WebSphere® software

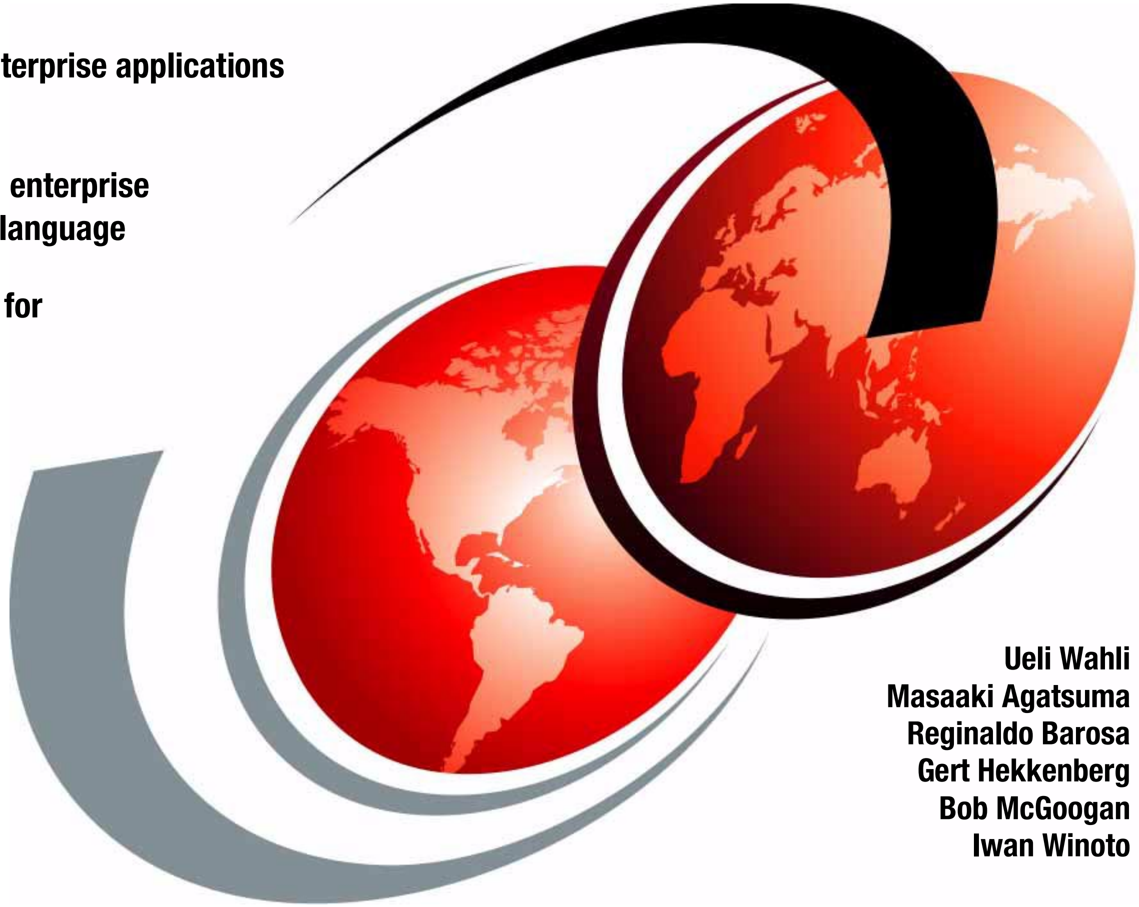
IBM

Legacy Modernization with WebSphere Studio Enterprise Developer

Creating enterprise applications with Struts

Introducing enterprise generation language

Developing for z/OS



Ueli Wahli
Masaaki Agatsuma
Reginaldo Barosa
Gert Hekkenberg
Bob McGoogan
Iwan Winoto

Redbooks

ibm.com/redbooks

International Technical Support Organization

Legacy Modernization with WebSphere Studio Enterprise Developer

December 2002

SG24-6806-00

Take Note! Before using this information and the product it supports, read thel information in “Notices” on page xv.

First Edition (December 2002)

This edition applies to WebSphere Studio Enterprise Developer Version 5 **Early Availability** Release and WebSphere Application Server Version 5 for use with the Windows 2000 and WIndows NT Operating Systems.

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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

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

Trademarks

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

Redbooks(logo)™
AIX®
CICS®
CICS/MVS®
COBOL/370™
Database 2™
DB2®
IBM®
IMS™
Language Environment®
MQSeries®
MVS™
OS/390®
RACF®
Redbooks™
SAA®
S/390®
SP™
VisualAge®
WebSphere®
z/OS™

The following terms are trademarks of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both:

Lotus®
Word Pro®
Lotus Notes®
Notes®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

The ability to connect components is the first step in modernizing your application portfolio. In this IBM Redbook, we look at a real-world example of creating and connecting a Web application to enterprise business logic using the Struts-based model-view-controller (MVC) framework and associated tooling within the the Enterprise Developer that makes this a snap.

To address the needs of large enterprises, a model-based paradigm for building applications in a Struts-based MVC framework is being delivered in the WebSphere Studio Suite. It provides a visual construction and assembly-based environment supporting the implementation of enterprise-level applications and including support for the multiple developer roles and technologies required by those applications. Examples of the technologies supported include HTML, Java, servlet, EJB, COBOL, EGL, PL/I, and connectors.

EGL is a high-level language that supports the development of applications in either WebSphere (Java) or traditional transactional environments (CICS). EGL's focus is to allow developers of various backgrounds to be able to write mission-critical business processes for the Internet, which can be leveraged from Struts-based Web applications.

This redbook introduces a sample application that encompasses Enterprise Developer concepts and best practices.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Ueli Wahli is a Consultant IT Specialist at the IBM International Technical Support Organization in San Jose, California. Before joining the ITSO 18 years ago, Ueli worked in technical support at IBM Switzerland. He writes extensively and teaches IBM classes worldwide on application development, object technology, VisualAge for Java, WebSphere Studio and WebSphere Application Server products. Ueli holds a degree in Mathematics from the Swiss Federal Institute of Technology.

Masaaki Agatsuma is a member of the AD tool development team in the IBM Software Group, Yamato Software Laboratory, Japan. He has been working with IBM since 1986, in various areas such as product development, technical support, and systems integration. His areas of expertise include middleware and portals in general, as well as XML, Lotus Notes, and Java 2 Enterprise Edition (J2EE). He holds a degree in Computer Science from the Science University of Tokyo, Japan.

Reginaldo Barosa is an IBM Certified Application Development Specialist. He provides sales support, helping customers with WebSphere application development tools such as VisualAge Generator and WebSphere Studio. Before joining IBM US two years ago, Reginaldo worked for 27 years in IBM Brazil. He has co-authored IBM Redbooks, has written many articles, is the author of two books, and has written four articles for the WebSphere Developer Domain (WSDD). He holds a degree in Electrotechnic Engineering from Instituto Mauá de Technologia, São Paulo, Brazil.

Gert Hekkenberg is a Senior IT Specialist from IBM Software Group EMEA region North, based in Amsterdam, The Netherlands. He has over 18 years of application-enabling experience with a focus on application development solutions. One area of special interest is software configuration management (SCM). He is currently working as Technical Sales Consultant designing E2E application development solutions for the larger customers in his region and The Netherlands in particular. He has written extensively on application development and SCM in various Redbooks over the years and was involved in developing various ITSO workshops as well. Gert holds a Masters degree in Business Information Systems from Erasmus University, Rotterdam, The Netherlands, and a Bachelors degree in economics from Vrije Universiteit, Amsterdam, The Netherlands.

Bob McGoogan is an IT Specialist in RTP, NC. He has 17 years of experience in the IT field, in both software development and technical sales. He has spent the last 4½ years developing Java-based Proof-of-Concepts for customers, using VisualAge for Java, WebSphere Studio, and WebSphere Studio Application Developer.

Iwan Winoto is a Senior Technical IT Specialist in IBM Software Group in Sydney, Australia. He started with IBM in December 2000 and has specialized in WebSphere Application Server and related development tools. Iwan has been in the IT industry since 1990, working mostly in the finance and insurance industry in Australia and Switzerland. He has had various roles from application developer to project manager and has worked with various programming languages including Clipper, COBOL, SmallTalk and Java. He holds a degree in Electronic Engineering from Swinburne Institute of Technology in Melbourne, Australia.

Thanks to the following people for their contributions

Clifford Meyers, John Casey, Stephen Hancock, Jason Garcowski, Jon Gregory, John Snyder, Mark Evans, Roger Newton, Rajesh Daswani, Keith Tapp and Henry Koch of the Enterprise Developer development team in the IBM RTP lab, Raleigh.

Larry England, Gary Mazo, Wilbert Kho, Kent Hawley, Mel Fowler, Pavan Immaneni, Venkat Balabhadrapatruni, Teodoro Cipresso, and Anthony Flusche of the Enterprise Developer development team in the IBM Silicon Valley Lab, San Jose.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

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

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Part 1

Introduction

In Part 1 we introduce the challenge of modernization of legacy applications, the WebSphere Studio Enterprise Developer product, and the sample application that is used to illustrate the concepts.

1

Modernization of enterprise applications

This chapter discusses the problems that today's businesses face when optimizing access to enterprise data and applications through modern dynamic Web interfaces.

In this chapter, the following topics are discussed:

- Current business pressures, driving the move to enable Web access to some enterprise application capability
- Current state of enterprise applications and Web application development
- New approach in modernizing legacy applications using a version of the model-view-controller pattern for Web applications

Business pressures

There is significant pressure to move many phases of business to emerging e-business models because solutions based on e-business models optimize fundamental business processes. The pressure to optimize business processes comes from an increasingly competitive marketplace, where timeliness, access to information, and accuracy of information are critical to success. As a result, today's enterprise must make applications and data available to any level of the business at any time. With e-business solutions, an enterprise can more effectively manage relationships with customers and suppliers, speed decision making, reduce project cycle times and increase control over inventory.

e-business means making business applications and data more accessible to the user, whether they be an external customer, Business Partner or employee of the enterprise. With e-business, users are expecting access to information and functions that make their job easier. Moreover, users want that access to be fast, efficient and flexible. e-business also allows the business processes to be dynamically changed.

Existing information technology investment

Many computer industry analysts agree that large enterprises' IT investment over the last 30 years has been in building and maintaining legacy systems (source: CBDi interact report). Legacy systems do not only include host-based COBOL business applications. A more general definition of legacy systems are systems that are not flexible enough to adapt to new business computing paradigms as they emerge.

Current legacy systems that have been built over the last 30 years have used paradigms such as:

- Mainframe batch
- Host terminals
- Client/server
- Distributed components and packaged applications

The evolution of each of these paradigms has been accompanied by a new programming language and toolset and therefore a new set of skill requirements. In order for a new paradigm to make use of existing legacy systems, that new paradigm must be able to interact with the technologies of the paradigms that came before it.

Typically legacy systems, especially the older ones, have evolved over many years as business requirements have changed. These systems have become monolithic and unable to support evolutionary enhancements as new paradigms have emerged. Monolithic legacy systems also have layers of functionality and technologies (to manage data, process and presentation), which have hard-wired dependencies on each other. This is true also of some client/server-based systems, where there is a tight coupling between the presentation technology and the underlying business logic implementation.

Legacy applications also arise through a lack of foresight in skills and tools investment. Many systems exist that are maintained using low-level APIs and tools such as Notepad. Replacing and maintaining such systems becomes difficult because the knowledge and skills are held by a few expert developers.

The evolution of the software development team

The process for planning and building software applications has been evolving for many years. Historically, requirements were gathered, project leads were put in place, and a team was enlisted. Next, the work was distributed, milestones were set, and project-tracking mechanisms were put in place. An entire software project might be built in Cobol, PL/I, C, C++, or some other programming language. In the past, software development teams were fairly homogeneous. Generally, the entire application would run on a single platform. Within the project everyone basically spoke the same language, learned the same processes, and set similar milestones. This is not the case for teams creating e-business solutions.

Today's Web application development teams often include business analysts, managers, host programmers, application programmers, Web page designers, graphics designers, Java programmers, and component developers. One person might fill each role, or any one person might be required to play multiple roles. Planning a Web application has become complex because of the varied skills and numerous roles required.

For example, the enterprise information systems may have COBOL or PL/I programmers with experience building CICS transactions or other applications and databases associated with current business logic. Their approach to development is probably based on models of structured programming (and most likely do not separate the user interface from the business logic). The processes for building such systems are specific to the host environment.

As indicated previously, your team needs people with experience in your current business applications (perhaps host programmers) working on aspects of the middle-tier. Yet, on the middle-tier, you have Java programmers building servlets, classes, or perhaps JSPs. Their development and architectural model is likely to be more object-oriented. The middle-tier application server environment has its own set of run-time issues. Build and deployment tasks are also particular to the middle-tier. It is crucial to get the business knowledge that is embedded in your existing applications and leverage it as you develop in the middle-tier.

Graphic designers and HTML programmers develop for presentation on client systems (graphics, JSPs, HTML). Most recently, this has meant browsers but other client platforms, such as hand-held devices and data-enabled phones are becoming popular. Team members with these skills tend to have backgrounds in building user interfaces.

Of course, managers of teams developing Web applications might come from any of these programming disciplines or perhaps a technical business role, or some other technical lead position. Teams require development tooling that not only allows diverse roles to interact, but tooling that enables them to work together as an integrated team.

Software architecture considerations

In this section we discuss what forces have driven application implementation up to now, and what can be done to bring these applications to the modern e-business computing paradigm.

A brief history of software development

Today's enterprise may find it difficult to create e-business solutions and Web applications. Why is this so?

Over the last 30 years, enterprises have made significant information technology investments. During that time frame, enterprises have sought to increase their effectiveness by centralizing critical data and applications, by leveraging high-end transaction processing systems, and by developing untold numbers of software applications that have, over time, become core to the day-to-day operation of their businesses.

The result is 30 years' worth of applications and significant business processing power that must be surfaced in an architecture, where it can be used and extended in e-business solutions. This situation creates a significant challenge, which hinders Web application development, modernization, and ultimately, time to market.

Chaos in Web application development

The problems facing enterprise development teams are compounded by the fact that the evolution of Web application development has been rocky, at best. The Web application development environment has been fractured and piece-meal and so have been the results. Many early Web applications have been little more than chaotic strings of HTML and scattered programs written in CGI or PERL scripts. These are not the components of a robust, 24x7 enterprise application. Security, reliability, and maintainability have often been suspect. A significant contributing factor to this situation has been the lack of adequate development tooling for building complex Web applications.

Web applications and their middle-tier components have often been created by what can best be called “point-tools”; development tools that were extremely narrow in terms of their function and focus. Stated plainly, there have been no tools that provide sufficient assistance with the creation of complete end-to-end e-business solutions. Development teams entering the emerging Internet space early have been forced to cobble together development environments as best they could.

To make matters worse, the basic development process of building Web applications can be described as somewhat haphazard, or perhaps even as an “anti-development process”. It can be argued that some developers entering the new Web space initially set aside (or perhaps at times abandoned) approaches and processes that had become standard practice in enterprise development shops.

Generally, this turning away from process has been defended by intent to build applications in “Web-time”. The reality is that the development processes that have historically helped ensure quality code are just as important as ever. Today’s point tools do not incorporate the knowledge and best practices of the last 30 years of application development.

Patterns for e-business

To bring structure into Web application development, IBM has come up with the Patterns for e-business, a set of proven topologies, technologies, and products. For more information, refer to the Web site at:

http://www.ibm.com/developerworks/patterns/

Model-view-controller

In order to improve the Web application development process, we must consider the following technologies in the design of a Web application for the J2EE environment:

- Java servlets and JavaServer Pages (JSP)
- JavaBeans and Enterprise JavaBeans (EJB)

Together, these object types form the core of a powerful J2EE architecture expressed in the model-view-controller (MVC) design pattern that was originally developed to help manage change in software application development.

Model-view-controller separates the user interface from business logic and data. The key aspects are:

Model The model contains the core of the application function. The model captures the state of the application. It does not include knowledge of the view or controller.

View The view is the look of the application. The view presents, gathers, and submits information, but it does not include knowledge of the model or controller.

Controller The controller manages the execution flow of the application, passing appropriate state information between the model and the view.

MVC applied to Web applications

Applying the model-view-controller approach to Web application design allows the key aspects of a Web application to be isolated and maintained independently.

For Web applications, the classical form of model-view-controller needs modification. This is true because the Web brings unique challenges to software developers, most importantly the stateless connection between the client and server. This stateless behavior makes it difficult for the model to notify the view of changes. On the Web, the browser must re-query the server to discover modification to the state of the data within the application.

Another change is that the view is implemented using different technologies (for example, Java, PERL, C/C++) from the model or controller. This fact creates the requirement to separate key development roles. For instance:

- Business programmers should focus on developing services, not HTML.
- The page designer does not require direct involvement in (or awareness of) service development.

- Changes to page layout (by a page designer) should not require changes to code (of a service developer).
- Customers of the service should be able to create views to meet their specific needs.

Model-view-controller, modified for the Web, is called MVC2 or model-2. Figure 1-1 presents the MVC2 approach as it might appear in an N-tier environment:

- Input from the user is taken at the client and passed to a controller (servlet) that examines the input and the current state of the model.
- The model (or business objects) may reside on application servers, host servers, or both.
- The controller then populates the appropriate response view with data obtained from the model.
- The view (display page) is then presented to the user.

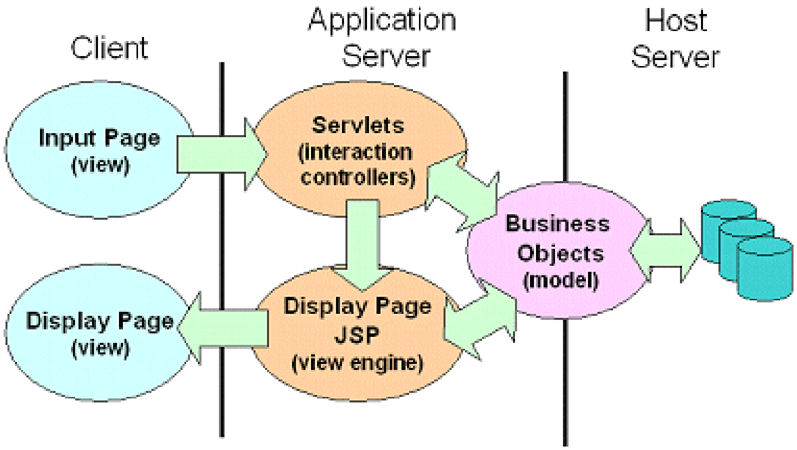


Figure 1-1 Model-2

We can see that using a design pattern such as model-2 helps to separate code responsibility and associated roles within the development team. Such separation helps ensure that changes (regardless of where they occur) are isolated.

Struts

In an effort to bring the advantages of the MVC design pattern to J2EE Web application development, this model has been implemented in an open source framework called *Struts*, released under the Apache Software License in July 2001.

The Struts framework provides the benefits of modularity, flexibility, and reusability of components, combined with the easy development associated with MVC-based Web applications.

In Chapter 4, “Components of a Struts-based application” on page 67, we describe the basic Web application technologies and introduce the components of a Struts-based application.

2

Introduction to WebSphere Studio Enterprise Developer

This chapter provides an overview of WebSphere Studio Enterprise Developer (abbreviated as Enterprise Developer), which we use to develop the sample application. We provide a tour of the integrated development environment (IDE) in which we discuss:

- WebSphere Studio Workbench open tooling platform
- The different perspectives available in Enterprise Developer, such as the Web, Java, J2EE, EGL, Data, XML, Server, z/OS Projects, and Debug perspectives
- The different types of projects within Enterprise Developer, such as the Java, EAR, Web, EJB, server, and MVS projects
- Support for Struts development as a special Web project
- Servers and their configuration for testing of applications
- Support for z/OS development

WebSphere Studio Enterprise Developer

Enterprise Developer builds on WebSphere Studio Application Developer (Application Developer). Application Developer brings together most of the features offered by VisualAge for Java and WebSphere Studio *Classic Edition*. Besides the features of these two products, new features were added, as shown in Figure 2-1.

You can learn about the new and adopted features of Enterprise Developer when you explore this chapter. To gain a more detailed understanding and some practical experience of Application Developer, please read the redbook *WebSphere Studio Application Developer Programming Guide*, SG24-6585.

Figure 2-1 also shows the capabilities that Enterprise Developer adds to Application Developer. These features are discussed in more detail in this chapter. The rest of this book focuses on the features in the Enterprise Developer; however, it is not necessary for the reader to have an in-depth knowledge of Application Developer.

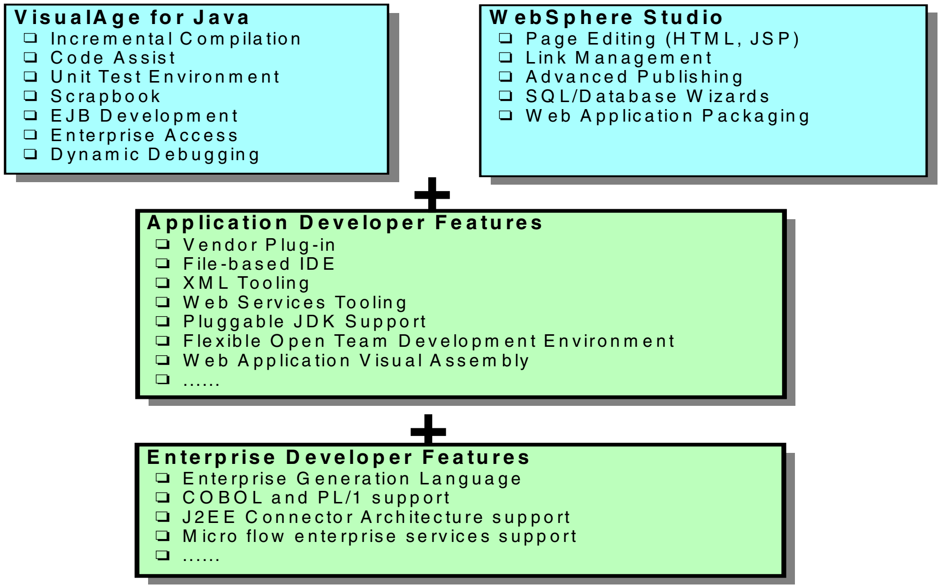


Figure 2-1 Enterprise Developer features

WebSphere Studio Workbench

WebSphere Studio Workbench is the brand name for the new open, portable universal tooling platform and integration technology from IBM. It forms the base for the new WebSphere Studio suite (WebSphere Studio Site Developer, WebSphere Studio Application Developer, WebSphere Studio Application Developer Integration Edition, and WebSphere Studio Enterprise Developer).

The Workbench is for customers and for tool builders who want to plug their tools into the WebSphere Studio product set. The Eclipse open source project (http://www.eclipse.org) enables other tool vendors to develop plug-ins for the WebSphere Studio Workbench. The tool providers write their tools as plug-ins for the Workbench, which operates on files in the workspace.

When the Workbench is launched, the user sees the integrated development environment composed of the different plug-ins. WebSphere Studio Workbench provides APIs, building blocks, and frameworks to facilitate the development of new plug-ins. There can be interconnections between plug-ins by means of extension points. Each plug-in can define extension points that can be used by other plug-ins to add function. For example, the Workbench plug-in defines an extension point for user preferences. When a tool plug-in wants to add items in that preferences list, it just uses that extension point and extends it.

Workspace

The resources you work with are stored in the workspace. By default, the workspace is a directory called *workspace* inside the product installation directory.

It is possible and suggested to place the workspace directory anywhere on the file system by starting the Enterprise Developer with a flag:

```
wsenterprise.exe -data d:\MyWorkspace
```

Restriction: This does not work in the early availability product, which displays a pop-up window where you can enter the workspace path and directory name. Do not use Select to not display this dialog again, because you cannot get it back to change the workspace location.

User interface

The Workbench user interface (UI) is implemented using two toolkits:

- Standard widget toolkit (SWT)—a widget set and graphical library integrated with the native Window operating system but with an OS-independent API.

- JFace—a UI toolkit implemented using SWT.

The whole Workbench architecture is shown in Figure 2-2.

