDefinitions
- com.ibm.cics.sm.ui.views.lsrPoolDefinitions
- com.ibm.cics.sm.ui.views.mapSetDefinitions
- com.ibm.cics.sm.ui.views.partitionSetDefinitions
- com.ibm.cics.sm.ui.views.partnerDefinitions
- com.ibm.cics.sm.ui.views.pipelineDefinitions
- com.ibm.cics.sm.ui.views.processTypeDefinitions
- com.ibm.cics.sm.ui.views.profileDefinitions
- com.ibm.cics.sm.ui.views.programDefinitions
- com.ibm.cics.sm.ui.views.requestModelDefinitions
- com.ibm.cics.sm.ui.views.resourceDescriptionDefinitions

- com.ibm.cics.sm.ui.views.resourceGroupDefinitions
- com.ibm.cics.sm.ui.views.sessionDefinitions
- com.ibm.cics.sm.ui.views.tcpipServiceDefinitions
- com.ibm.cics.sm.ui.views.tdQueueDefinitions
- com.ibm.cics.sm.ui.views.terminalDefinitions
- com.ibm.cics.sm.ui.views.transactionClassDefinitions
- com.ibm.cics.sm.ui.views.tsModelDefinitions
- com.ibm.cics.sm.ui.views.typetermDefinitions
- com.ibm.cics.sm.ui.views.uriMapDefinitions
- com.ibm.cics.sm.ui.views.webServiceDefinitions
- com.ibm.cics.sm.ui.views.wmqConnectionDefinitions

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247819`

Alternatively, you can go to the IBM Redbooks Web site at:

`ibm.com``/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247819.

Using the Web material

The additional Web material that accompanies this paperbook includes the following files:

File name	*Description*
CETRTraceWorkSpace.zip	CETRTrace sample Eclipse CICS Explorer workspace
OMEGAMON Chapter source code (and project data).zip	OMEGAMON source code
Shaylaworkspaces.zip	CETRTrace sample Eclipse CICS Explorer workspace
Steveworkspaces.zip	CEBR sample Eclipse CICS Explorer workspace
Workspace_cicsex1.zip	Basic Hello World sample Eclipse CICS Explorer workspace
Workspace_cicsex2_20091023.zip	Hello World sample Eclipse CICS Explorer workspace
Workspace_cicsex6_20091028_fullspec.zip	Hello World full function sample Eclipse CICS Explorer workspace
Workspace_sticky notes.zip	Sticky Notes sample Eclipse CICS Explorer workspace

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 318. Note that some of the documents referenced here might be available in softcopy only.

- *CICS Explorer*, SG24-7778

Other publications

These publications are also relevant as further information sources:

- *Eclipse BuildIng Commercial-Quality Plug-ins,*Clayberg and Rubel ISBN 0-321-42672-X

Online resources

These Web sites are also relevant as further information sources:

- Description of Eclipse RCP from www.eclipse.org

  http://www.eclipse.org/downloads/download.php?file=/technology/phoenix/talks/What-is-Eclipse-and-Eclipse-RCP-3.2.6.ppt
- Architectural Styles and the Desgn of Network-based Software Architectures

  http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

  http://www.surfscranton.com/architecture
  http://rest.blueoxen.net/cgi-bin/wiki.pl?whatIsREST
- Ajax and REST, Part 1

  http://www.ibm.com/developerworks/web/library/wa-ajaxarch/

- Restful Architecture

  http://www.surfscranton.com/architecture
- RESTwiki

  http://rest.blueoxn.net/cgi-bin/wiki.pl?whatIsREST
- CICS TS Version 4.1 Information Center

  http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM

Redbooks

Extend the CICS Explorer: A Better Way to Manage Your CICS

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Extend the CICS Explorer

A Better Way to Manage Your CICS

Add value to the CICS Explorer with Eclipse plug-ins

Unlock the CICS Explorer Software Development Kit

Follow examples of tool integration

CICS Explorer is the latest significant evolution in the management and analysis of your CICS environment. It is a statement of intent from the CICS Development organization, which is determined to ensure you can manage your CICS estate in a simple and easily extensible way, using a combination of the following approaches:

- Tried and trusted CICS expertise and technology
- The widely accepted user interfaces and integration power of the open source Eclipse platform
- Web 2.0 and RESTful programming (this technology underpins the CICS Explorer concept)

This IBM Redbooks publication shows how you can use the extensible design of CICS Explorer to complement the functionality already provided, with added functionality tailored to the needs of your business. We show you how to perform the following tasks:

- Install the CICS plug-in SDK into your eclipse environment
- Develop a simple plug-in for the CICS Explorer
- Deploy the plug-in into CICS Explorer

We provide several useful examples of plug-ins that we developed during the residency using the methodology we describe.

The starting point for the book is that you already have CICS Explorer installed and configured with connectivity to your CICS region or CICSPlex, and that you are looking for ways to customize CICS Explorer.

SG24-7819-00 0738434019