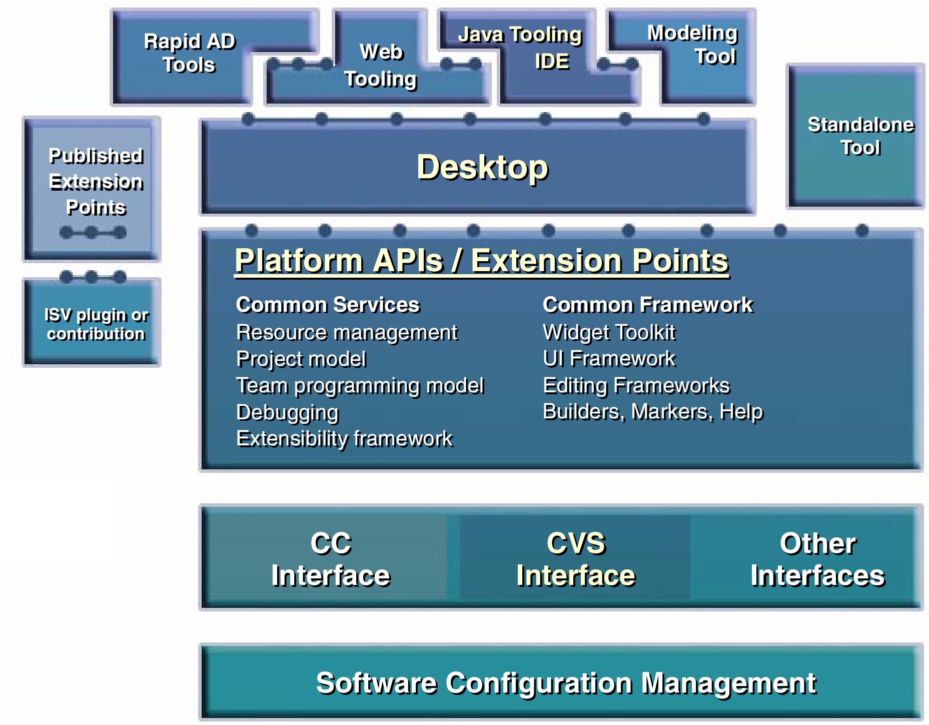


Figure 2-2 The Workbench architecture

Here are some explanations about the acronyms used in Figure 2-2.

Concurrent Versions System (CVS)

CVS is the open standard for version control systems. More information on CVS can be found at `http://www.cvshome.org`.

ClearCase (CC and CCLT)

ClearCase LT is from Rational and is shipped with the Enterprise Developer. ClearCase full function can be purchased separately from Rational. More information can be found at `http://www.rational.com`.

Other versioning systems will be supported in future versions of the Enterprise Developer product or by code shipped by other vendors (Merant, for example).

Workbench window

In this redbook, we refer to the interface of Enterprise Developer as the Workbench. It is an integrated development environment that promotes role-based development. For each role in the development of your e-business application, it has a different and customizable perspective.

A perspective is the initial set and layout of the views in the Workbench. Each perspective is related to a development task or role. For example, if you want to develop Java applications, you first create a Java project. When you work in a Java project, you probably use the Java perspective because that is the most useful perspective to do Java developing. We give an overview of the different perspectives and projects in the next sections.

Perspectives

Perspectives are a way to look through different glasses to a project. Depending on the role you are in (Web developer, Java developer, EJB developer) and/or the task you have to do (developing, debugging, deploying) you open a different perspective. The Workbench window can have several perspectives opened, and each can have its own window on the desktop if desired (select *Window -> Preferences -> Workbench -> Perspectives* to set the option for multiple windows).

Perspective basics

Figure 2-3 shows the Web perspective. You can switch easily between perspectives by clicking the different icons in the perspective tool bar.

- You can open a new perspective by clicking the ⊞ icon in the perspective toolbar. Alternatively you can select *Window -> Open Perspective* and then select the desired perspective from the list (in some cases you have to select *Other* to get a list of all perspectives).
- Each perspective has its own views and editors that are arranged for presentation on the screen (some may be hidden at any given moment). Several different types of views and editors can be open at the same time within a perspective.
- There are several perspectives predefined (Resource, Java, Web, J2EE, EGL, Data, XML, Server, Debug) in the Workbench. You can customize them easily by adding, deleting, or moving the different views.
- You can also compose your own perspective by defining the views it should contain.

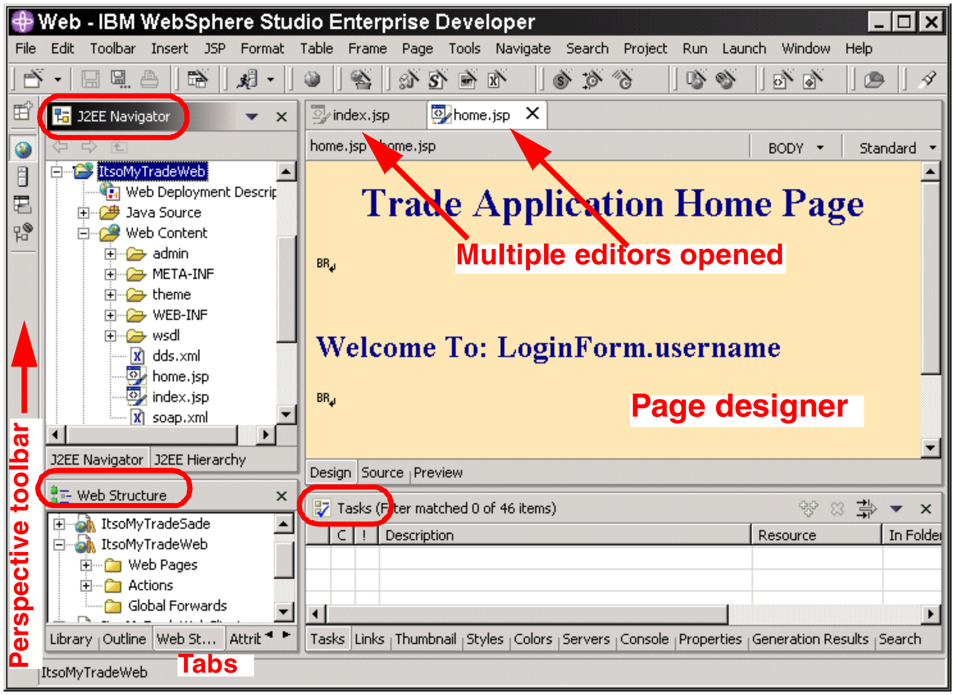


Figure 2-3 Web perspective

Views and editors

We first discuss the different views that appear in most perspectives and then take a closer look at some of the most used perspectives. Some of the views appear in most of the perspectives.

Navigator view

The Navigator view shows you how your resources are structured into folders. The Navigator view is available in most perspectives and it always displays all the folders and files of all projects. There are three kinds of resources:

Projects You use projects to organize all your resources and for version management. When you create a new project, a folder with the name of the project is created in the workspace.

Folders Folders are like directories in the file system. They can contain files as well as other folders. Folders are usually stored in the project directory, but they can also be outside of the workspace.

Files Files correspond to files in the file system and reside in folders.

Editors

By double-clicking a resource, the associated editor opens and allows you to modify it. In Figure 2-3, the active editor is the page designer, associated with JSP and HTML files. If no editor is currently associated with a particular file extension, the Workbench checks if there is one associated in the operating system and uses that editor to edit the file. You can also open OLE document editors such as Word, which is associated with the *.doc* extension.

You can change or add editors associated with a file extensions:

- From the menu bar, select *Window -> Preferences*.
- In the left pane, select *File editors* under the *Workbench* hierarchy.
- You can then select a file extension and associate an internal or external editors for it.

When you double-click another resource, a different editor shows up. You can easily switch between the different opened resources by selecting them on the top bar above the editor area. If the tab of your editor contains an asterisk (*), it means that it contains unsaved changes.

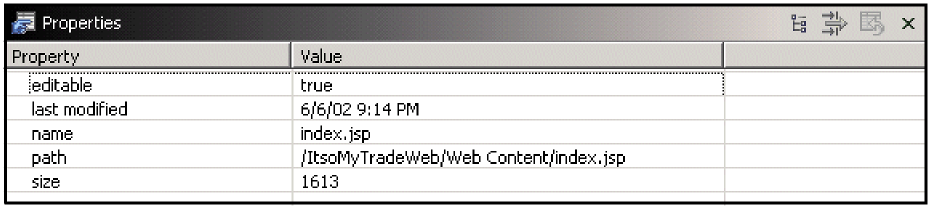
Outline view

The Outline view is always associated with the active editor.

The Outline view gives you an overview of the key elements that make up the resource that is being edited. It allows quick and easy navigation through your resource. By selecting one of the elements in the Outline view, the line in the editor view that contains the selected element gets highlighted and the editor pane is adjusted to make the element visible.

Properties view

When you click a resource in the Navigator view and then open the Properties view, you can view the different properties of that resource. The Properties view contains general things such as the full path on the file system, the date when it was last modified, and the size, as shown in Figure 2-4.



Property	Value
editable	true
last modified	6/6/02 9:14 PM
name	index.jsp
path	/ItsoMyTradeWeb/Web Content/index.jsp
size	1613

Figure 2-4 Properties view

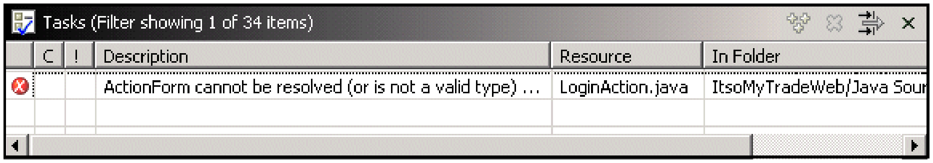
Tasks view

The Tasks view contains a list of two types of elements:

Problems Problems are tool-determined issues that have to be resolved. Example problems are Java compile errors, or broken links for HTML/JSP files. They are automatically added to the Tasks view when working with the tool. When you double-click a problem, the editor for the file containing the problem opens and the cursor is pointed at the location of the problem.

Tasks You can manually add tasks yourself. For example, you can add a task that reminds you that you have to implement a Java method. Place the cursor in the method's implementation, right-click and select *Add* -> *Task*. When you double-click, the file opens and the cursor is located in the method. You can also add general tasks that do not refer to a specific file.

You can set up several filters to show only the tasks you really want to see. For example, by clicking the filter icon, you can specify to show only the Java compile errors from a particular Java class or for the particular project. An example of the Tasks view with a Java code error is shown in Figure 2-5.



Tasks (Filter showing 1 of 34 items)

	C	!	Description	Resource	In Folder
⊗			ActionForm cannot be resolved (or is not a valid type) ...	LoginAction.java	ItsoMyTradeWeb/Java Sour

Figure 2-5 Tasks view

Double-clicking an error opens the file with the error at the point of the error.

Console view

The Console view displays console output of Java programs that you run or messages of servers that you start.

Other views

There are many other views in the different perspectives, tailored for certain user tasks. Some of the other views will be explained when we discuss other perspectives.

Customizing perspectives

You can highly customize the different perspectives by:

- Closing or opening views.
- Maximizing the view by double-clicking the title bar. You do this when you need a large pane for code editing. Double-click again to restore the layout.
- Moving views to other panes or stack them behind other views. To move a view:
  - Select the view's title bar and start dragging the view.
  - While you drag the view, the mouse cursor changes into a drop cursor. The drop cursor indicates what will happen when you release the view you are dragging:

⬇	The floating view appears below the view underneath the cursor.
⬅	The floating view appears to the left of the view underneath the cursor.
➡	The floating view appears to the right of the view underneath the cursor.
⬆	The floating view appears above the view underneath the cursor.
🗂	The floating view appears as a tab in the same pane as the view underneath the cursor. You can also drop the view on the perspective toolbar to make it a fast view.
🚫	You cannot dock the floating view at this point.

- Adding views and icons. You can add a view or a set of icons through:
  - Select *Window -> Customize Perspective* from the main menu bar.
  - Select the views you want to add and the icons (*Other*) you want to add and click *OK*.
  - Select *Window -> Show View* and select the view you just added.

Once you have configured the perspective to your liking, you can also save it as your own perspective by selecting *Window -> Save Perspective As.*

When you want to reset a perspective to its original state, select *Window -> Reset Perspective* from the main menu.

New icon

The icon (*New*) opens a dialog where you can create any type of resource from a selection list. This dialog is also opened when selecting *File -> New -> Other* from the main tool bar (Figure 2-6).

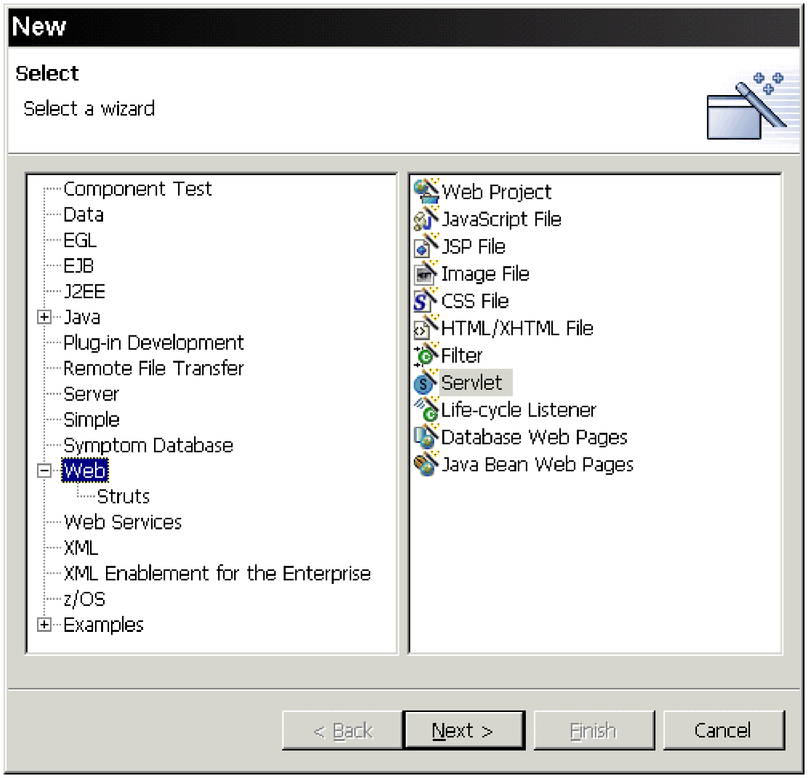


Figure 2-6 New wizard

You can also select the drop-down menu next to the New icon. This action displays a list of most used resources that you may want to create in the current perspective. If the desired resource is not in the list, select *Other* to open the New dialog.

Web perspective

In Figure 2-3 on page 16 you see the Workbench opened in the Web perspective. You use the Web perspective when you want to develop Web applications. The Web perspective is the best perspective to add and organize static content (HTML, images) and dynamic content (servlets and JSPs) to a Web application.

On top of the perspective, you see the Workbench toolbar. The contents of the toolbar change based on the active editor for a particular resource. The current editor is the page designer for editing our JSP page. The toolbar now reflects JSP development and contains icons to add JSP tags and a JSP menu item.

The Outline view shows the outline of a JSP page. It contains all the tags from which the JSP page is constructed. When you switch to the source tab of the page designer and you select a tag in the Outline view, the matching line in the Source view is highlighted.

We use the Web perspective in the chapters that follow, where we develop the sample Web application and the Web services.

Web Structure view

The Web Structure view (Figure 2-7) shows the logical layout of a Web application with Web pages and actions. This is most useful in a Struts-based Web application:

- Web pages are shown with referenced files and Struts actions.
- Struts actions are shown with forms and action mappings.

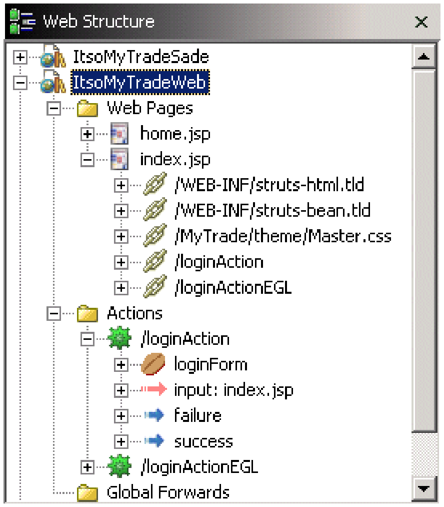
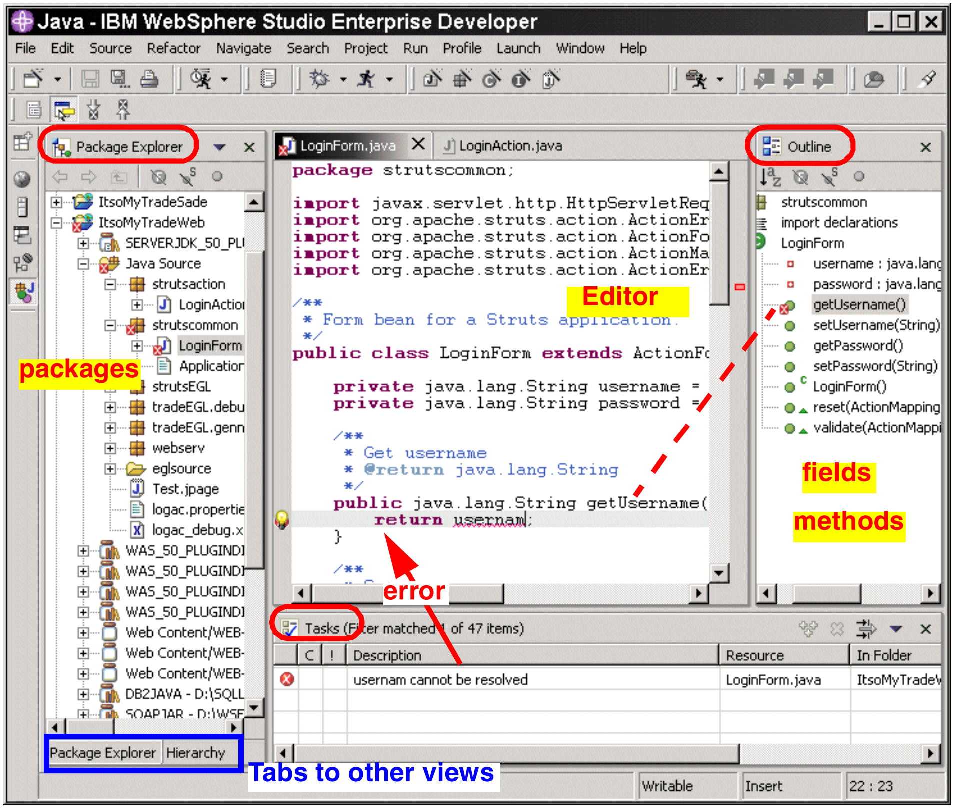


Figure 2-7 Web Structure view

Java perspective

When you want to develop Java applications, you use the Java perspective. The Java perspective is shown in Figure 2-8. It contains a lot of useful editors and views which help you in your Java development.



```
package strutscommon;

import javax.servlet.http.HttpServletReq
import org.apache.struts.action.ActionEr
import org.apache.struts.action.ActionFo
import org.apache.struts.action.ActionMa
import org.apache.struts.action.ActionEr

/**
 * Form bean for a Struts application.
 */
public class LoginForm extends ActionFo

    private java.lang.String username =
    private java.lang.String password =

    /**
     * Get username
     * @return java.lang.String
     */
    public java.lang.String getUsername(
        return usernam;
    }

    /**
```

Tasks (Filter matched 1 of 47 items)

C	!	Description	Resource	In Folder
		username cannot be resolved	LoginForm.java	ItsoMyTradeW

Figure 2-8 Java perspective

You navigate in the Java perspective through the **Package Explorer** view, which enables you to define and manage Java packages and the Java classes defined in the packages.

When you select a Java class in the Packages view and select *Navigate -> Open Type Hierarchy*, the **Hierarchy** view for that Java class opens. The Hierarchy view allows you to see the full hierarchy of a Java class. In Figure 2-8, the Hierarchy view is currently hidden by the Package Explorer view.

When you double-click a Java file the Java editor opens. You can open multiple Java files at the same time. The Java editor features syntax highlighting and a code assistant by pressing *Ctrl+spacebar.*

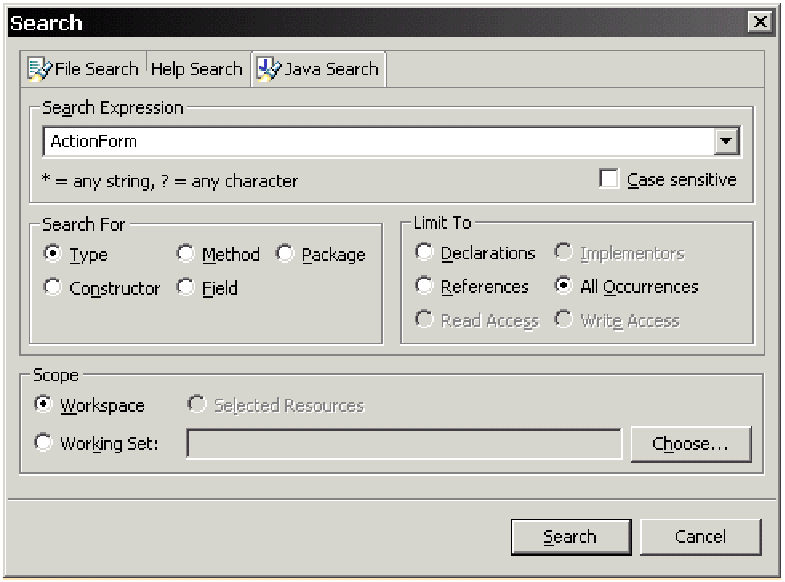
The Outline view in the Java perspective gives an overview of all the methods and fields for the Java file that is currently opened. When you click a method in the Outline view, the cursor is positioned in the method signature in the Java editor. The tool bar at the top contains filters to include or exclude static methods or fields, and to sort the Outline view.

In the Java perspective, the Workbench toolbar contains several icons to add new packages, new Java classes, new Java interfaces, or to create a new Scrapbook page.

Search

Clicking the search icon invokes the search dialog as shown in Figure 2-9. Now you can either do a full text search, or a more intelligent Java search, to look, for example, for a particular type declaration or references to it.

The **Search** view (Figure 2-10) shows the results of a search action. From the search view you can double-click any of the result lines to open the class that contains the declaration or reference.



Search

File Search | Help Search | Java Search

Search Expression

ActionForm

* = any string, ? = any character

☐ Case sensitive

Search For

● Type ○ Method ○ Package

○ Constructor ○ Field

Limit To

○ Declarations ○ Implementors

○ References ● All Occurrences

○ Read Access ○ Write Access

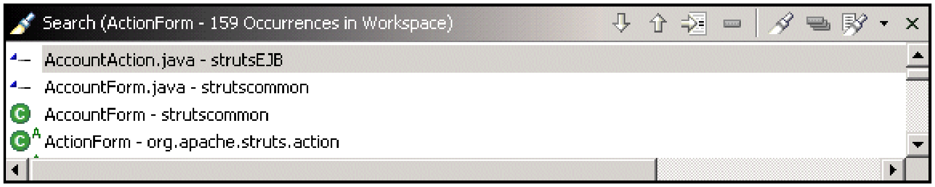
Scope

● Workspace ○ Selected Resources

○ Working Set: Choose...

Search | Cancel

Figure 2-9 Search dialog



Search (ActionForm - 159 Occurrences in Workspace)

AccountAction.java - strutsEJB

AccountForm.java - strutscommon

AccountForm - strutscommon

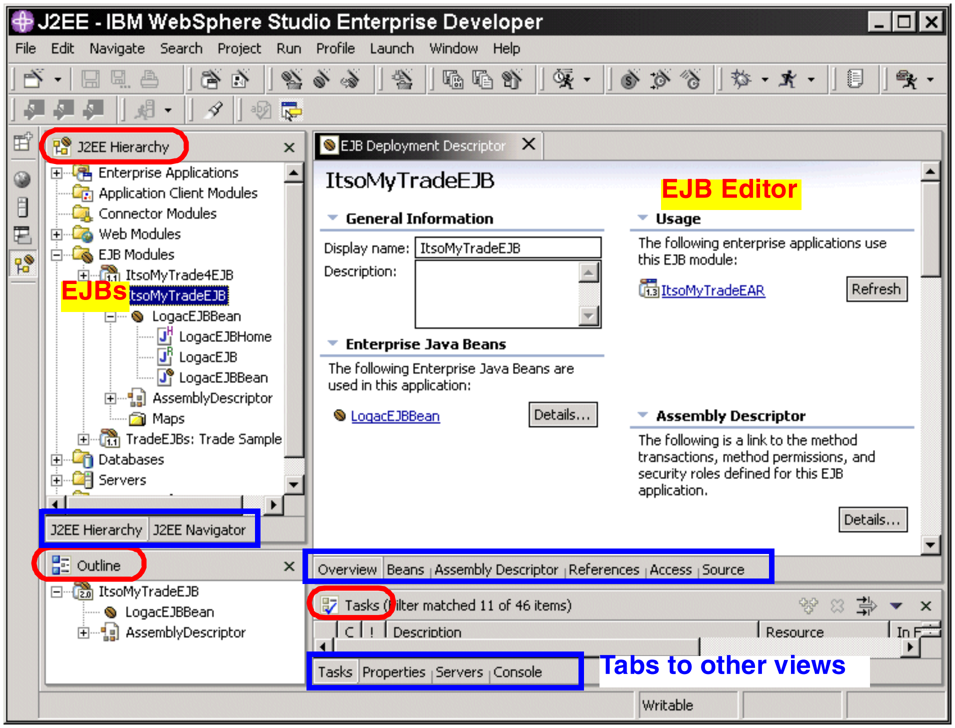
ActionForm - org.apache.struts.action

Figure 2-10 Search results

J2EE perspective

The J2EE perspective provides useful views for the J2EE or EJB developer. The **J2EE Hierarchy** view in Figure 2-11 shows you a list of all the different modules such as Web modules, EJB modules or servers and configurations that make up your enterprise application. You can expand the module you want to explore and you can edit the associated deployment descriptors for that module by double-clicking.

In Figure 2-11, the EJB deployment descriptor (`ejb-jar.xml`) is currently opened in the EJB editor.



ItsoMyTradeEJB

General Information

Display name: ItsoMyTradeEJB

Description:

Usage

The following enterprise applications use this EJB module:

ItsoMyTradeEAR

Enterprise Java Beans

The following Enterprise Java Beans are used in this application:

LogacEJBBean

Assembly Descriptor

The following is a link to the method transactions, method permissions, and security roles defined for this EJB application.

Figure 2-11 J2EE perspective

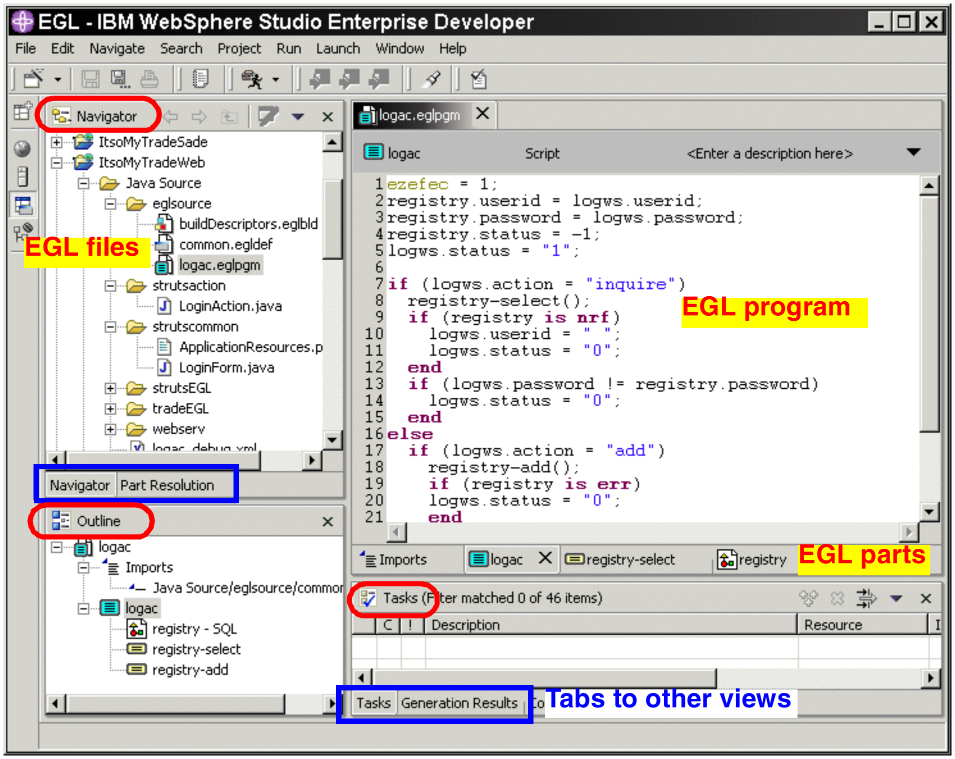
The **J2EE Navigator** view, hidden by the J2EE Hierarchy view in Figure 2-11, shows a hierarchical view of all the resources in the workspace. When you double-click a resource, the registered editor for that file extension opens and the Outline view shows the outline for the file you are editing.

EGL perspective

As part of Enterprise Developer, IBM is also introducing enterprise generation language (EGL), a high-level programming language (based on IBM's VisualAge Generator product). EGL is a fourth-generation programming language that enables traditional developers to code model aspects of an application at a high level and then generate the appropriate source code for targeted run-time environments.

From the visual assembly environment (Figure 2-22 on page 41), a traditional developer can choose to implement a particular action using EGL. Then, using the **EGL part editor** and the associated scripting language (optimized for rapid application development), developers can create programs, functions, records, and other structures, and then generate COBOL or Java source code as needed. The EGL tooling also includes task wizards to help developers quickly create the parts they need.

Figure 2-12 shows the EGL perspective with an EGL part opened in the editor.



```
ezefec = 1;
registry.userid = logws.userid;
registry.password = logws.password;
registry.status = -1;
logws.status = "1";

if (logws.action = "inquire")
  registry-select();
  if (registry is nrf)
    logws.userid = " ";
    logws.status = "0";
  end
  if (logws.password != registry.password)
    logws.status = "0";
  end
else
  if (logws.action = "add")
    registry-add();
    if (registry is err)
      logws.status = "0";
    end
```

Figure 2-12 EGL perspective with EGL part editor

The EGL language is similar to COBOL and PL/I, providing the developers with those skills sets the ability to quickly generate applications that can run in either the middle-tier (Java) or enterprise information systems (EIS usually implemented in COBOL/CICS) of a Web application. This capability provides the developer with options when they want to write code once and then have it potentially run in multiple environments (through COBOL or Java generation as needed).

Moreover, EGL provides traditional developers with an easy way to create new back-end COBOL applications, and the associated Java wrappers (and their connectors) needed in the middle-tier. This capability helps development teams bridge the middle-tier problem. Because the Java wrappers (and associated connectors) are created automatically, Java programmers are not required to develop wrappers for COBOL applications they did not create. The wrappers are created automatically and can be directly leveraged in the visual assembly environment.

EGL source code is contained in a resource called an EGL file. An EGL file contains parts, such as programs, functions, structures, records, and data items.

Creating EGL files with EGL parts is described in Chapter 8, “Implementing EGL actions” on page 171.

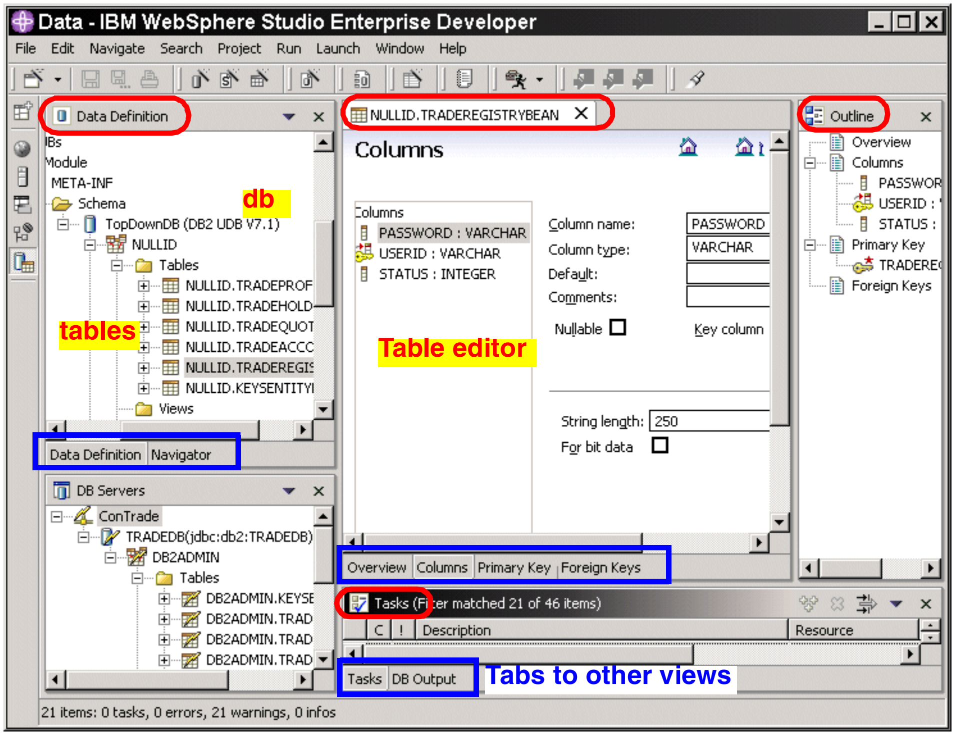
Data perspective

You use the Data perspective (Figure 2-13) for relational database design for your application. You can either create a relational database schema yourself, or import it from an existing database. Afterwards, you can browse, query or modify it. The data perspective provides the views to manage and work with database definitions.

In the **DB Servers** view, you can create a connection to an existing database and browse its schema. When you want to modify or extend the schema, you have to import it into the Data Definition view.

The **Data Definition** view allows you to define new tables, or to modify existing tables. If you double-click a table in the Data Definition view, the table editor opens and you can add or change columns and primary or foreign keys.

The Navigator view shows all the resources in the folder structure.



Column name: PASSWORD
Column type: VARCHAR
Default:
Comments:
Nullable □ Key column
String length: 250
For bit data □

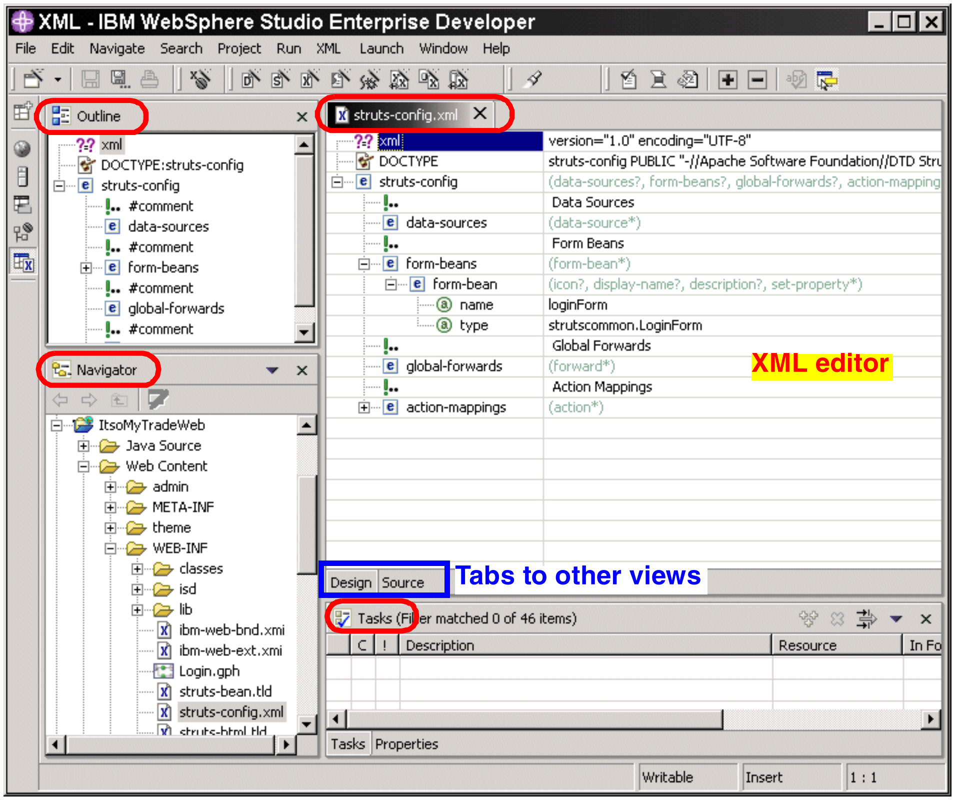
21 items: 0 tasks, 0 errors, 21 warnings, 0 infos

Figure 2-13 Data perspective

XML perspective

The XML perspective is the perspective for XML development. The XML perspective contains several editors and views that help you in building XML, XML schemas, XSD, DTD, and integration between relational data and XML.

In Figure 2-14, the XML editor is opened on a Struts configuration file. You can switch between the Design and Source tabs of the editor to develop your XML file. The Outline view contains all the XML tags that make up the XML document that is currently opened in the XML editor.



xml	version="1.0" encoding="UTF-8"
DOCTYPE	struts-config PUBLIC "-//Apache Software Foundation//DTD Stru
struts-config	(data-sources?, form-beans?, global-forwards?, action-mapping
	Data Sources
data-sources	(data-source*)
	Form Beans
form-beans	(form-bean*)
form-bean	(icon?, display-name?, description?, set-property*)
name	loginForm
type	strutscommon.LoginForm
	Global Forwards
global-forwards	(forward*)
	Action Mappings
action-mappings	(action*)

Figure 2-14 XML perspective

Server perspective

When you want to test a Web application or EJB module, you need the Server perspective (Figure 2-15). The server perspective contains views and editors that enable you to define, configure, and manage server instances and configurations.

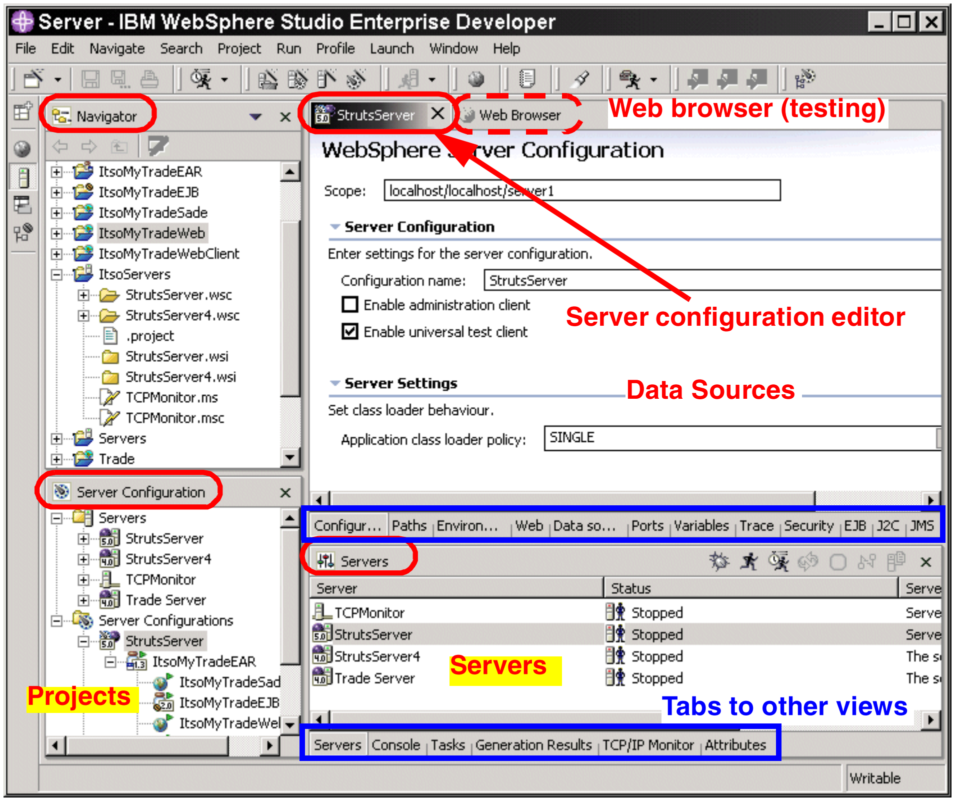
The **Server Configuration** view (left bottom) enables you to define or modify server instances and configurations, and bind them to a project. When you double-click the server configuration file in the Server Configuration view, the Server Configuration editor opens.

The **Servers** view (right bottom) lists all the currently defined server instances. Here you can start or stop their execution, or assign another server configuration to a server instance.

The **Console** view (currently hidden by the Servers view) shows all the output listed by a running server instance.

The **Debug** view allows you to step through the code when debugging. This view is not opened when running in normal mode.

The **Variables** view allows you to inspect the values of variables when debugging. This view is not opened when running in normal mode.



Scope: localhost/localhost/server1

Server Configuration

Enter settings for the server configuration.

Configuration name: StrutsServer

☐ Enable administration client

☑ Enable universal test client

Server Settings

Set class loader behaviour.

Application class loader policy: SINGLE

Server	Status	Serve
TCPMonitor	Stopped	Serve
StrutsServer	Stopped	Serve
StrutsServer4	Stopped	The s
Trade Server	Stopped	The s

Web browser (testing)

Server configuration editor

Data Sources

Servers

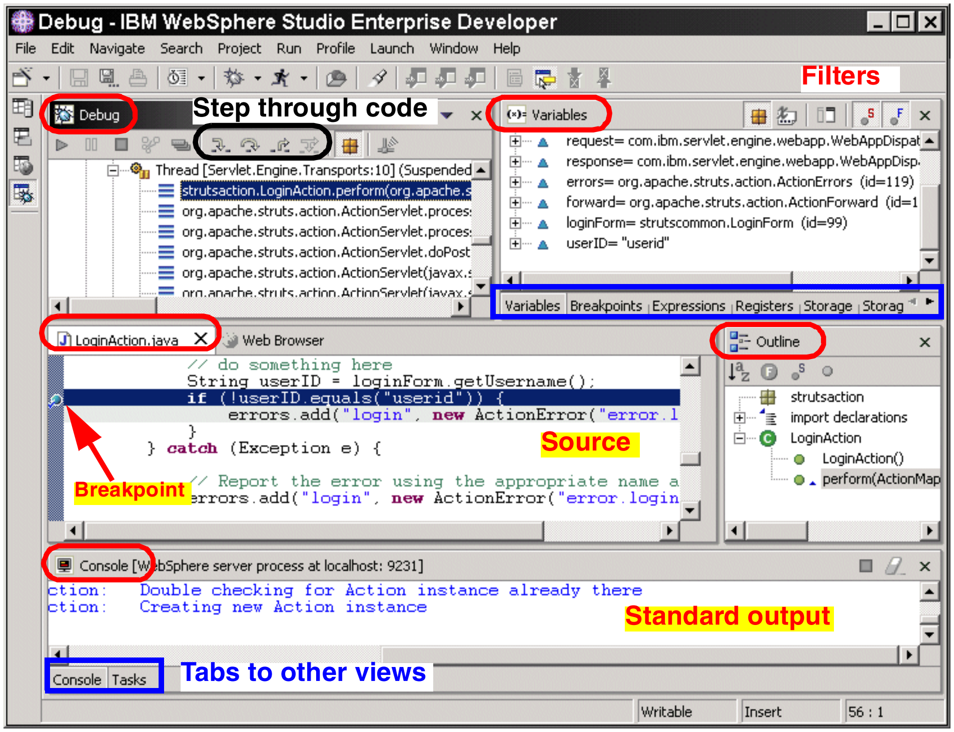
Projects

Tabs to other views

Figure 2-15 Server perspective

Debug perspective

Use the Debug perspective (Figure 2-16) when you want to debug your code. The Debug perspective automatically opens when you click the *Debug* icon ☼ in the Java perspective to run an application in debug mode. It allows you to step through your code, inspect the values of variables, modify your code and resume execution.


Debug - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Run Profile Launch Window Help
Filters
Debug
Step through code
Variables
LoginAction.java
Web Browser
Outline
Source
Breakpoint
Console
Standard output
Tabs to other views
Variables Breakpoints Expressions Registers Storage Storag
Console Tasks
Writable
Insert
56 : 1

```
Thread [Servlet.Engine.Transports:10] (Suspended
    strutsaction.LoginAction.perform(org.apache.s
    org.apache.struts.action.ActionServlet.proces
    org.apache.struts.action.ActionServlet.proces
    org.apache.struts.action.ActionServlet.doPost
    org.apache.struts.action.ActionServlet(javax.s
    org.apache.struts.action.ActionServlet(javax.s
```

```
request= com.ibm.servlet.engine.webapp.WebAppDispat
response= com.ibm.servlet.engine.webapp.WebAppDisp.
errors= org.apache.struts.action.ActionErrors  (id=119)
forward= org.apache.struts.action.ActionForward  (id=1
loginForm= strutscommon.LoginForm  (id=99)
userID= "userid"
```

```
            // do something here
            String userID = loginForm.getUsername();
            if (!userID.equals("userid")) {
                errors.add("login", new ActionError("error.l
            }
        } catch (Exception e) {

            // Report the error using the appropriate name a
            errors.add("login", new ActionError("error.login
```

```
strutsaction
import declarations
LoginAction
    LoginAction()
    perform(ActionMap
```

```
Console [WebSphere server process at localhost: 9231]
ction:   Double checking for Action instance already there
ction:   Creating new Action instance
```

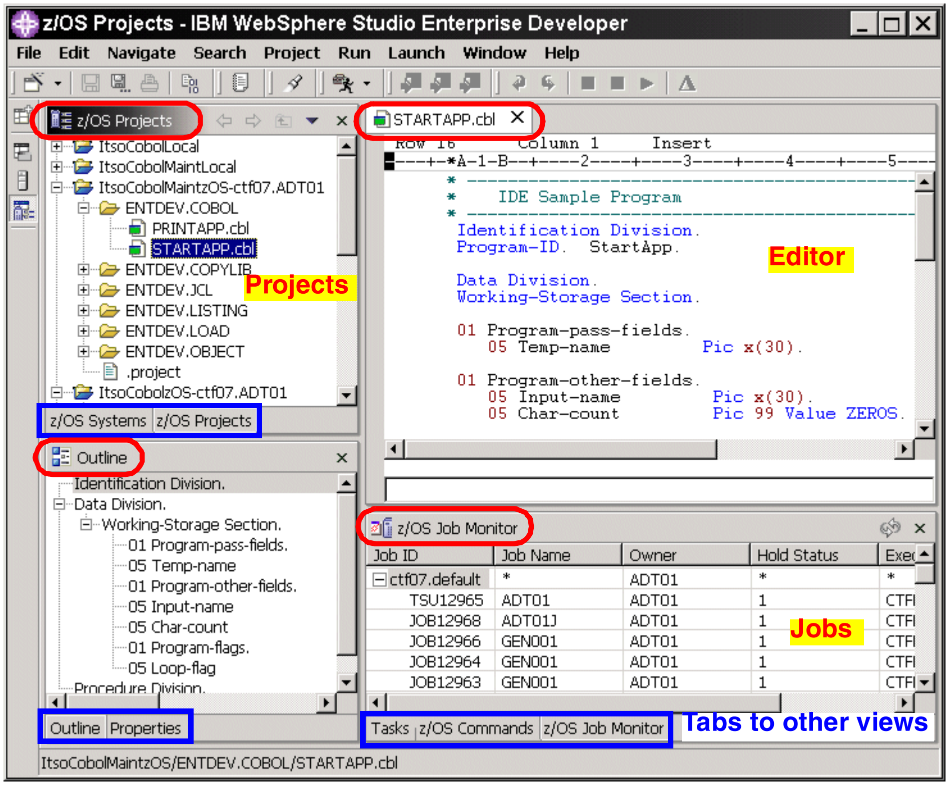
Figure 2-16   Debug perspective

The Debug perspective is built from the following views:

- The **Debug view** lists all threads within the different processes and shows you where the execution is halted when reaching a breakpoint.
- Beneath the Debug view there is a Java editor that shows the source of the code you are stepping into.
- The **Breakpoint view** lists all currently defined breakpoints. The *Exception* icon on top of the Breakpoint view allows you to define exceptions that will halt execution when thrown.
- The **Variables view** lists all variables defined currently in the running thread. You can view and modify their values and set up filters to exclude for example static fields.
- In the **Expressions view** you can enter Java code and execute it using all the variables that are visible at the current breakpoint.
- The **Console view** (bottom) shows the output of your application.

z/OS Projects perspective

The z/OS Projects perspective provides views tailored for connecting to a z/OS system and for developing z/OS applications (Figure 2-17).



```
 Row 16       Column 1      Insert
----+-*A-1-B--+----2----+----3----+----4----+----5----
      * ----------------------------------------------
      *    IDE Sample Program
      * ----------------------------------------------
       Identification Division.
       Program-ID.  StartApp.

       Data Division.
       Working-Storage Section.

       01 Program-pass-fields.
          05 Temp-name           Pic x(30).

       01 Program-other-fields.
          05 Input-name           Pic x(30).
          05 Char-count           Pic 99 Value ZEROS.
```

Job ID	Job Name	Owner	Hold Status	Exec
ctf07.default	*	ADT01	*	*
TSU12965	ADT01	ADT01	1	CTFI
JOB12968	ADT01J	ADT01	1	CTFI
JOB12966	GEN001	ADT01	1	CTFI
JOB12964	GEN001	ADT01	1	CTFI
JOB12963	GEN001	ADT01	1	CTFI

Figure 2-17 z/OS Projects perspective

Projects for z/OS development can be local projects or remote projects. Remote projects keep the data on the z/OS system. Editors are provided for COBOL, PL/I, and other languages.

The **z/OS Projects** view shows the list of projects. Remote data sets are mapped to local folders that show the members within the data set.

The **z/OS Systems** view (not shown) is used to connect to the z/OS system.

The **z/OS Job Monitor** view shows the remote jobs and can be used to retrieve job output.

The **z/OS Commands** view is used to issue TSO commands to the remote system.

z/OS Systems perspective

The z/OS Systems perspective provides a subset of the z/OS Projects perspective, tailored to connect to z/OS systems and configure the interface.

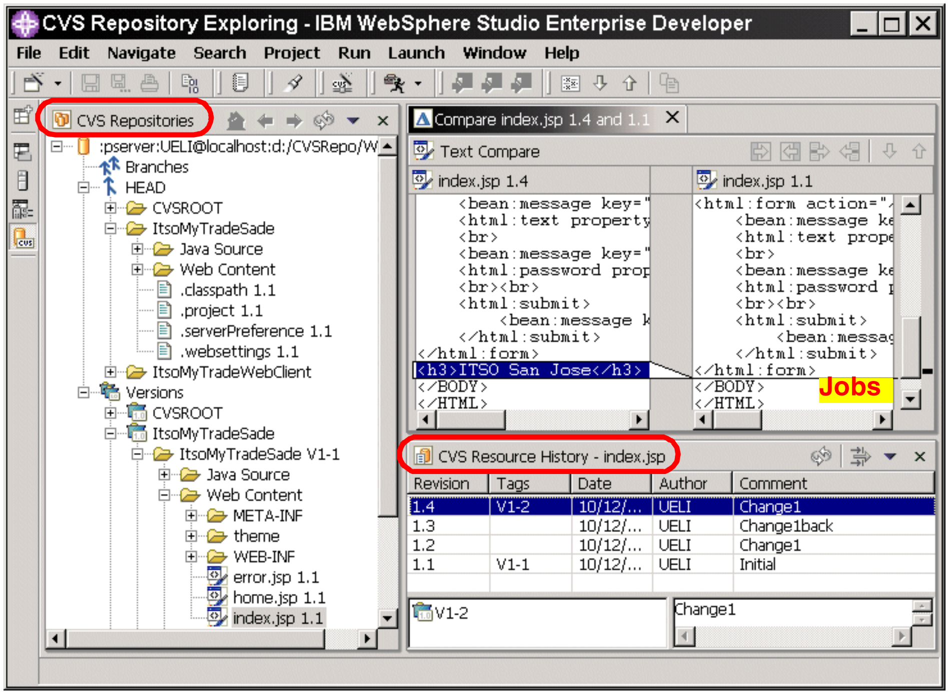
The **z/OS Directories** view is used to define high-level qualifiers that are used to access z/OS data sets.

The **z/OS File Extension Mappings** view defines the mapping between local file extensions and z/OS data sets, including how the files are transferred.

Figure 14-20 on page 371 shows the z/OS Systems view and Figure 14-21 on page 372 shows the z/OS Directories and z/OS File Extension Mappings views.

CVS Repository Exploring perspective

The CVS Repository Exploring perspective provides an interface to the Concurrent Versions System (CVS), which is one of the supported products for team development (Figure 2-18).



```
    <bean:message key="
    <html:text property
    <br>
    <bean:message key="
    <html:password prop
    <br><br>
    <html:submit>
        <bean:message k
    </html:submit>
</html:form>
<h3>ITSO San Jose</h3>
</BODY>
</HTML>
```

```
<html:form action="/
    <bean:message ke
    <html:text prope
    <br>
    <bean:message ke
    <html:password p
    <br><br>
    <html:submit>
        <bean:messag
    </html:submit>
</html:form>
</BODY>
</HTML>
```

Revision	Tags	Date	Author	Comment
1.4	V1-2	10/12/...	UELI	Change1
1.3		10/12/...	UELI	Change1back
1.2		10/12/...	UELI	Change1
1.1	V1-1	10/12/...	UELI	Initial

V1-2

Change1

Figure 2-18 CVS Repository Exploring perspective

The **CVS Repositories** view displays connections to repositories, the projects that have been shared with team members, either as a branch (current code) or as versions (frozen code).

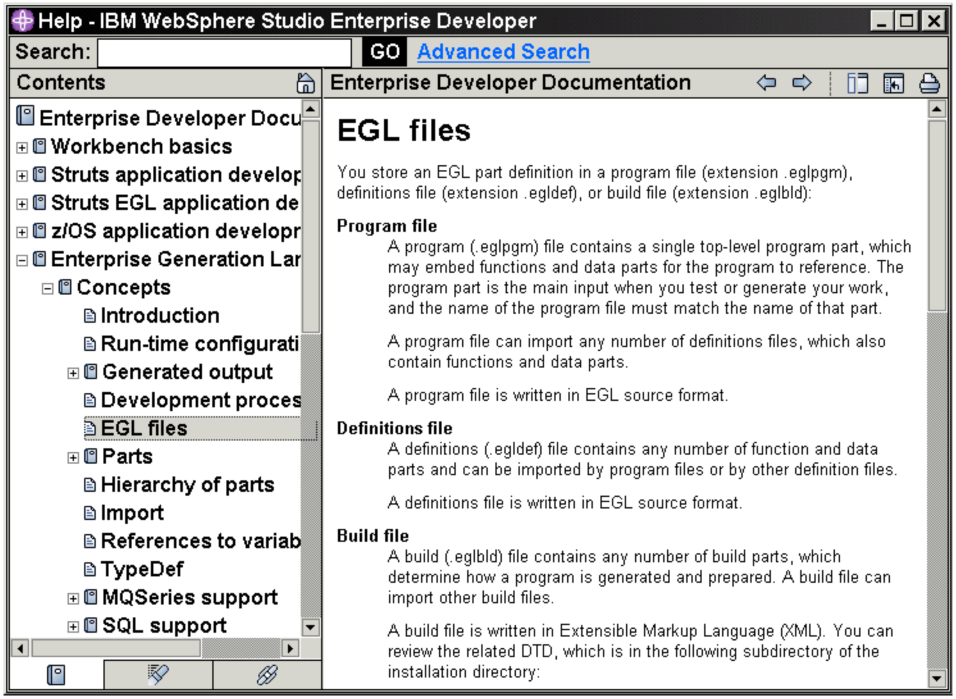
The **CVS Resource History** view shows the revisions that have been performed on a file.

Revisions of files can be compared and the differences are shown in the **Compare** view.

We describe simple usage procedures for CVS in Appendix A, “Team development” on page 427.

Help

The Enterprise Developer provides help in a separate window that you can open using *Help -> Help Contents* (Figure 2-19).



EGL files

You store an EGL part definition in a program file (extension .eglpgm), definitions file (extension .egldef), or build file (extension .eglbld):

Program file

A program (.eglpgm) file contains a single top-level program part, which may embed functions and data parts for the program to reference. The program part is the main input when you test or generate your work, and the name of the program file must match the name of that part.

A program file can import any number of definitions files, which also contain functions and data parts.

A program file is written in EGL source format.

Definitions file

A definitions (.egldef) file contains any number of function and data parts and can be imported by program files or by other definition files.

A definitions file is written in EGL source format.

Build file

A build (.eglbld) file contains any number of build parts, which determine how a program is generated and prepared. A build file can import other build files.

A build file is written in Extensible Markup Language (XML). You can review the related DTD, which is in the following subdirectory of the installation directory:

Figure 2-19 Help window

The Help window contains a lot of useful information about the Workbench. It provides information about the different concepts used by the Workbench, the different Tasks you can do within the Workbench and some useful samples. The Search field allows you to do a search in the help.

Memory considerations

After working with Enterprise Developer for some time, you will have opened several perspectives. You might have the impression that Enterprise Developer is working slower. It is good practice to close down the perspectives you have not been using for a while, because they can consume a lot of memory, and hence, slow down the overall performance.

Projects

A project is the top-level construct for organizing the different resources. It contains files as well as folders. In the Workbench you can create different kinds of projects, and they will have a different structure. A Web project, for example, has a different nature from a Java project; therefore it will have a different folder structure.

We will now briefly discuss the types of projects referred to in this document:

- Java project
- EAR project
- Web project with Struts
- EJB project
- Server project
- z/OS Local project
- z/OS MVS project

Java project

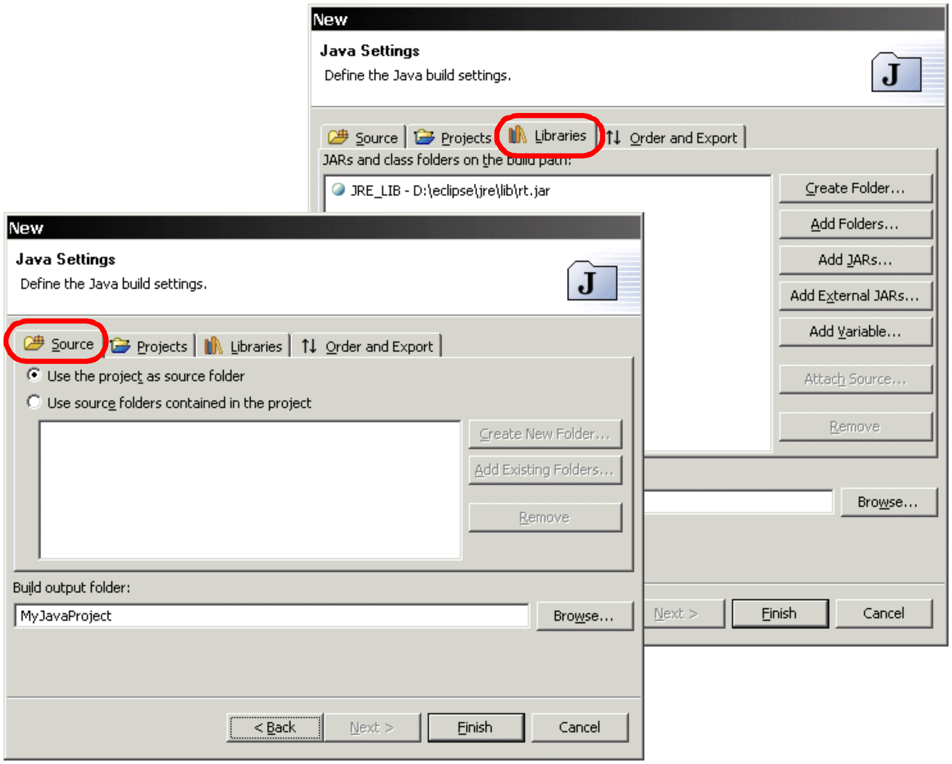
When you want to create a Java application, you first have to create a Java project to contain the Java files. Each Java project has a Java builder and builder path associated with it, which are used to compile the Java source files.

Creating a Java project

Here are the steps to create a Java project:

- Select *File -> New -> Project*.
- In the New dialog select *Java* in the left pane and *Java Project* in the right pane. Click *Next*.

- Specify a name for the project and the location of the project contents. By default the content is stored in the workspace. Click *Next*.
- The *Java build settings* dialog (Figure 2-20) contains four tabs to specify the folders, projects, and libraries used for compilation.



New

Java Settings

Define the Java build settings.

JARs and class folders on the build path:

JRE_LIB - D:\eclipse\jre\lib\rt.jar

New

Java Settings

Define the Java build settings.

● Use the project as source folder

○ Use source folders contained in the project

Build output folder:

MyJavaProject

Figure 2-20 Java build settings

Source In the *Source* tab you specify where the source files should be stored, either within the normal project folders or in folders designated by you.

Projects In the *Projects* tab you specify whether other projects are required in the build path. For example, a Web project may require an EJB project.

Libraries In the *Libraries* tab you can add internal and external JAR files to the build path:

- An internal JAR is contained in the Workbench. The advantage of an internal JAR is that it can be treated like a normal resource within the Workbench, which allows version management.

- External JARs are referenced with absolute paths in the file system. This may make it difficult to share them in a team environment. Variables can be used to alleviate the issue introduced by absolute paths. An example of an external JAR file would be the `db2java.zip` file used for JDBC access to DB2.

Order/Export The *Order and Export* tab enables you to specify the order in which the different items in the build path are accessed, when loading Java classes. You can also select which directories and files are exported with the project.

> **Tip:** Use the predefined variables instead of adding external JARs with absolute paths to your build path whenever possible. The Workbench contains various predefined variables such as the `DB2JAVA` variable, which defines the `db2java.zip` file. You can add variables for other JAR files through the *Window -> Preferences -> Java -> Classpath Variables* dialog.

You can modify the Java build path after you have created a project through the *Properties* context menu of the project.

When you are finished creating the Java project, the Workbench switches automatically to the Java perspective.

Creating a package and a class

To create a Java package select *File -> New -> Java Package* or click the *New Package* icon in the toolbar. Enter the package name, click *Finish*. The package appears in the Packages view.

To create a class in the new package, select the package and select *File -> New -> Java Class* or select the *New Class* icon in the toolbar. In the SmartGuide, check the package name and enter the desired class name and superclass. If you want a *main* method in your class, select the *main* method under *Which method stubs would you like to create.* Click *Finish*. The class appears under the package and a Java editor opens and you can start coding.

Java editing

The following useful features are available when you edit Java code:

- Double-clicking in the title bar of the Java editor maximizes the editor so that it occupies the whole perspective. Double-click again to restore its original size.
- Use *Ctrl+spacebar* to launch the code assist in the Java editor when coding.

- If you select the edited Java source and you click the *Show Source of Selected Element Only* icon in the toolbar then only the source of the selected element in the Outline view is displayed in the editor.
- If you place the cursor in the Java editor on a variable then the full package name of that variable displays in the hover help (a small pop-up that opens at the cursor location over the text). Hide the hover help by clicking the *Text Hover* icon in the toolbar .
- If you select a method in the Outline view and then select *Replace from Local History* from the context menu, a dialog opens and shows all the states of the method that you saved. You can replace the method with an older version. The same can be done for the class itself from the Navigator view.

EAR project

To develop a J2EE enterprise application you have to create an enterprise application project (EAR project). An EAR project usually consists of one or more EJB modules, one or more Web modules (Web applications), and one or more application client modules.

Creating an EAR project

To create an EAR project do the following:

- Select *File* -> *New Project*.
- Select *J2EE* in the left pane and *Enterprise Application Project* in the right pane, click *Next*.
- Specify a *Name* for the EAR project.
- Specify contained modules (client, EJB, and Web projects) that you want to include in the EAR project, and click *Finish*.
- We recommend that you follow a naming standard for your projects.

EAR deployment descriptor (application.xml)

When you create an EAR project, a deployment descriptor (`application.xml`) is created in the `/META-INF` folder. The EAR deployment descriptor defines all modules in the EAR file.

To open the EAR deployment descriptor do the following:

- Open the J2EE perspective and J2EE Hierarchy view.
- Expand *Enterprise Applications* and double-click the EAR project. Alternatively double-click the `application.xml` file in the Navigator view.

J2EE packaging

An EAR project can be packaged as an enterprise archive file (EAR file). An enterprise application consists of the following modules:

- Web applications, which are packaged in `.WAR` files. The WAR file contains the resources that compose the Web application and a deployment descriptor (`web.xml`). A Web application is contained in a Web project, which we discuss in "Web project" on page 39.
- EJB modules, which are packaged in `.JAR` files. The EJB JAR file contains all the EJBs and a deployment descriptor (`ejb-jar.xml`). EJBs are contained in an EJB project, which we discuss in "EJB project" on page 42.
- Optionally we can have stand-alone client applications that use EJBs, for example. An application client is also packaged in a JAR file. The application client JAR contains all the Java classes of the application client and a deployment descriptor (`application-client.xml`).

Figure 2-21 shows how WAR files and JAR files together constitute the EAR file, which also contains the application deployment descriptor (`application.xml`).

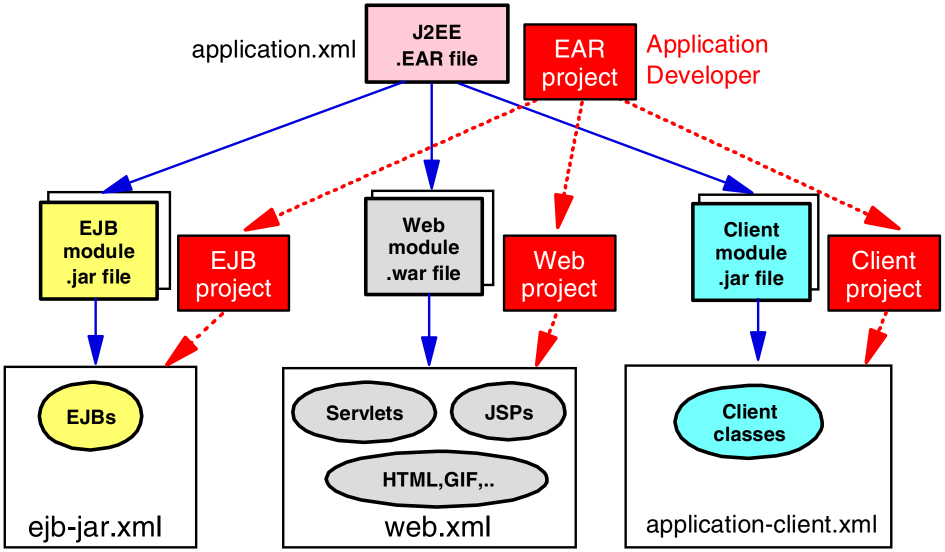


Figure 2-21 J2EE packaging

EAR deployment

Exporting an EAR project into an EAR file assembles all the components (EJB, Web, and client projects) into a `.EAR` file that can be deployed into a J2EE conforming application server, such as WebSphere Application Server.

Web project

You use a Web project when you want to create and maintain resources that compose a Web application. A Web project contains the structure of folders to contain all files that are needed to build a Web application. Typically, a Web application consists of HTML pages, images, XML, servlets, JSPs, and JavaBeans. How to build a Web application is described in “Creating a Struts application” on page 106.

Creating a Web project

To create a Web project do the following:

- Select *File* -> *New* -> *Project.*
- Select *Web* at the left side of the pane, *Web Project* at the right side of the pane, and click *Next* to start the wizard.
- Specify the Project name and the workspace location for the project. Select between a *J2EE Web project* (with servlets, JSPs, and EJBs) and a *Static Web project* (HTML only). For this document, we always create J2EE Web project. Optionally, select *Create a CSS file* (HTML style sheet).

  Here is also where you specify if you want Struts support in the J2EE Web project. See “Components of a Struts-based application” on page 67 for information on Struts.
- On the J2EE Settings page you specify the EAR project, either an existing one, or a new one. Specify the *Context root*, the alias that will be used in URLs to access project resources. Select the J2EE level, 1.3 or 1.2. Note that you can run a 1.3 project only in a WebSphere Version 5 server, but you can run a 1.2 project in either WebSphere Version 4 or Version 5.
- On the Module Dependencies page you can specify JAR files required by the Web application, for example EJB modules within the same EAR project.
- On the Struts Settings page you specify if and where the Struts resource bundle should be created. The resource bundle holds text constants that can be used in Web pages.
- Click *Finish.* Your Web project is automatically opened in the Web perspective.

When you create a new Web project, a default directory structure is created that reflects the J2EE view of a Web application. A Web deployment descriptor `web.xml` is generated in the `/webApplication/WEB-INF` folder.

Web application archive files (WAR files)

As defined in the J2EE specification, a WAR file is an archive format for Web applications. The WAR file is a packaged format of your Web application that contains all the resources (HTML, servlets, JavaBeans, JSPs, XML, XML schemas) that compose your Web application.

You can deploy a Web application by itself to an application server by creating a WAR file. Select *File* -> *Export* -> *WAR* and specify the output file and directory.

In general, however, it is easier to have the Workbench create the EAR file that contains the WAR file and deploy the EAR to an application server.

Struts

A difficult aspect of building a Web application is connecting components that comprise disparate technologies (for example, building the controller, in MVC terms).

Enterprise Developer leverages Struts, an emerging open standard for constructing MVC-based Web applications. Struts provides (among other things) an action servlet that manages the run-time relationship between JSPs and Actions. The use of Struts helps to ensure an effective separation of code responsibilities and developer roles (see “Components of a Struts-based application” on page 67 for a more detailed description of Struts).

Enterprise Developer provides a powerful visual component assembly environment, the Struts application diagram editor, for Struts-based Web applications (Figure 2-22).

The diagram editor is used to define basic flow of the Web application graphically, connecting JSPs with component services (or actions) as desired. This approach simplifies the creation of an MVC application by masking the complexity of the disparate technologies involved.

The diagram editor is used initially as part of the design process, helping a development team quickly lay out view (JSP) and model (action) components without having to consider the technical issues of combining disparate technologies that have yet to be created or harvested from existing capability. Throughout the development process the diagram editor can be used to extend and test a Web application’s capability.

As we will see later, the actions defined in the visual assembly environment can be implemented in whatever technology is most appropriate for your specific needs (COBOL, PL/I, Java, or IBM’s enterprise generation language).

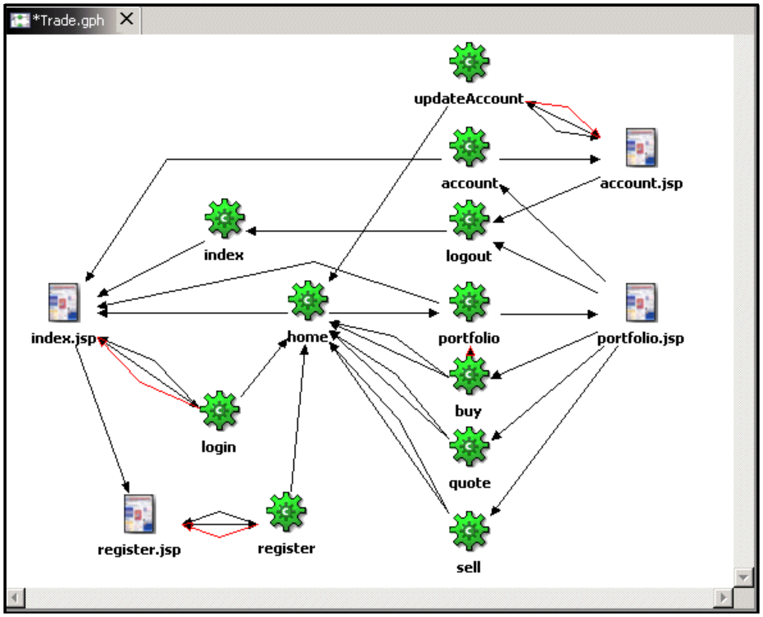


Figure 2-22   Visual assembly of Web application flow

The Enterprise Developer also provides a number of wizards to define Struts components, such as action classes, action forms, and JSPs.

Struts project

A Struts project is a special case of a Web project that includes:

- Struts taglibs and Struts configuration file (`struts-config.xml`) in the `WEB-INF` folder
- Struts run-time JAR file (`struts.jar`) in the `WEB_INF/lib` directory
- Struts resource bundle (`ApplicationResources.properties`) for the externalized user interface resources

A Struts project is defined as a Web project. The Struts support is added to the Web project during the creation of the project through the new Web project wizard.

EJB project

If you want to develop Enterprise JavaBeans (EJBs) you have to create an EJB project first. An EJB project is a logical group for organizing the EJBs. To create an EJB project:

- Select *File* -> *New* -> *Project*.
- Select *EJB* on the left pane and *EJB project* on the right pane and click *Next.*
- Specify the *Name* of the EJB project and the workspace location. You also have to specify an EAR project name that will contain your EJB project. You can select an existing EAR project or create a new one. Click *Next*.
- On the *Module Dependencies* page you can specify JAR files required by the EJB application, for example other EJB modules within the same EAR project.
- When you click *Finish*, the EJB project opens in the J2EE perspective. The deployment descriptor for the EJB module (`ejb-jar.xml`)is created in the `/YourProject/ejb-module/META-INF` folder.

The use of EJB projects is illustrated in “Generating EJB session beans from EGL” on page 238.

EJB deployment descriptor (ejb-jar.xml)

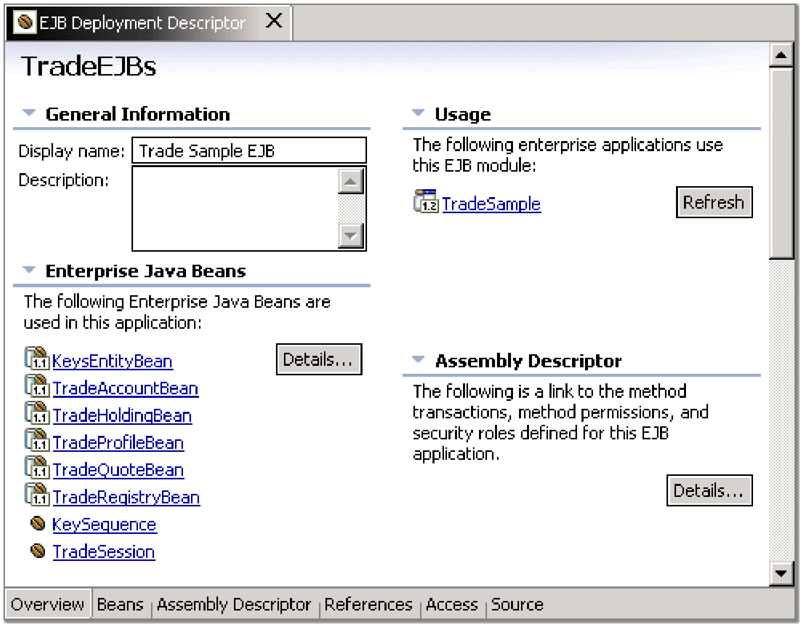
An EJB module requires a deployment descriptor (`ejb-jar.xml`) in the same way a Web application requires a deployment descriptor (`web.xml`).

In addition to the standard deployment descriptor, the Workbench also defines EJB bindings and extensions. Both binding and extension descriptors are stored in XMI files, `ibm-ejb-jar-bnd.xmi` and `ibm-ejb-jar-ext.xmi`, respectively.

EJB editor

To edit the deployment descriptor for the EJB module:

- In the *J2EE* view of the J2EE perspective, expand EJB Modules.
- Right-click the EJB module and select *Open With -> EJB Editor,* or just double-click the module*.*
- The `ejb-jar.xml` deployment descriptor opens in the EJB editor (Figure 2-23).



TradeEJBs

General Information

Display name: Trade Sample EJB

Description:

Enterprise Java Beans

The following Enterprise Java Beans are used in this application:

Usage

The following enterprise applications use this EJB module:

Assembly Descriptor

The following is a link to the method transactions, method permissions, and security roles defined for this EJB application.

Figure 2-23 EJB deployment descriptor editor

The `ejb-jar.xml` is presented in several sections defined by the tabs at the bottom of the EJB editor.

Server project

To test an EJB or Web project, you have to define a server and a server configuration to publish and run the code. Servers and server configurations are defined in server projects.

Creating a server project

To create a new server project:

- Select *File -> New -> Project*.
- Select *Server* on the left pane and *Server project* on the right pane.
- Specify a *Name* for your project and click *Finish*. We will use `ItsoServers` as our project name.

After creating a project, the Server perspective opens and you can now add a server and a server configuration.

Servers

A server identifies the server used to test your application. Unlike VisualAge for Java, Enterprise Developer has the option to deploy to and test with both local and remote instances of the WebSphere application server, and additionally Apache Tomcat. Here is a brief explanation of each of the servers.

WebSphere Version 5.0

This enables the developer to work with an integrated version of WebSphere Application Server Version 5, which supports the entire J2EE 1.3 and 1.2 programming model. This is the best option for testing EJB-based applications. Three options are provided:

- **Test Environment**—Built-in server inside the Enterprise Developer. Enterprise Developer publishes the code to the server and starts it.
- **Remote Server**—Stand-alone server on the same or another machine. When the application is executed, Enterprise Developer publishes the code to the external server and attempts to start the server using the IBM Agent Controller service, which is supplied with Enterprise Developer. This feature provides a very efficient approach to remotely deploying an application.
- **Remote Server Attach**—A server instance that will attach to a WebSphere Version 5 server that is already started.

WebSphere Version 4.0

This enables the developer to work with an integrated version of WebSphere Application Server Advanced Edition Single Server V4.0.1, which supports the entire J2EE 1.2 programming model. The same three options as for Version 5.0 are provided.

Apache Tomcat Version 4.0

Tomcat Version 4 has been developed by the Apache group on a completely separate code base from the V3.2 release, and is the reference implementation for the Servlet 2.3 and JSP 1.2 specifications. For more information on Tomcat and the Apache Jakarta project, see `http://jakarta.apache.org`. Enterprise Developer does not ship with the Tomcat binaries, only a toolkit to support its execution. You must already have a working Tomcat instance installed in order for this to work. Two options are provided:

- **Test Environment**—Runs Tomcat inside the Enterprise Developer.
- **Local Server**—Stand-alone Tomcat server on the same machine. Permits publishing and execution of the Web application to an external version of Tomcat 4.0. Unlike the WebSphere Remote Server option, this is only supported for a local instance on the same machine.

Apache Tomcat Version 3.2

This release supports the servlet 2.2 and JSP 1.1 specifications. The same two options as for Version 4.0 are provided.

Publishing Server

The publishing server supports the publishing of static Web projects, as well as J2EE projects such as EAR projects, Web projects, and EJB projects.

Static Web Server

A static Web server is a Web server that runs static Web projects. Use the static Web server for the testing of HTML and Java script files. JSPs and EJBs are not supported.

Remote Application Server Attach

This is a server instance that will attach to a generic application server that is already started.

TCP/IP Monitoring Server

This is a simple server that forwards requests and responses, and monitors test activity. This run-time environment can only be run locally, and it only supports Web projects. You cannot deploy projects to the TCP/IP Monitoring Server. The TCP/IP Monitoring Server is illustrated in “Configuring a TCP/IP monitoring server” on page 320.

Because Tomcat does not have EJB support, you cannot deploy EAR files to it, only WAR files containing servlets and JSPs.

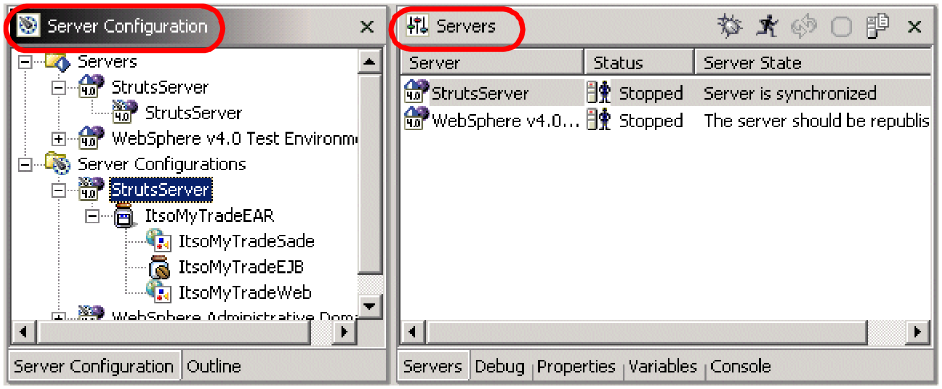
Important: Before you can do a remote unit test you have to install and run the IBM Agent Controller, which comes with Enterprise Developer, on the remote machine. IBM Agent Controller is a process that runs on the remote machine and which enables client applications to start new host processes.

Server configuration

A server configuration contains the information about the server.

The Server Configuration view of the Server perspective shows the servers, the configurations, and the projects that are assigned to a configuration. The Servers view shows the servers for start and stop operations (Figure 2-24).

A server configuration is stored in XML files in the server project. The properties can be set by opening (double-clicking) the configuration in an editor.



Server	Status	Server State
StrutsServer	Stopped	Server is synchronized
WebSphere v4.0...	Stopped	The server should be republis

Figure 2-24 Server configuration and servers

A server configuration defines:

- Port numbers for the different processes such as the naming service
- Mime types
- JDBC drivers
- Data sources
- Security enablement
- EJB test client enablement

A server configuration can be reused by multiple servers but each server will only point to one server configuration. Each server configuration can point to multiple projects as shown in Figure 2-25.

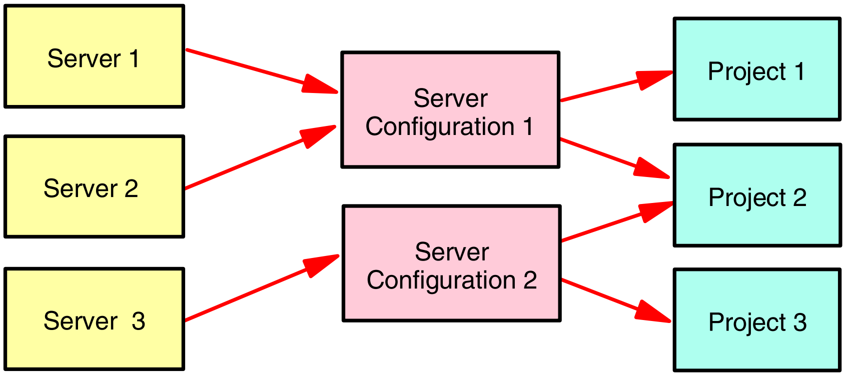


Figure 2-25 Relationship between server instance, configurations, and projects

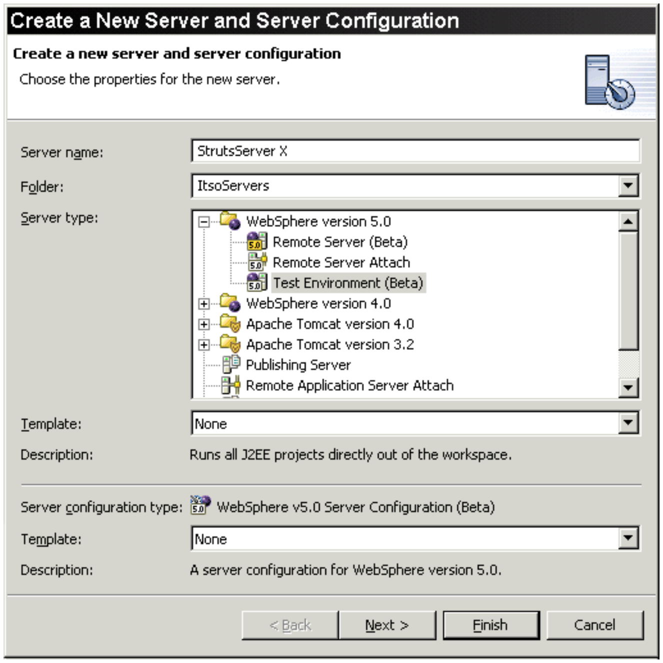
Each project has a preferred server configuration that is used when the project is run by selecting *Run on Server* from its context menu (this can be set in the project by selecting *Properties -> Server Preference*).

Creating a server and a configuration

In the server perspective, select *New -> Server -> Server and server configuration* and complete the dialog as shown in Figure 2-26.

On the next page you can set the port (default is 8080) and click *Finish*.

The new server appears in the Server perspective and you can assign EAR projects to the server. Those projects will be loaded when the server is started. To assign a project to a server, select the server configuration and select *Add -> Project* from the context menu.



Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: StrutsServer X

Folder: ItsoServers

Server type: Test Environment (Beta)

Template: None

Description: Runs all J2EE projects directly out of the workspace.

Server configuration type: WebSphere v5.0 Server Configuration (Beta)

Template: None

Description: A server configuration for WebSphere version 5.0.

Figure 2-26 Creating a server instance and configuration

Server templates

When you have to create several similar servers or configurations, you can create them from a template to save time. You could also share a template across a team so that team members can start personalizing their server configuration or instance starting from a template. To create a template:

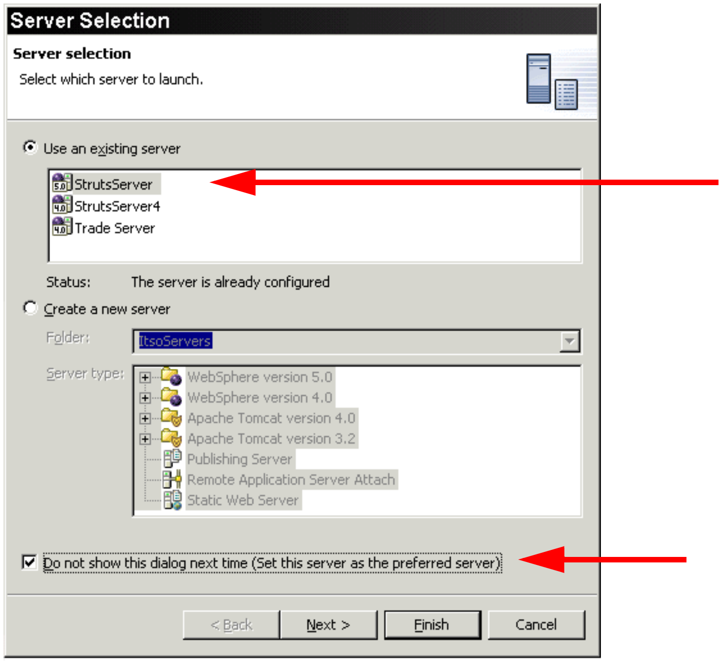
- Select *Window* -> *Preferences*.
- Expand *Server* on the left pane and select *Templates*.
- Click *Add* and specify the server to be stored as a template.

Starting and stopping a server

A server can be started from the Servers view in normal or debug mode. To debug servlets or JSPs at the source level, you must start the server in debug mode. Note that startup and execution is slower in debug mode.

A server can be started implicitly by selecting a Web resource (for example an HTML file) and *Run on Server* or *Debug on Server* from the context menu:

- The first time you are prompted to select a server (Figure 2-27). You can bypass this dialog in the future by selecting *Do not show this dialog the next time*. From then on, the server associated with the Web project is started automatically.



Server Selection

Server selection

Select which server to launch.

● Use an existing server

- StrutsServer
- StrutsServer4
- Trade Server

Status: The server is already configured

○ Create a new server

Folder: ItsoServers

Server type:

- WebSphere version 5.0
- WebSphere version 4.0
- Apache Tomcat version 4.0
- Apache Tomcat version 3.2
- Publishing Server
- Remote Application Server Attach
- Static Web Server

☑ Do not show this dialog next time (Set this server as the preferred server)

Figure 2-27 Selecting a server and making it the preferred server

Before a server is started, the associated projects are published.

When you are done with testing, you stop the server explicitly from the Servers view. When certain resources are modified (or example EJB definitions), a server must be restarted.

Publishing

Publishing means copying all the resources that are required to test a project to the right place so that the server can find them. In cases when you are testing within the Workbench, the resources might already be at the right place. However, when you are testing with WebSphere Application Server on a local or remote machine, or with Tomcat on a local machine, publishing means to copy the resources outside of the Workbench.

By default, the *Automatically publish before starting server* option is turned on. This option can be found in the *Window* -> *Preferences* -> *Server* item. When this option is turned on, all files and folders on the server are synchronized with the Workbench whenever starting or restarting that server.

You can manually publish by selecting the server in the Servers view of the Server perspective and selecting *Publish* from the context menu.

Remote server

For a remote WebSphere Application Server, you must configure how the files are transferred to the server. The Create a Server dialog prompts you for:

- The installation directory where the server is installed, for example, `C:\WebSphere\AppServer`.
- The platform (*Windows* or *Other*).
- One of two possible transfer mechanisms:
  - Copy file transfer mechanism—you must specify the remote target directory (as seen from the local machine on a LAN drive), for example, `X:\WebSphere\AppServer`.
  - FTP file transfer mechanism—you must specify the remote target directory (on the remote machine), the host name, login user ID and password, connection time-out, and optionally firewall settings.

Agent Controller

The IBM Agent Controller is a service that must be installed on the remote machine. The Agent Controller code is provided with the Enterprise Developer for all platforms supported by WebSphere Application Server.

Development for z/OS

Actions defined in the visual assembly environment may also be implemented in COBOL or PL/I. Traditional developers can create and edit host-based resources using Enterprise Developer's z/OS projects perspective. It includes the ability to connect to various host systems to locate the development resources. It also includes an editor with syntax highlighting, a job monitor to retrieve job output, and the ability to enter TSO commands.

z/OS local and remote projects are used for z/OS development. Using a remote projec,t a developer can work directly with the files stored in z/OS data sets, without having a local copy of the file. Distributed builds can be issued from the workstation to preprocess, compile, and link z/OS programs.

Chapter 14, "Developing for z/OS" on page 351 describes the development process for z/OS applications in more detail.

Connectors

A key aspect of creating a Web application involves leveraging existing applications, and harvesting components from within the enterprise. However, developers may encounter significant difficulties when they try to create components based on traditional applications. Enterprise Developer simplifies the process by providing powerful componentization tooling that helps development teams turn existing applications into reusable components.

Enterprise Developer's adapter tooling provides a wizard-based user interface that helps the developer identify important aspects of the host component. The tooling automatically creates a J2C interface to the host component. This component interface is a Java class that runs on the Web server but can communicate with transactions or other capability on the host. This component interface can then be incorporated into the visual design tool, making it available as an action within the Web application.

Note: The connector support was not available in the product when this book was written. We could only use the J2C connector in a WebSphere Application Server to connect a Struts application to an EGL-generated CICS COBOL program.

3

Sample application: Trade

This chapter describes the sample application that is used throughout the remainder of this book to illustrate the many features of WebSphere Studio Enterprise Developer. It shows you how to install the sample application, configure a WebSphere test environment, and test the sample application.

This chapter also discusses development roles that would be required in a project that aims to build a similar application as well as the design process for the sample application.

Introduction to the trade sample

The trade sample application provides an example of how heterogeneous technologies can be brought together effectively to solve a real business problem using Enterprise Developer. At its core, the trade sample application is a Struts-based Web application that includes source code from native Java, Enterprise generation language (EGL), and COBOL. It also includes examples of connector technologies used to bridge the divide between middle-tier and traditional enterprise information systems (EIS) systems.

The trade sample application includes basic functional elements that involve registering users, login, logout, a list of items that can be purchased, and the ability to buy and sell items. Such capability might be expected in any business Web site.

The Enterprise Developer trade sample application is modeled on a fictitious financial brokerage (called *TRADE*). Users can log in, change their personal account information, view their portfolio, buy and sell stocks and get stock quotes. Because this sample application is intended to provide a context for understanding the structure and capability of Enterprise Developer (and not as a primer on building Web applications, per se), the underlying capability has been greatly simplified and some complex details of building Web applications (such as embedded JavaServer Pages, user management, security, and so forth) are not covered. IBM provides numerous Redbooks that describe various aspects of building Web applications (see `http://www.redbooks.ibm.com/`).

We recommend that you install the trade sample and run it, to become familiar with its design and flow. Subsequent chapters in this book will demonstrate how to use various features of WebSphere Studio Enterprise Developer by having you build part of the login function of the trade sample application. Being familiar with how to use the application will help you understand what you are building.

Assembling a development team

Systems that bring legacy systems into the e-business world are created by multidisciplinary teams. The skills required are contributed from graphic artists, Web page designers, client- and server-side script writers, Java and host programmers.

Whether there is only one person or one hundred, the concept of the separation of roles and responsibilities is key to the successful creation and maintenance of the e-business application.

The model-view-controller pattern involves separating the tasks by each role, such as:

- The *HTML developer* uses a tool like WebSphere Page Designer to generate HTML pages and JSP templates.
- The *Script developer* uses a Java programming environment such as WebSphere Studio Enterprise Developer to edit, test, and debug servlets and JSPs.
- The *Java business logic developer* uses a Java programming environment, such as WebSphere Studio Enterprise Developer, and builders, such as the integrated EJB builder, to specify the details of the business, to access legacy applications and data, and to build the commands.
- Depending on the implementation language, the *host developer* uses a programming environment to remotely edit, compile, and debug the host application code to perform specific business function and access enterprise data. WebSphere Studio Enterprise Developer supports COBOL and PL/I in this fashion.

Further reading

A good reference for assembling a development team is the book *The Rational Unified Process, An Introduction* by Philippe Kruchten.

Development roles

In this book, for the sake of clarity, when we refer to, for instance, the senior business analyst, we mean “the person who plays the role of senior business analyst”. It might be the same person who plays the role of, say, the junior Java developer.

In considering our fictitious project analysis and design, we assume that the project team will consist of a number of persons who broadly fit into the following categories:

Business analyst This person is supposed to have a lot of business knowledge while few technical skills. He can write documents in good language. In this particular context he knows about the use case-driven approach and has basic notions of OO analysis and application development.

OO designer This person is highly skilled in IT and OO and knows most of the technical environment. He works intensively with the application architect.

Java developer

Senior developers have significant experience from working on previous projects. They are entrusted with delivering the more technically challenging and performance-critical sections of the code.

Junior developers are competent Java programmers with less experience than their more senior colleagues. The junior developers concentrate their coding efforts on the servlet and client portions of the code.

Application architect

The application architect is the technical lead on the project. He has overall responsibility for determining the high-level structure of the application, and the interaction between components. He is also a senior developer.

Web developer

Web developers are skilled in the use of HTML and the tools used to develop and maintain Web content. They are responsible for delivering HTML and JSP pages and the various interface elements such as images contained within them.

Host Developer

The host developer has knowledge of the legacy system that the e-business applications need to interact with. The Java developer needs to have a good relationship with the host developer in order to understand the interface that needs to be built between the Java components and the host components.

Web application design session

The first page of the trade sample allows the user to log into their account or register as a new customer of the trade brokerage.

After logging in, the user is presented with their personal portfolio page. From this page, they can see their current holdings, buy stocks (assuming they have enough money), sell all stocks in a particular lot, and obtain quotes. They can also access their account information (name, address, etc.) or log out.

Note: The stock symbols used in the sample are fictitious. Within the sample application, valid stock symbols are in the form of “s:1” to “s:499”.

When the user buys or sells a parcel of stock, the portfolio page is updated, as are the holdings and account balance.

From a software perspective, the trade sample application has a set of JSPs that access business logic via action mappings defined in one or more XML configuration files. The action servlet uses these action mappings (and associated action classes) to pass data and control flow between the JSPs (view) and the business logic (model). This approach helps ensure that the JSPs include no business logic, and the business logic need not be concerned with presentation issues.

Another important benefit of this approach is that Struts action classes can be used to access business logic that has been created using various technologies. In the sample application the business logic (model) portion has been implemented four different ways:

- Using native Java (via Enterprise JavaBeans)
- Using Java generated using EGL
- Using native COBOL / CICS transactions
- Using COBOL transactions generated using EGL

Note: IBM's EGL is a high-level language (based on IBM's VisualAge Generator) that allows developers to create programs and then generate Java or COBOL depending on their target platform. Java classes generated using EGL are directly accessible via Struts action classes. When EGL is used to generate CICS COBOL, appropriate connectors are used to invoke the target program.

The action classes that access CICS COBOL transactions can do so using IBM's connector technologies. Creating such connectors is a key aspect of modernizing traditional host-based applications for use as part of Web applications.

Sample application deployment topology

The trade sample application comprises JSPs, the Struts action servlet, a struts-config.xml file, a struts-config.gph file, Struts action classes, and business logic written in native Java, generated Java (via EGL), native COBOL (including the appropriate connectors), and generated COBOL (via EGL). Figure 3-1 provides a simplified view of how these elements are deployed and how they interact.

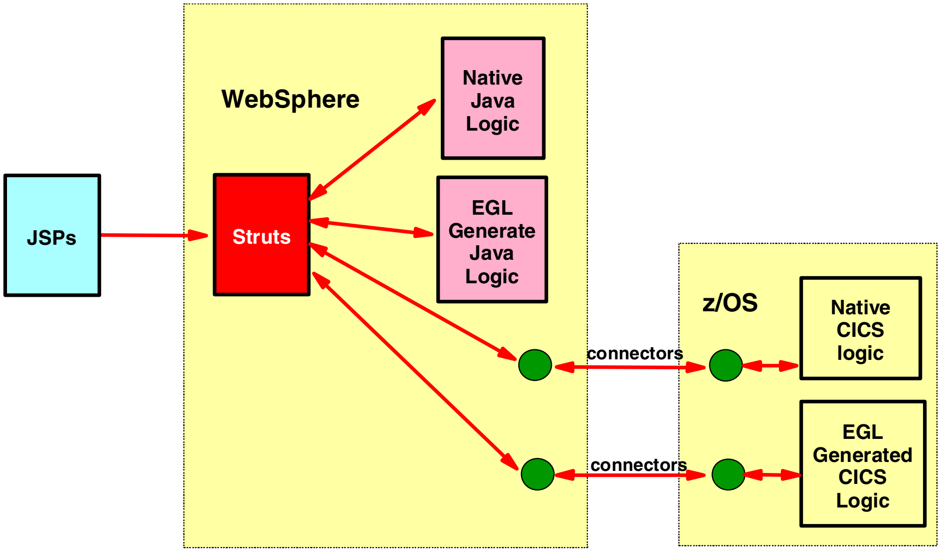


Figure 3-1 Simplified deployment topology

The JSPs reside on the WebSphere server and are presented on the client system. The Struts action servlet (and actions classes), and Java business logic also reside in the middle-tier (both native Java and EGL-generated Java). The CICS transactions reside in the S/390 (z/OS) environment (both native CICS and EGL-generated CICS). Notice that the connectors link Struts actions in the middle-tier with the associated transactions in the host environment.

The Struts-based model-view-controller design approach helps ensure that the JSPs and the business logic (regardless of how implemented) remain separate so that changes to one do not require changes to the other.

Installing the trade sample application

The trade sample application is shipped with the Enterprise Developer and can be installed using a wizard.

Prerequisites

The trade sample application uses a DB2 database. You will have to install DB2 Version 7.2 with the latest Fixpack, and know the DB2 user ID and password. You do not have to define a database; that will be done as part of the installation of the sample.

In addition, you have to make sure you use the JDBC 2.0 driver by running the command file `x:\SQLLIB\java12\usejdbc2.bat` (where *`x:\sqllib`* is the location of your DB2 installation).

Note that you have to stop all DB2 services before you run this command. Upon successful completion, a file named `x:\SQLLIB\java12\inuse` is created.

Loading the trade sample

To load the trade sample application, perform the following steps:

- Select the *New Wizard icon* (or select *File -> New -> Project*).
- In the left-hand pane, expand *Examples* and select *Trade Sample*.
- In the right-hand pane, select *All Trade Samples* then click *Next*.
- Keep the default project name (`TradeSample`) and location then click *Next*.
- Select the *Install Database using IBM DB2* radio button and enter your DB2 user ID and password (for example, `db2admin`), then click *Finish*.

Important: Remember the user ID and password you used to install the trade sample application database. You will have to specify these values when accessing the database from EGL programs.

The database will be created and populated and the sample will be imported. You should see an enterprise application named *`TradeSample`*, an EJB project named *`TradeEJBs`*, and two Web projects named *`Trade`* and *`TradeTutorial`* (Figure 3-2).

Also, you can open the DB2 Control Center and verify a database named `TRADEDB` was created and populated.

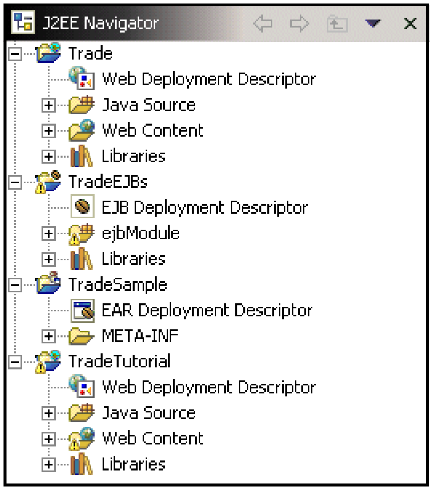


Figure 3-2 The imported trade sample application

Setting up a test server

WebSphere Studio Enterprise Developer includes a server environment in which you can run and test your applications. You must first set up a server configuration and instance before you can run any applications.

Important: A server named *Trade Server* should be defined automatically when importing the `Trade` application. If the server is not defined, follow the instructions in the `sampleguide.pdf` to define a WebSphere Version 4.0 Test Environment (the sample as shipped uses a WebSphere Version 4 server).

The trade sample application was exported with a server already configured. To see the server, perform these steps.

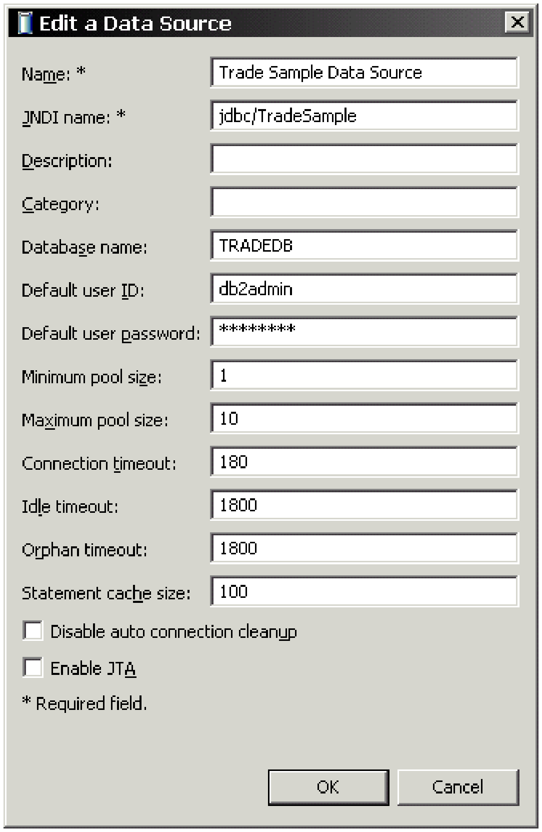
- Open a Server perspective (select *Window -> Open Perspective -> Other -> Server*).
- In the Server Configuration view (bottom left) and in the Servers view (bottom right) a server and a server configuration named *Trade Server* is visible.
- Before we start the server we verify that a data source is configured to access the `TRADEDB` database.

Defining data sources

The trade sample application uses EJBs to access the database. Before you can run the sample application, you have to define to the server the data source that the EJBs are using.

To define a data source, perform these steps:

- In the Server Configuration view, in the Server Configurations folder, open the Trade Server configuration by double-clicking it.
- Select the *Data source* tab. Select the *Db2JdbcDriver* and check the data sources configured for that driver. If the Trade Sample Data Source is configured, click *Edit*; if not, click *Add* to define the data source. The Data Source dialog should be filled as shown in Figure 3-3.



Name: *	Trade Sample Data Source
JNDI name: *	jdbc/TradeSample
Description:	
Category:	
Database name:	TRADEDB
Default user ID:	db2admin
Default user password:	*********
Minimum pool size:	1
Maximum pool size:	10
Connection timeout:	180
Idle timeout:	1800
Orphan timeout:	1800
Statement cache size:	100

□ Disable auto connection cleanup

□ Enable JTA

* Required field.

Figure 3-3 Defining a data source

- The JNDI name should be `jdbc/TradeSample` and the Database Name `TRADEDB`.

- For Default user ID enter your DB2 user ID (your actual user ID or the user ID that installed DB2).
- For Default user password, enter your DB2 password (your actual password or the password that installed DB2).
- Click *OK*.
- Press *Ctrl+S* to save your work.

Note: You can use this process for each data source you have to define. You will only need to define a data source one time for a particular server; the definition will be saved in the server's configuration.

Running the trade sample application

Once trade sample has been imported, the server configuration restored and the data source defined, you are ready to run the application.

- In the Server perspective, select the `Trade` project and *Run on Server* from the context menu.
- In the Server Selection dialog, select the Trade Server (under existing servers) and select the check box *Do not show this dialog next time (Set this server as the preferred server)*. Click *Finish*.

The project is added to the server configuration and published to the server. The server starts. You can see the messages in the console. If everything works correctly, a browser opens with the trade sample home page.

Explore the sample application. Notice from the home page you can go to a registration page, or enter your user ID and password to log in.

Note: The user ID and password that appear on the home page are valid. You can use them for exploring the sample, without having to create an account.

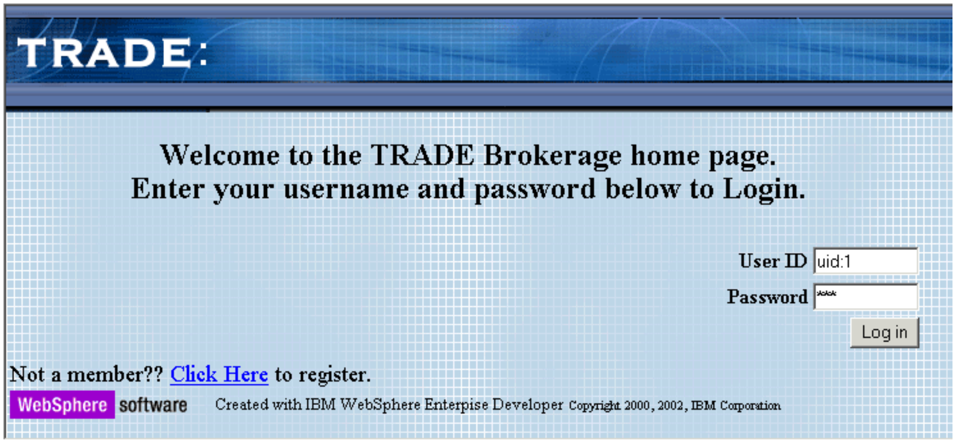
Note upon login that you receive your current portfolio. You can buy or sell shares, or obtain a quote (all fictitious transactions, of course, although they will update the `TRADEDB` database).

Once you are done exploring the sample application, you can log out. To stop the test server, go to the Servers view, and stop the server. Once the server has stopped, you can close the Server perspective.

Sample run

Here are a few screen captures of a sample run of the trade sample.

Login



TRADE:

Welcome to the TRADE Brokerage home page.
Enter your username and password below to Login.

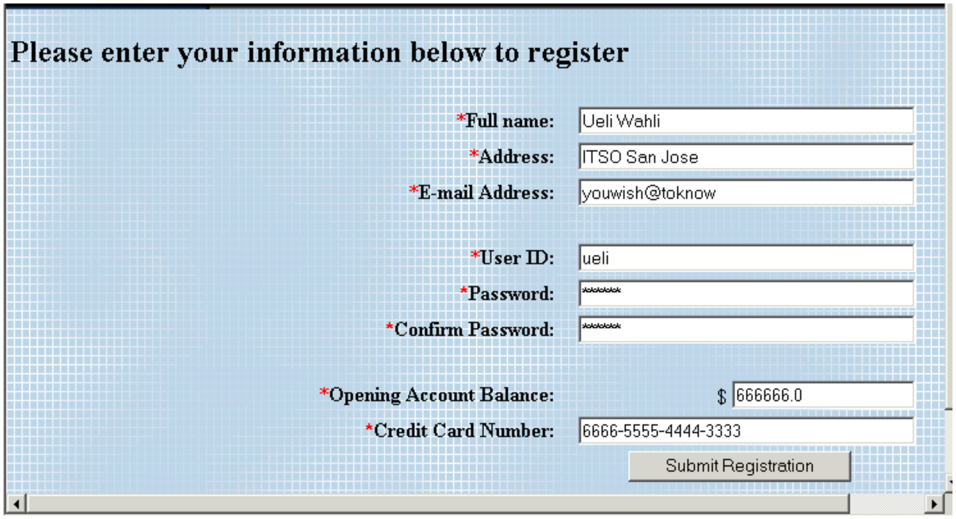
User ID	uid:1
Password	***

Not a member?? Click Here to register.

Created with IBM WebSphere Enterprise Developer Copyright 2000, 2002, IBM Corporation

Figure 3-4 Trade sample: home page

Register

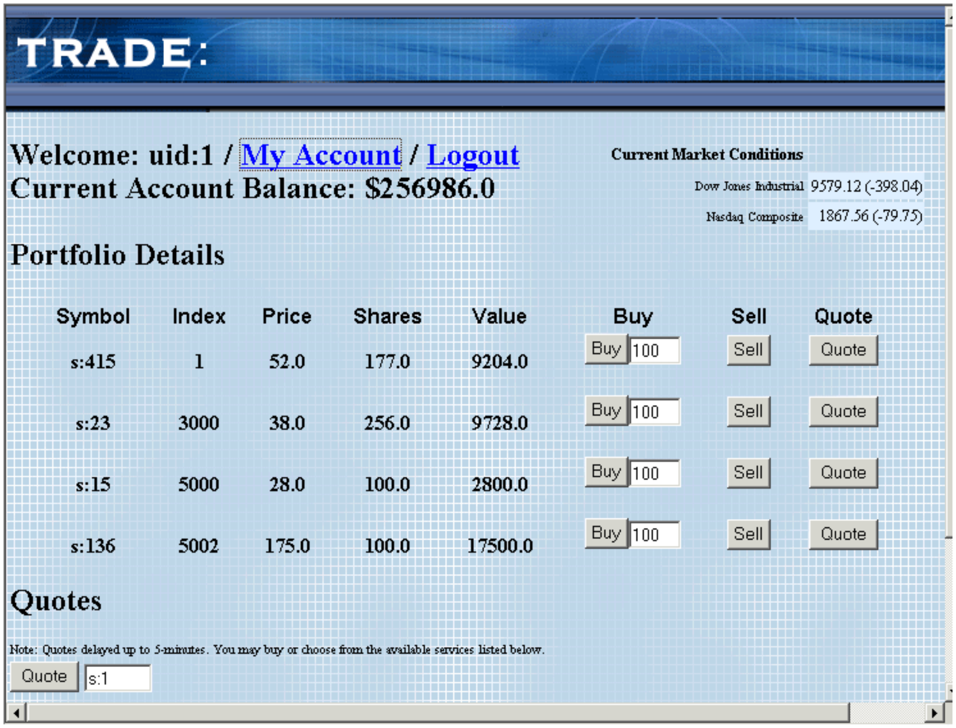


Please enter your information below to register

*Full name:	Ueli Wahli
*Address:	ITSO San Jose
*E-mail Address:	youwish@toknow
*User ID:	ueli
*Password:	******
*Confirm Password:	******
*Opening Account Balance:	$ 666666.0
*Credit Card Number:	6666-5555-4444-3333

Figure 3-5 Trade sample: register

Portfolio



Welcome: uid:1 / My Account / Logout
Current Account Balance: $256986.0

Current Market Conditions

Dow Jones Industrial	9579.12 (-398.04)
Nasdaq Composite	1867.56 (-79.75)

Portfolio Details

Symbol	Index	Price	Shares	Value	Buy	Sell	Quote
s:415	1	52.0	177.0	9204.0	Buy 100	Sell	Quote
s:23	3000	38.0	256.0	9728.0	Buy 100	Sell	Quote
s:15	5000	28.0	100.0	2800.0	Buy 100	Sell	Quote
s:136	5002	175.0	100.0	17500.0	Buy 100	Sell	Quote

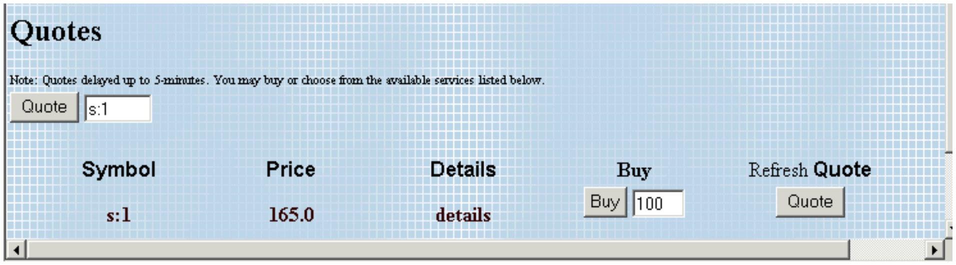
Quotes

Note: Quotes delayed up to 5-minutes. You may buy or choose from the available services listed below.

Quote s:1

Figure 3-6 Trade sample: portfolio

Quote



Quotes

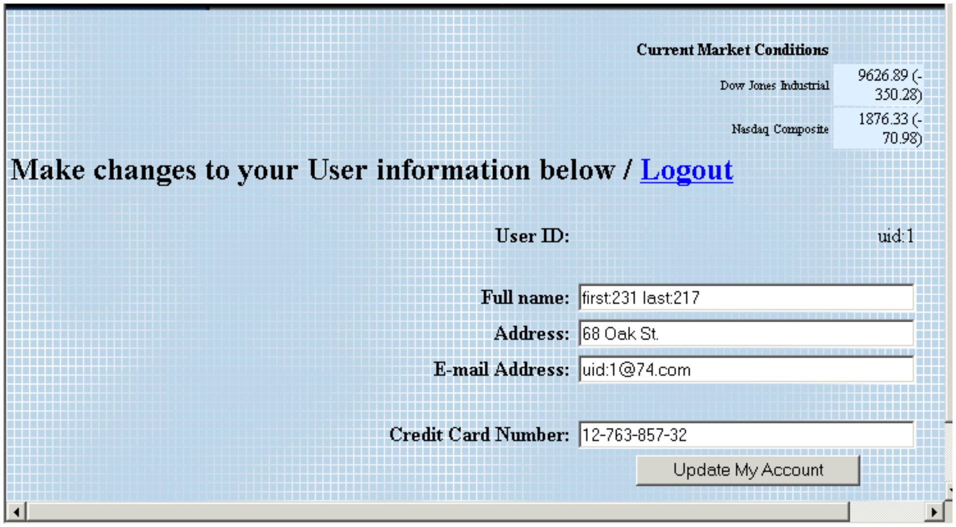
Note: Quotes delayed up to 5-minutes. You may buy or choose from the available services listed below.

Quote s:1

Symbol	Price	Details	Buy	Refresh Quote
s:1	165.0	details	Buy 100	Quote

Figure 3-7 Trade sample: quote

Account



Current Market Conditions

Dow Jones Industrial	9626.89 (-350.28)
Nasdaq Composite	1876.33 (-70.98)

Make changes to your User information below / Logout

User ID:	uid:1
Full name:	first:231 last:217
Address:	68 Oak St.
E-mail Address:	uid:1@74.com
Credit Card Number:	12-763-857-32

Figure 3-8 Trade sample: account

Summary

In this chapter, you learned about the trade sample application that ships with WebSphere Studio Enterprise Developer. You also learned about the roles needed for a development team. Finally, you loaded the trade sample application and tested it.

The next chapters will help you understand how to develop using WebSphere Studio Enterprise Developer, and will focus on developing the login capability of the trade sample application.

Part 2

Struts-based applications

In Part 2 we introduce the Struts concepts and then start building a subset of the sample application to illustrate the concepts.

We touch on basic Struts components and implement a simple action in a Struts application. We also introduce the Struts application diagram editor and build the same example using the diagram editor.

4

Components of a Struts-based application

Struts is an open source framework for building Web applications using the model-view-controller (MVC) architecture. This chapter introduces Struts by describing:

- MVC architecture and Web applications
- Struts introduction
- Components of a Struts-based application
- Configuration of a Struts-based application
- JSP details

Subsequent chapters show you how to use WebSphere Studio Enterprise Developer to create the components of a Struts application, complete the application, and test it.

Overview

Struts provides a way for developers to apply the model-view-controller (MVC) design pattern to Web applications. In order to understand how that is done, we must first understand what Web applications are, what MVC is, and what it means to apply MVC to a Web application.

In this chapter, we first look at MVC and Web applications. We then introduce Struts and how Struts ties MVC and Web applications together. Then we look at the components and configuration of a Struts-based application in detail.

Finally, we detail JSPs and taglibs, which are major components of Struts.

Model-view-controller

The model-view-controller (MVC) design pattern separates the parts of an application. MVC is not unique to Web applications; it was around well before Web applications.

MVC separates an application into:

Model The model contains the core of the application function. The model captures the state of the application. It does not include knowledge of the view or controller.

View The view is the look of the application. The view presents, gathers, and submits information, but it does not include knowledge of the model or controller.

Controller The controller manages the execution flow of the application, passing appropriate state information between the model and the view.

The parts are independent of each other, so that changing how one part is implemented does not require changes to the other parts. For example, the view of a Web application may change many times due to usability testing. However, the business logic (the model) acting on the input does not need to change (assuming the inputs to the business logic stay the same).

Web application

Web applications are defined in the servlet specification as “a collection of servlets, HTML pages, classes, and other resources that can be bundled and run...”. The specification defines the elements of a Web application:

- Servlets
- JavaServer Pages (JSP)
- Utility classes
- Static documents (HTML, images, sounds, etc.)
- Client-side applets, beans and classes
- Descriptive meta information that ties all of the above elements together

This definition of a Web application evolved as problems were identified and solved. Initially, servlets and static documents made up a “Web application”. JSPs were designed to help solve a problem with returning dynamic HTML pages to users and were added to the definition.

Note that there is no definition of model, view, or controller components.

In this section, we briefly discuss servlets and JSPs, problems they solved—and created—and how they can work together to implement an MVC architecture.

Servlets

Servlets are Java alternatives to Common Gateway Interface (CGI) programs. As in CGI programs, servlets can respond to user events from an HTML request, and then dynamically construct an HTML response that is sent back to the client. Servlets have the following advantages over traditional CGI programs:

- *Java-based*—Because servlets are written in Java, they inherit all the benefits of the Java technologies.
- *Persistence and performance*—A servlet is loaded once by a Web server and invoked for each client request. Servlets do not incur the overhead of instantiating a new servlet with each request. CGI processes typically must be loaded with each invocation.

Servlets can work better than CGI programs, especially for business logic and control flow.

However, for a servlet to return an HTML page to a browser, it must output HTML from within the Java code. A Java programmer ends up writing a bunch of `out.println` statements to return the HTML. This ends up mixing the roles of the content developer and the Java programmer, even limiting the usefulness of content-authoring tools.

JSPs were developed to address the problem of writing HTML statements in Java source code.

JavaServer Pages

JSPs are an HTML extension for doing server-side scripting in Web pages. JSPs are similar to HTML files, but provide the ability to display dynamic content within Web pages. Here are some of the advantages of using JSP technology over other methods of dynamic content creation:

- *Separation of dynamic and static content*—This allows for the separation of application logic and Web page design, reducing the complexity of Web site development and making sites easier to maintain.
- *Platform independence*—Because JSP technology is Java-based, it is platform independent. JSPs can run on nearly any Web site application server. JSPs can be developed on any platform and viewed by any browser because the output of a compiled JSP page is HTML.
- *Scripting and tags*—JSPs support both embedded Java and tags. Java is typically used to add page-level capability to the JSP. Tags provide an easy way to embed and modify JavaBean properties and to specify other directives and actions.

While JSPs look like HTML pages that access dynamic data, they are actually servlets. The application server compiles JSPs and executes them. Being a servlet is what allows JSPs to easily add dynamic content.

Because JSPs solved the problem of having HTML produced by Java code, many Web applications were written solely as JSPs (known as JSP Model 1).

However, while JSPs are good for producing HTML, they are not good for writing business logic and control because having Java code inside JSPs could easily make them hard to read or maintain.

Web applications using MVC

JSPs were designed to help simplify the process of servlets returning HTML pages. But because JSPs are themselves servlets, you now have all the advantages of a servlet and the advantages of the JSP. Many Web applications were written solely as JSPs. This helped by not having to write HTML in Java, but introduced a new problem—writing Java code in an otherwise HTML document. Whereas servlets had the problem of writing HTML from within Java code, Model 1 JSPs had the problem of writing Java code in HTML.

A solution to that problem is to use servlets and JSPs together to implement Web applications using the MVC architecture. This became known as JSP model-2, or just model-2. According to the Struts *User’s Guide*, “It is now commonplace to use the terms model-2 and MVC interchangeably.“

Using a servlet as the controller and JSPs as the view, we can use the MVC design pattern for Web applications.

Model The business logic invoked by the servlet. This is typically implemented by JavaBeans or Enterprise JavaBeans, accessing back-end databases or legacy systems.

View The JSPs used to interact with the application.

Controller The servlet controlling the flow between the JSPs and the business logic classes.

Figure 4-1 shows a basic Web application structure that complies with the MVC design pattern.

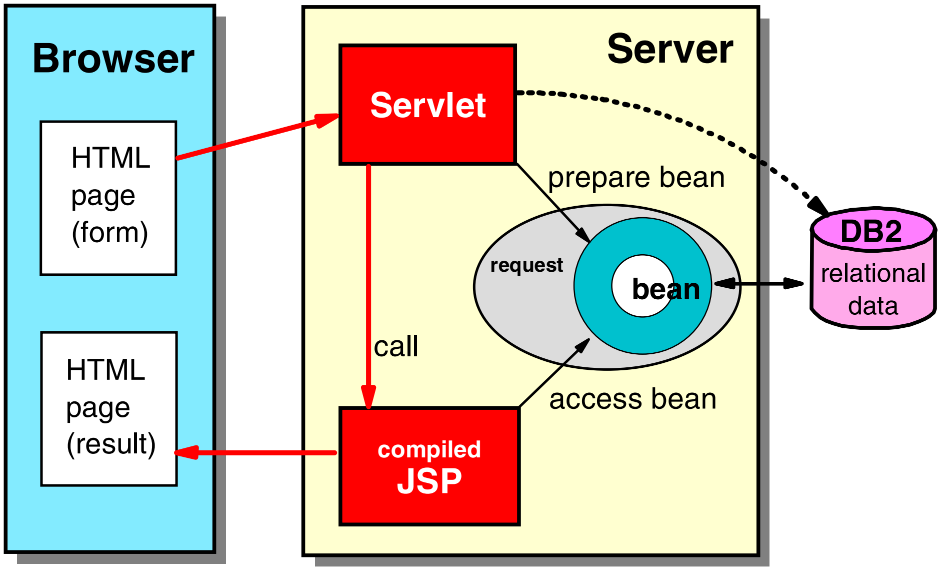


Figure 4-1 Basic Web application using the MVC design pattern

Let us review how the Web interaction works:

- An HTML page is displayed in a browser. The HTML page contains a form where the user can enter data and submit the form for processing.
- The Web server passes the request to an application server that schedules a servlet to process the form.
- In the model-view-controller (MVC) design pattern, the servlet is the controller. The servlet uses a JavaBean (the model) for the business logic. The JavaBean performs the requested tasks, for example, by accessing a relational database.

- The servlet then invokes a JSP (the view) to format the HTML result page. The JSP accesses the JavaBean to retrieve the result data of the model.

In many real Web applications, processing is more complex. Figure 4-2 shows a refined structure of the basic Web application.

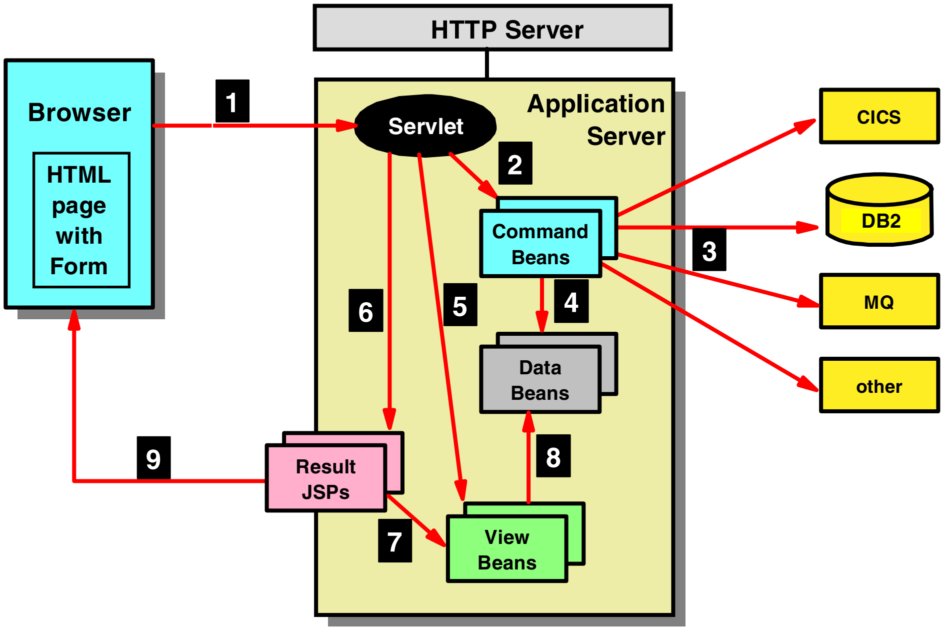


Figure 4-2 Basic Web application: refined

1. A servlet is invoked from an HTML form.
2. The servlet uses command beans to process the request.
3. Command beans perform the business logic by accessing databases and/or back-end transaction systems.
4. The result of commands are data beans (JavaBeans); for example, the result of a CICS transaction is a COMMAREA represented in a Java record.
5. The servlet allocates view beans that are used to process and format the data stored in the data beans into formats suitable for HTML output. (This is optional, but sometimes required data beans may be predefined.)
6. The servlet invokes a JSP to generate the HTML output. Depending on return codes from the command beans, one of multiple JSPs may be invoked.
7. The JSP uses the view beans to retrieve formatted results.
8. The view beans use the data beans to process and format the results.
9. The JSP generates the HTML result page.

Struts application overview

Struts is an open source framework for building MVC-based Web applications. Struts is part of the Jakarta project, sponsored by the Apache Software Foundation.

In this section we introduce Struts and its components and put them in the MVC context.

> **Note:** This section and its subsections contain documentation taken from the official Jakarta project Struts home page and from the official Struts user's guide at:
>
> `http://jakarta.apache.org/struts`
> `http://jakarta.apache.org/struts/userGuide/introduction.html`
>
> It also contains some quotes from Kyle Brown's articles on Struts in the VisualAge Developer Domain (VADD):
>
> `http://www7b.software.ibm.com/wsdd/`
>
> - Apache Struts and VisualAge for Java, Part 1: Building Web-based Applications using Apache Struts
> - Apache Struts and VisualAge for Java, Part 2: Using Struts in VisualAge for Java 3.5.2 and 3.5.3

Struts

The goal of the Struts project is to provide an open source framework useful in building MVC-based Web applications using servlet and JSP technologies. From an MVC point of view, Struts provides:

Model Struts provides no special support for the model. The business logic must be provided by the Web application developer usually as Java objects (JavaBeans or Enterprise JavaBeans).

View An `org.apache.struts.action.ActionForm` class to create form beans that are used to pass data between the controller and view. In addition, Struts custom tag libraries that assist developers in creating interactive form-based applications.

Controller An `org.apache.struts.action.Action` class that developers use to create the classes that control the flow of the application. Also, Struts provides an `org.apache.struts.action.ActionServlet` class to implement a controller servlet.

Struts also provides utility classes to support XML parsing, automatic population of JavaBeans properties based on the Java reflection APIs, and internationalization of prompts and messages.

A typical Struts Web application has a single servlet (extending `org.apache.struts.action.ActionServlet`), which uses an XML file for configuration information. There would be multiple Action classes (extending `org.apache.struts.action.Action`) and JSPs (using the Struts taglibs). Figure 4-3 shows the Struts components in the MVC architecture.

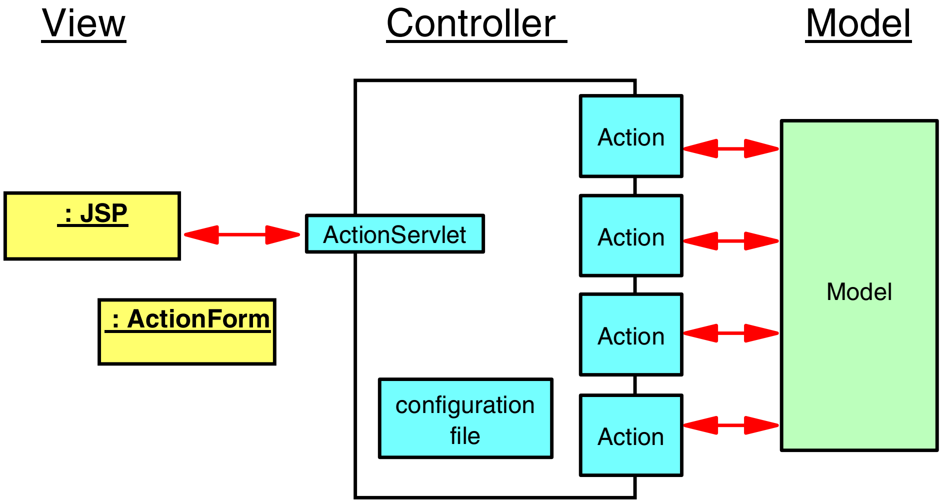


Figure 4-3 Struts components in the MVC architecture

When to use Struts

Most common Web applications can find some benefit in using Struts. As we have seen earlier, the MVC pattern allows us to design the model (business logic) of the application in a traditional fashion. Adding a Web controller and view transforms this model into a Web application. Struts helps building the *controller* and *view* parts, thus allowing you to focus on the business logic.

While J2EE APIs make it possible to develop Web-based applications that implement the MVC pattern, there are a number of common problems that must be solved in every servlet project (from Kyle Brown’s articles on Struts in the VisualAge Developer Domain):

- Mapping HTTP parameters to JavaBeans—One of the most common tasks facing servlet programmers is to map a set of HTTP parameters (from the command line or from the POST of an HTML form) to a JavaBean for manipulation. This can be done using the `<jsp:useBean>` and

`<jsp:setProperty>` tags, but this arrangement is cumbersome because it requires POSTing to a JSP, something that is not encouraged in a model-2 MVC architecture.

- Validation—There is no standard way in servlet/JSP programming to validate that an HTML form is filled in correctly. This leaves every servlet programmer to develop his own validation procedures, or not, as is too often the case.
- Error display—There is no standard way to display error messages in a JSP page or generate error messages in a servlet.
- Message internationalization—Even when developers strive to keep as much of the HTML as possible in JSPs, there are often hidden obstacles to internationalization spread throughout servlet and model code in the form of short error or informational messages. While it is possible to introduce internationalization with the use of Java resource managers, this is rarely done due to the complexity of adding these references.
- Hard coded JSP URIs—One of the more insidious problems in a servlet architecture is that the URIs of the JSP pages are usually coded directly into the code of the calling servlet in the form of a static string reference used in the `ServletContext.getRequestDispatcher` method. This means that it is impossible to reorganize the JSPs in a Web site, or even change their names, without updating Java code in the servlets.

The problem is that programmers are too often faced with "reinventing the wheel" each time they begin building a new Web-based application. Having a framework to do this work for them would make them more productive and let them focus more on the essence of the business problems they are trying to solve, rather than on the accidents of programming caused by the limitations of the technology (from *No Silver Bullet: Essence and Accident in Software Engineering*, by Fred Brooks. IEEE Computer, April 1987).

Simply put, Struts is an open-source framework for solving the kind of problems described above. Information on Struts, a set of installable JAR files, and the full Struts source code is available at the Struts framework Web site. Struts has been designed from the ground up to be easy to use, modular (so that you can choose to use one part of Struts without having to use all the others), and efficient. It has also been designed so that tool builders can easily write their tools to generate code that sits on top of the Struts framework.

Struts components

True to the model-view-controller design pattern, Struts applications have three major components: a servlet (the *controller*), JavaServer Pages (the *view*), and the application's business logic (the *model*).

Struts model

Struts does not define its own model component. In a Web application (and a Struts application), most of the model (the business logic) can be represented using JavaBeans or EJBs. Access to the business logic is through Struts action objects (classes that subclass `org.apache.struts.action.Action`).

The action object can handle the request and respond to the client (usually a Web browser), or indicate that control should be forwarded to another action. For example, if a login succeeds, a `loginAction` object may want to forward control to a `mainMenu` action.

Action objects are linked to the application controller, and so have access to that servlet's methods. When forwarding control, an object can indirectly forward one or more shared objects, including JavaBeans, by placing them in one of the standard collections shared by Java servlets.

An action object can for instance create a shopping cart bean, add an item to the cart, place the bean in the session collection, and then forward control to another action, which may use a JSP to display the contents of the user's cart. Because each client has its own session, each also has its own shopping cart.

Struts view

The view in a Struts application is made up of various components. JSPs are the main component. JSPs, of course, are not Struts components. However, Struts provides two components that work with JSPs:

- Form beans
- Custom tags

Form beans

JavaBeans can also be used to manage input forms. A key problem in designing Web applications is retaining and validating what a user has entered between requests. With Struts, you can easily store the data for an input form in a form bean (a class that extends `org.apache.struts.action.ActionForm`). The bean is saved in one of the standard, shared context collections, so that it can be used by other objects. The action receives the form bean as input to perform its task.

The form bean can be used:

- To collect data from the user
- To validate what the user entered
- By the JSP to repopulate the form fields

In the case of validation errors, Struts has a shared mechanism for raising and displaying error messages. It automatically invokes the `ActionForm.validate` method whenever the JSP page containing the form corresponding to this `ActionForm` submits the form. Any type of validation can be performed in this method. The only requirement is that it returns a set of `ActionError` objects in the return value. Each `ActionError` corresponds to a single validation failure, which maps to a specific error message. These error messages are held in a properties file that the Struts application refers to.

Custom tags

There are four JSP tag libraries that Struts includes:

1. The HTML tag library, which includes tags for describing dynamic pages, especially forms.
2. The beans tag library, which provides additional tags for providing improved access to Java beans and additional support for internationalization.
3. The logic tag library, which provides tags that support conditional execution and looping.
4. The template tag library for producing and using common JSP templates in multiple pages.

Using these custom tags, the Struts framework can automatically populate fields from and to a form bean, raising two advantages:

- The only thing most JSPs need to know about the rest of the framework is the proper field names and where to submit the form. The associated form bean automatically receives the corresponding value.
- If a bean is present in the appropriate scope, for instance after an input validation routine, the form fields will be automatically initialized with the matching property values.

Therefore, an input field declared in a JSP using Java code as:

```
<input type="text" name="fName" value="<%= bean.getFirstName() %>">
```

can be replaced by a more elegant and efficient Struts tag:

```
<html:text property="fName"/>
```

with no need to explicitly refer to the JavaBean from which the initial value is retrieved. That is handled automatically by the JSP tag, using facilities provided by the framework.

Struts controller

The controller component in a Struts application is implemented in two parts: the action servlet and action classes.

Action servlet

The Struts framework provides the `org.apache.struts.action.ActionServlet` class for servlets. The action servlet bundles and routes HTTP requests from the client (typically a user running a Web browser) to action classes and corresponding extended objects, deciding what business logic function is to be performed, then delegates responsibility for producing the next phase of the user interface to an appropriate view component like a JSP.

When initialized, the action servlet parses a configuration file. The configuration file defines, among other things, the action mappings for the application. The controller uses these mappings to turn HTTP requests into application actions.

At a minimum, a mapping must specify:

- A request path
- The object type to act upon the request

Each mapping defines a path that is matched against the request URI of the incoming request, and the fully qualified class name of an action class.

Action classes

An action class is one that extends `org.apache.struts.action.Action`. The action classes interface with the application's business logic. Based on the results of the processing, the action class determines how control should proceed. The action class specifies which JSP or servlet control should be forwarded to.

Note: The action class can contain the actual business logic, in which case it would be considered the model and not the controller. However, this practice is discouraged, as it would then mix the application's business logic with the Struts framework code; this would limit the ability to reuse the business logic. The recommended practice is to use the action class as an interface to the business logic, and allow it to share in the controller role, guiding the flow of the application.

Struts application flow

A Struts form bean is defined in the configuration file and linked to an action mapping using a common property name. When a request calls for an action that uses a form bean, the controller servlet retrieves the form bean (or creates it if it does not exist), and passes it to the action (Figure 4-4).

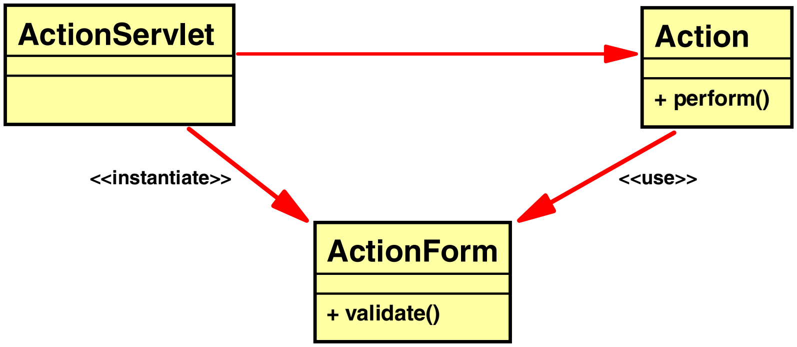


Figure 4-4 ActionForm handling

The action can then check the contents of the form bean before its input form is displayed, and also queue messages to be handled by the form. When ready, the action can return control with a forwarding to its output form, usually a JSP. The controller can then respond to the HTTP request and direct the client to the JSP. Figure 4-5 summarizes these operations.

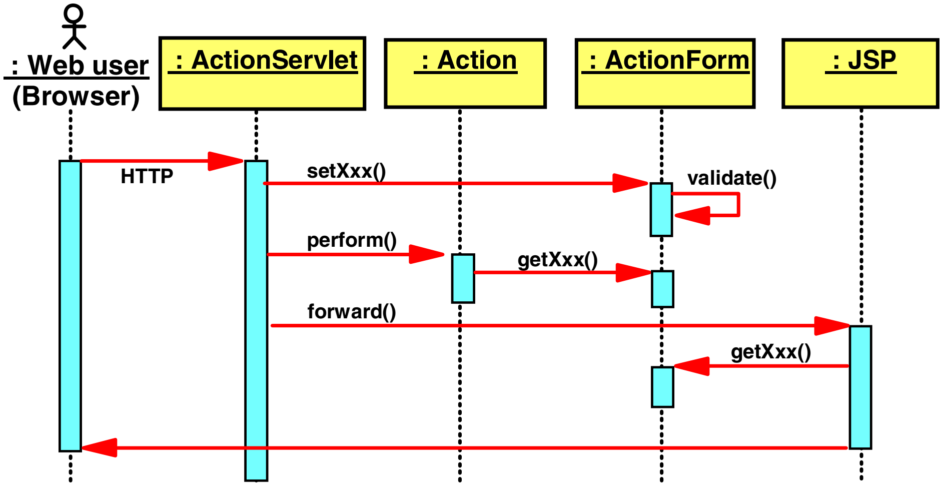


Figure 4-5 Struts request sequences

Configurations

Struts includes a servlet that implements the primary function of mapping a request URI to an action class. Therefore, your primary responsibilities related to the controller are:

- Write an action class for each logical request that may be received (extend `org.apache.action.Action`).
- Configure an action mapping (in XML) for each logical request that may be submitted. The XML configuration file is usually named *`struts-config.xml`*.
- Update the Web application deployment descriptor file (in XML) for your application to include the necessary Struts components.
- Add the appropriate Struts components to your application.

Action classes

The action class defines two methods that could be executed depending on your servlet environment:

```
public ActionForward perform(ActionMapping mapping,
                             ActionForm form,
                             ServletRequest request,
                             ServletResponse response)
                      throws IOException, ServletException;

public ActionForward perform(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
                      throws IOException, ServletException;
```

Most projects would only use the `HttpServletRequest` version.

The goal of an action class is to process a request, via its *perform* method, and return an `ActionForward` object that identifies where control should be forwarded (in most cases a JSP) to provide the appropriate response. In the MVC model-2 design pattern, a typical action class often implements logic performing these functions in its `perform` method:

- Validate the current state of the user's session (for example, checking that the user has successfully logged on). If the action class finds that no logon exists, the request can be forwarded to the JSP page that displays the username and password prompts for logging on. This could occur because a user tried to enter an application "in the middle" (say, from a bookmark), or because the session has timed out, and the servlet container created a new one.

- If validation is not complete, validate the form bean properties as needed. If a problem is found, store the appropriate error message keys as a request attribute, and forward control back to the input form so that the errors can be corrected.
- Perform the processing required to deal with this request (such as saving a row into a database). This can be done by logic code embedded within the action class itself, but should generally be performed by calling an appropriate method of a business logic bean.
- Update the server-side objects that will be used to create the next page of the user interface (typically request scope or session scope beans, depending on how long you need to keep these items available).
- Return an appropriate `ActionForward` object that identifies the JSP page to be used to generate this response, based on the newly updated beans. Typically, you acquire a reference to such an object by calling `findForward` on either the `ActionMapping` object you received (if you are using a logical name local to this mapping), or on the controller servlet itself (if you are using a logical name global to the application).

Action mapping implementation

In order to operate successfully, the Struts controller servlet has to know several things about how each request URI should be mapped to an appropriate *action* class. The required knowledge has been encapsulated in a Java class named `ActionMapping`.

Action mapping configuration file

How does the controller servlet learn about the mappings you want? It would be possible (but tedious) to write a small Java class that simply instantiated new `ActionMapping` instances, and called all of the appropriate setter methods. To make this process easier, Struts has a capability of reading an XML-based description of the desired mappings, creating the appropriate objects along the way.

The developer's responsibility is to create a `struts-config.xml` file and place it in the `WEB-INF` directory of your application. The format of the document is defined in `struts-config_1_0.dtd`. The outermost XML element must be `<struts-config>`.

> **Note:** By convention the configuration file is named `struts-config.xml`. This name is configurable as an initialization parameter to the action servlet in the `web.xml` deployment descriptor file.

Inside of the `<struts-config>` element, there are four elements that are used to describe your actions:

- Form beans
- Action mappings
- Global and local forwards
- Data sources

Form beans

This section contains your form bean definitions. You use a `<form-bean>` element for each form bean, which has the following important attributes:

- **name**—A unique identifier for this bean, which is used to reference it in corresponding action mappings. Usually, this is also the name of the request or session attribute under which this form bean is stored.
- **type**—The fully qualified Java classname of your form bean.

Action mappings

This section contains your action definitions. You use an `<action>` element for each of your actions you would like to define. The following are the attributes of action element:

attribute Name of the request-scope or session-scope attribute under which our form bean is accessed, if it is other than the bean's specified *name*. Optional if *name* is specified, else not allowed.

className Fully qualified Java class name of the action mapping implementation class to use. Defaults to the value configured as the mapping initialization parameter to the Struts controller servlet.

forward Context-relative path of the servlet or JSP resource that will process this request, instead of instantiating and calling the action class specified by *type*. Exactly one of *forward*, *include*, or *type* must be specified.

include Same as forward, but a request dispatcher *include* is issued, that is, control comes back after calling the target servlet or JSP.

input Context-relative path of the input form to which control should be returned if a validation error is encountered. Required if *name* is specified and the input bean returns validation errors. Optional if *name* is specified and the input bean does not return validation errors. Not allowed if *name* is not specified.

name Name of the form bean, if any, that is associated with this action.

path The context-relative path of the submitted request, starting with a "/" character, and without the filename extension if extension mapping is used.

parameter General-purpose configuration parameter that can be used to pass extra information to the action selected by this mapping.

prefix Prefix used to match request parameter names to form bean property names, if any. Optional if *name* is specified, else not allowed.

scope Identifier of the scope (*request* or *session*) within which our form bean is accessed, if any. Optional if *name* is specified, else not allowed.

suffix Suffix used to match request parameter names to form bean property names, if any. Optional if *name* is specified, else not allowed.

type Fully qualified Java class name of the action class (implements `org.apache.struts.action.Action`) to be used to process requests for this mapping if the *forward* or *include* attribute is not included. Exactly one of *forward*, *include*, or *type* must be specified.

unknown Set to *true* if this action should be configured as the default for this application, to handle all requests not handled by another action. Only one action can be defined as a default within a single application.

validate Set to *true* if the `validate` method of the form bean should be called prior to calling this action, or set to *false* if you do not want validation performed.

Global forwards

The `<global-forwards>` section is used to create logical name mappings for commonly used JSP pages. Each of these forwards is available through a call to your action mapping instance, for example:

```
mapping.findForward("logicalName")
```

Local forwards

Optional but very useful are the local forward elements. They are similar to global forwards, except that they are defined for a specific action. This allows an action to refer to a forward by a logical name rather than a specific file name or servlet URI.

Data sources

One more section of good use is the <data-sources> section, which specifies data sources that your application can use. This is how you would specify a basic data source for your application inside of struts-config.xml:

```
<data-sources>
  <data-source
    autoCommit="false"
   description="Example Data Source Description"
   driverClass="org.postgresql.Driver"
      maxCount="4"
      minCount="2"
      password="mypassword"
           url="jdbc:postgresql://localhost/mydatabase"
          user="myusername"/>
</data-sources>
```

Sample configuration file

The struts-config.xml file from the example application includes the following mapping entry for the logon function, which we will use to illustrate the requirements. Note that the entries for all the other actions are left out:

```
<struts-config>
  <form-beans>
    <form-bean
      name="logonForm"
      type="org.apache.struts.example.LogonForm" />
  </form-beans>
  <global-forwards
      type="org.apache.struts.action.ActionForward" />
    <forward name="logon" path="/logon.jsp"
         redirect="false" />
  </global-forwards>
  <action-mappings>
    <action
        path="/logon"
        type="org.apache.struts.example.LogonAction"
        name="logonForm"
        scope="request"
        input="/logon.jsp"
     unknown="false"
    validate="true" />
  </action-mappings>
</struts-config>
```

First the form bean is defined. A basic bean of class org.apache.struts.example.LogonForm is mapped to the logical name logonForm. This name is used as a session or request attribute name for the form bean.

As you can see, this mapping matches the path /logon. When a request that matches this path is received, an instance of the LogonAction class is created (the first time only) and used. The controller servlet will look for a session or request scoped bean under key logonForm, creating and saving a bean of the specified class if needed.

Also notice the local forwards. In the example application, many actions include a local success and/or failure forward as part of an action mapping.

```
<!-- Edit mail subscription -->
<action    path="/editSubscription"
  type="org.apache.struts.example.EditSubscriptionAction"
  name="subscriptionForm"
  scope="request"
  validate="false">
  <forward name="failure" path="/mainMenu.jsp"/>
  <forward name="success" path="/subscription.jsp"/>
  </action>
```

Using just these two extra properties, the action classes in the example application are almost totally independent of the actual names of the JSP pages that are used by the page designers. The pages can be renamed (for example) during a redesign, with negligible impact on the action classes themselves. If the names of the next JSP pages were hard coded into the action classes, all of these classes would also need to be modified. Of course, you can define whatever local forward properties makes sense for your own application.

Web application deployment descriptor

The final step in setting up the application is to configure the application deployment descriptor (stored in file WEB-INF/web.xml) to include all the Struts components that are required. Using the deployment descriptor for the example application as a guide, we see that the following entries have to be created or modified.

Configure the action servlet instance

Add an entry defining the action servlet itself, along with the appropriate initialization parameters. Such an entry might look like this:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
```

```
      org.apache.struts.action.ActionServlet
    </servlet-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>
        org.apache.struts.example.ApplicationResources
      </param-value>
    </init-param>
    <init-param>
      <param-name>config</param-name>
      <param-value>
        /WEB-INF/struts-config.xml
      </param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>mapping</param-name>
      <param-value>
        org.apache.struts.example.ApplicationMapping
      </param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
```

Note: The definition of the `ActionServlet` can be done using the `web.xml` editor by adding the servlet and defining the initialization parameters.

The initialization parameters supported by the controller servlet are described below. Square brackets describe the default values that are assumed if you do not provide a value for that initialization parameter.

application Java class name of the application resources bundle base class. [`NONE`]

bufferSize The size of the input buffer used when processing file uploads. [4096]

config Context-relative path to the XML resource containing our configuration information. [`/WEB-INF/struts-config.xml`]

content Default content type and character encoding to be set on each response; may be overridden by a forwarded-to servlet or JSP page. [`text/html`]

debug The debugging detail level for this servlet, which controls how much information is logged. [0]

detail The debugging detail level for the digester we utilize in *initMapping*, which logs to `System.out` instead of the servlet log. [0]

factory The Java class name of the `MessageResourcesFactory` used to create the application `MessageResources` object. [`org.apache.struts.util.PropertyMessageResourcesFactory`]

formBean The Java class name of the `ActionFormBean` implementation to use [`org.apache.struts.action.ActionFormBean`].

forward The Java class name of the `ActionForward` implementation to use [`org.apache.struts.action.ActionForward`].

locale If set to true, and there is a user session, identify and store an appropriate java.util.Locale object (under the standard key identified by `Action.LOCALE_KEY`) in the user's session if there is not a Locale object there already. [true]

mapping The Java class name of the `ActionMapping` implementation to use [`org.apache.struts.action.ActionMapping`].

maxFileSize The maximum size (in bytes) of a file to be accepted as a file upload. Can be expressed as a number followed by a "K" "M", or "G", which are interpreted to mean kilobytes, megabytes, or gigabytes, respectively. [250M]

multipartClass The fully qualified name of the `MultipartRequestHandler` implementation class to be used for processing file uploads. [`org.apache.struts.upload.DiskMultipartRequestHandler`]

nocache If set to true, add HTTP headers to every response intended to defeat browser caching of any response we generate or forward to. [false]

null If set to true, set our application resources to return null if an unknown message key is used. Otherwise, an error message including the offending message key is returned. [true]

tempDir The temporary working directory to use when processing file uploads. [The working directory provided to this Web application as a servlet context attribute]

validate Are we using the new configuration file format? [true]

validating Should we use a validating XML parse to process the configuration file (strongly recommended)? [true]

Configure the action servlet mapping

Note: The material in this section is not specific to Struts. The configuration of servlet mappings is defined in the Java servlet specification. This section describes the most common means of configuring a Struts application.

There are two approaches to defining the URLs that are processed by the controller servlet—prefix matching and extension matching. An appropriate mapping entry for each approach will be described below.

Prefix matching

Prefix matching means that you want all URLs that start (after the context path part) with a particular value to be passed to this servlet. Such an entry might look like this:

```
<servlet-mapping>
     <servlet-name>action</servlet-name>
     <url-pattern>/execute/*</url-pattern>
</servlet-mapping>
```

which means that a request URI to match the */logon* path described earlier might look like this:

```
http://www.mycompany.com/myapplication/execute/logon
```

where `/myapplication` is the context path under which your application is deployed.

Extension mapping

Extension mapping, on the other hand, matches request URIs to the action servlet based on the fact that the URI ends with a period followed by a defined set of characters. For example, the JSP processing servlet is mapped to the **.jsp* pattern so that it is called to process every JSP page that is requested. To use the **.do* extension (which implies “do something”), the mapping entry would look like this:

```
<servlet-mapping>
     <servlet-name>action</servlet-name>
     <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

A request URI to match the /logon path might look like this:

```
http://www.mycompany.com/myapplication/logon.do
```

Configure Struts tag library

Next, you must add an entry defining the Struts tag library. There are currently four taglibs that Struts is packaged with.

- **struts-bean**—The `struts-bean.tlb` taglib contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.
- **struts-html**—The `struts-html.tlb` taglib contains tags used to create struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces. There is also a **struts-form** taglib, which seems to be a subset of `struts-html`.
- **struts-logic**—The `struts-logic.tlb` taglib contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.
- **struts-template**—The `struts-template.tlb` taglib contains tags that define a template mechanism.

Below is the code for defining all taglibs for use within your application. In reality you would only specify the taglibs that your application uses:

```
<taglib>
  <taglib-uri>
    /WEB-INF/struts-bean.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/struts-bean.tld
  </taglib-location>
</taglib>
<taglib>
  <taglib-uri>
    /WEB-INF/struts-html.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/struts-html.tld
  </taglib-location>
</taglib>
<taglib>
  <taglib-uri>
    /WEB-INF/struts-logic.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/struts-logic.tld
  </taglib-location>
</taglib>
```

```
<taglib>
  <taglib-uri>
    /WEB-INF/struts-template.tld
  </taglib-uri>
  <taglib-location>
    /WEB-INF/struts-template.tld
  </taglib-location>
</taglib>
```

This tells the JSP system where to find the tag library descriptor, for example, in the Web application `WEB-INF` (or `WEB-INF\lib`) directory instead of on the Internet somewhere.

Add Struts components to your application

To use Struts, you must copy the `.tld` files that you require into your `WEB-INF` directory, and copy `struts.jar` into your `WEB-INF/lib` directory. This is done for you when you use Enterprise Developer to define a Web project with Struts support.

JSP overview

Struts makes extensive use of JSPs. Therefore it is important to understand how JSPs work, and in particular customer taglibs. This section focuses on providing background information on JSPs.

JSP technology provides the ability to build applications containing dynamic content such as HTML and XML. To facilitate embedding of dynamic content, JSPs use a number of *tags* that enable the page designer to insert the properties of a JavaBean object and script elements into a JSP file.

JSP technology was developed by Sun Microsystems. JSP specification is available at:

`http://java.sun.com/products/jsp/`

How JSPs work

The following process outlines the tasks performed on a JSP file on the *first invocation* of the file or when the underlying JSP file is changed by the developer.

- The Web browser makes a request to the JSP page.
- The application server Web parses the contents of the JSP file, creates temporary servlet source code based on the contents of the JSP, and compiles the source into a servlet class file. The generated servlet is

responsible for rendering the static elements of the JSP specified at design time in addition to creating the dynamic elements of the page.

- The servlet is instantiated. The *init* and `service` methods of the servlet are called, and the servlet logic is executed.
- The combination of static HTML and graphics combined with the dynamic elements specified in the original JSP page definition are sent to the Web browser through the output stream of the servlet's response object.

Subsequent invocations of the JSP file simply invoke the *service* method of the servlet created by the above process to serve the content to the Web browser. The servlet produced as a result of the above process remains in service until the application server is stopped or the Web application is restarted.

Note: JSP can be precompiled by development tools or by the application server when an EAR file is installed.

Components of JSPs

JSPs are composed of *template data* and *elements*. The template data such as HTML text is passed through the JSP container and is sent to the requesting client. Elements are known to the JSP container and classified as follows:

- Directive elements
  - Page directive
  - Include directive
  - Taglib directive
- Scripting elements
  - Declarations
  - Scriptlets
  - Expressions
- Comments
- Action elements
  - Standard actions
  - Custom actions

The following sections describe each of these categories in more detail.

Directive elements

A JSP directive is a global definition sent to the JSP container. A directive always appears at the top of the JSP file, before any other JSP tags. This is due to the way the JSP parsing engine produces servlet code from the JSP file.

The syntax of a directive is:

```
<%@ directive directive_attr_name = value %>
```

Page directive

The *page* directive defines page-dependent attributes to the JSP container.

```
<%@ page language="java" buffer="none" isThreadSafe="yes"
        errorPage="/error.jsp" %>
```

The attributes of the page directive are listed in Table 4-1.

Table 4-1 Attributes of the page directive

Attribute Name	Description
language	Identifies the scripting language used in scriptlets in the JSP file or any of its included files. JSP supports only the value of “java”. `<%@ page language = "java" %>`
extends	The fully qualified name of the superclass to which this JSP page will be transformed. Using this attribute can affect the JSP container’s ability to select specialized superclasses based on the JSP file content, and should be used with care.
import	When the language attribute of "java" is defined, the import attribute specifies the additional files containing the types used within Java programing language environment. `<%@ page import = "java.util.*" %>`
session "true" | "false"	If *true*, specifies that the page participates in an HTTP session and enables the JSP file access to the implicit session object. The default value is *true*.
buffer "none" | "sizekb"	Indicates the buffer size for the JspWriter. If *none*, the output from the JSP is written directly to the ServletResponse PrintWriter object. Any other value results in the JspWriter buffering the output up to the specified size. The buffer is flushed in accordance with the value of the autoFlush attribute. The default buffer size is no less than 8 kb.
autoFlush "true" | "false"	If *true*, the buffer is flushed automatically. If false, an exception is raised when the buffer becomes full. The default value is *true*.

Attribute Name	Description
isThreadSafe "true" | "false"	If *true*, the JSP processor may send multiple outstanding client requests to the page concurrently. If *false*, the JSP processor sends outstanding client requests to the page consecutively, in the same order in which they were received. The default is *true*.
info	Allows the definition of a string value that can be retrieved using `Servlet.getServletInfo()`.
errorPage	Specifies the URL to be directed to for error handling if an exception is thrown and not caught within the page implementation.
isErrorPage "true" | "false"	Identifies that the JSP page refers to a URL identified in another JSP's errorPage attribute. Default is *false*.
contentType	Specifies the character encoding and MIME type of the JSP response. Default value for `contentType` is *text/html*. Default value for `charSet` is *ISO-8859-1*. The syntax format is: `contentType="text/html; charSet=ISO-8859-1"`

Include directive

The *include* directive allows substitution of text or code to occur at translation time. You can use the include directive to provide a standard header on each JSP page, for example:

```
<%@ include file="copyright.html" %>
```

The include directive has the attributes shown in Table 4-2.

Table 4-2 Attributes for the include directive

Attribute Name	Description
file	Directs the JSP container to substitute the text or code specified by file or URL reference. The URL reference can be another JSP file.

Taglib directive

The *taglib* directive declares the usage of a custom tag library. Users have to specify a tablib directive on each JSP page where they want to use tags from the taglib. The syntax is:

```
<%@ taglib uri="tagLibraryURI” prefix=”tagPrefix” %>
```

Table 4-3 Attributes for the taglib directive

Attribute Name	Description
uri	Specifies the location of the tag library descripter associated with the prefix. An absolute URI or a relative URI can be accepted.
tagPrefix	Defines the `prefix` in `prefix:tagname` to distinguish a custom action.

Scripting elements

Scripting elements are Java code fragments.

Declarations

A declaration block contains Java variables and methods that are called from an *expression* block within the JSP file. Code within a declaration block is usually written in Java. Code within a declaration block is often used to perform additional processing on the dynamic data generated by a JavaBean property.

The syntax of a declaration is:

```
<%! declaration(s) %>
```

For example:

```
<%!
   private int getDateCount = 0;
   private String getDate(GregorianCalendar gc1)
      { ...method body here...}
%>
```

Scriptlets

JSP supports embedding of Java code fragments within a JSP by using a *scriptlet* block. Scriptlets are used to embed small code blocks within the JSP page, rather than to declare entire methods as performed in a declarations block. The syntax for a scriptlet is:

```
<% scriptlet %>
```

The following example uses a scriptlet to output an HTML message based on the time of day. You can see that the HTML elements appear *outside* the script declarations.

```
<% if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM)
   {%>
      How are you this morning ?
<% } else
```

```
    { %>
       How are you this afternoon ?
<% } %>
```

Expressions

Expressions are scriptlet fragments whose results can be converted to String objects and subsequently fed to the output stream for display in a browser. The syntax for an expression is:

```
<%= expression %>
```

Typically, expressions are used to execute and display the String representation of variables and methods declared within the declarations section of the JSP, or from JavaBeans that are accessed by the JSP. If the conversion of the expression result is unsuccessful, a `ClassCastException` is thrown at the time of the request.

The following example calls the *incrementCounter* method declared in the declarations block and prints the result:

```
<%= incrementCounter() %>
```

All primitive types such as short, int, and long can be automatically converted to Strings. Your own classes must provide a *toString* method for String conversion.

Comments

You can use two types of comments within a JSP. The first comment style, known as an *output comment,* enables the comment to appear in the output stream on the browser. This comment is an HTML-formatted comment whose syntax is:

```
<!-- comments ... -->
```

The second comment style is used to fully exclude the commented block from the output and is never delivered to the browser. The syntax is:

```
<%-- comment text --%>
```

You can also create comments containing dynamic content by embedding a scriptlet tag inside a comment tag. For example:

```
<!-- comment text <%= expression %> more comment text ->
```

Standard actions

JSP specifications include the standard action tags shown in Table 4-4.

Table 4-4 JSP standard actions

Tag	Description
jsp:forward	This tag is used for the run-time dispatching to an HTML file, a file or a Servlet.
jsp:getProperty	Once the bean has been declared with `jsp:useBean`, you can access its exposed properties through this tag.
jsp:include	This tag allows the inclusion of data from another file.
jsp:plugin	This tag downloads a Java plug-in to the Web browser to execute an applet or a bean.
jsp:setProperty	The properties of beans can be set by using this tag.
jsp:useBean	The *`jsp:useBean`* tag is used to declare a *JavaBean* object that you want to use within the JSP.

For a complete description of all tags supported by the JSP specifications, please refer to the Sun JavaServer Pages Specifications available on the Sun Web site.

`http://java.sun.com/products/jsp/`

Custom actions

Custom actions can be defined using the tag library extension mechanisms. The Struts tag libraries are implemented using the tag library extension mechanism.

Struts tag libraries

JSP tag library facility provides a mechanism to define a special sub-language used by JSP page authors. Struts provides the following custom libraries for the Web page developers to build the view component of the *Struts* application. In this section, we explain these tag libraries.

- Struts bean tags
- Struts HTML tags
- Struts logic tags
- Struts template tags

Note: These subsections contain documentation taken from the official Jakarta project Struts home page at:

`http://jakarta.apache.org/struts`

Struts bean tags

This tag library contains tags useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.

The following is the brief description of the tag names of *Struts bean* tags.

cookie Define a scripting variable based on the value(s) of the specified request cookie.

define Define a scripting variable based on the value(s) of the specified bean property.

header Define a scripting variable based on the value(s) of the specified request header.

include Load the response from a dynamic application request and make it available as a bean.

message Render an internationalized message string to the response.

page Expose a specified item from the page context as a bean.

parameter Define a scripting variable based on the value(s) of the specified request parameter.

resource Load a Web application resource and make it available as a bean.

size Define a bean containing the number of elements in a *Collection* or *Map*.

struts Expose a named Struts internal configuration object as a bean.

write Render the value of the specified bean property to the current `JspWriter`.

Struts bean tags in trade sample

Struts bean tags are utilized in the trade sample in Enterprise Developer. The whole source of `index.jsp` is listed in Example 4-1 on page 101. To use this tag library, users have to define the JSP directive in JSP page:

```
<%@ taglib uri="WEB-INF/lib/struts-bean.tld" prefix="bean" %>
```

The following message tags are used for internationalization.

```
<bean:message key="market.text.title" />
<bean:message key="market.text.dowJones" />
<bean:message key="market.text.nasdaq" />
```

The message contents referred by the key attribute are stored in the application resource file whose file name is specified in the deployment descripter (`web.xml`). The trade sample uses `ApplicationResources.properties` for the resource file and the corresponding data is defined as follows:

```
market.text.title=Current Market Conditions
market.text.dowJones=Dow Jones Industrial
market.text.nasdaq=Nasdaq Composite
```

When the Struts application detects the `locale` of the client request, the message can be changed to the corresponding language. To change the message, the application resource file named `ApplicationResources_XX.properties` whose language code defined in ISO 639 is “`XX`” should be placed in the same directory as the resource file.

Struts HTML tags

This taglib contains tags used to create *Struts* input forms, as well as other tags generally useful in the creation of HTML-based user interfaces.

base	Render an HTML *<base>* element
button	Render a button input field
cancel	Render a *Cancel* button
checkbox	Render a checkbox input field
errors	Conditionally display a set of accumulated error messages
file	Render a file select input field
form	Define an input form
frame	Render an HTML frame element
hidden	Render a hidden field
html	Render an HTML `<html>` element
image	Render an input tag of type *image*
img	Render an HTML`<img>` tag
javascript	Render JavaScript validation based on the validation rules loaded by the validator plug-in
link	Render an HTML anchor or hyperlink
messages	Conditionally display a set of accumulated messages
multibox	Render a checkbox input field
option	Render a select option
options	Render a collection of select options

optionsCollection	Render a collection of select options
password	Render a password input field
radio	Render a radio button input field
reset	Render a *Reset* button input field
rewrite	Render an URI
select Render	A select element
submit	Render a *Submit* button
text	Render an input field of type text
textarea	Render a text area

Struts HTML tags in trade sample

The following tag is used for displaying the error message(s). If error objects (`ActionError`), which represent error messages, exist in the `ActionErrors` object, the error messages are rendered; otherwise nothing is rendered. The `ActionError` objects are usually added by `Action` or `ActionForm` classes.

```
<html:errors />
```

To use this tag, user have to define a header and a footer description in the application resource file `ApplicationResources.properties`. The following definitions are in the trade sample:

```
errors.header=<p class="errors">The Action failed because of the following
reason(s):<ul class="errors">
errors.footer=</ul></p>
error.login.failed=<li>Login Failed, please try again.
error.login.database=<li>Could not access database, please try again later.
```

In the `index.jsp`, an `html:form` tag is used for rendering the HTML form. The form is enclosed between `<html:form>` and `</html:form>`. A text field and password field are rendered using `html:text` and `html:password` tags. The field lengths are specified using the `size` attribute, and default values are in the `value` attribute. The *Submit* button is rendered using the `html:submit` tag.

```
<html:form action="/login">
<bean:message key="global.field.username" />
<html:text property="username" size="10" value="uid:1" />
<bean:message key="global.field.password" />
<html:password property="password" size="10" value="xxx" />
<html:submit property="submit">
<bean:message key="welcome.button.login" />
</html:submit>
</html:form>
```

Struts logic tags

This tag library contains tags that are useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.

equal Evaluate the nested body content of this tag if the requested variable is equal to the specified value.

forward Forward control to the page specified by the specified `ActionForward` entry.

greaterEqual Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.

greaterThan Evaluate the nested body content of this tag if the requested variable is greater than the specified value.

iterate Repeat the nested body content of this tag over a specified collection.

lessEqual Evaluate the nested body content of this tag if the requested variable is greater than or equal to the specified value.

lessThan Evaluate the nested body content of this tag if the requested variable is less than the specified value.

match Evaluate the nested body content of this tag if the specified value is an appropriate substring of the requested variable.

notEqual Evaluate the nested body content of this tag if the requested variable is not equal to the specified value.

notMatch Evaluate the nested body content of this tag if the specified value is not an appropriate substring of the requested variable.

notPresent Generate the nested body content of this tag if the specified value is not present in this request.

present Generate the nested body content of this tag if the specified value is present in this request.

redirect Render an HTTP Redirect

Struts template tags

Struts template tags enable you to create dynamic JSP templates for sharing common format pages. These templates are useful when a shared layout of the pages is likely to change.

get Retrieves content from a request scope bean, for use in the template layout.

insert Retrieves (or includes) the specified template file, and then inserts the specified content into the template's layout. By changing the layout defined in the template file, any other file that inserts the template will automatically use the new layout.

put Creates a request scope bean that specifies the content to be used by the get tag. Content can be printed directly or included from a JSP or HTML file.

Example 4-1 Trade sample index.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<%@ taglib uri="WEB-INF/lib/struts-html.tld" prefix="html" %>
<%@ taglib uri="WEB-INF/lib/struts-bean.tld" prefix="bean" %>
<jsp:useBean
   id="tickerBean" class="tradecommon.TickerBean"
   scope="application" />
<%
  tickerBean.setDowJones(tickerBean.perform(tickerBean.getDowJones()));
  tickerBean.setNasdaq(tickerBean.perform(tickerBean.getNasdaq()));
%>

<html>
<head>
<link href="theme/Master.css" rel="stylesheet" type="text/css">
<title>Welcome to Trade Sample</title>
</head>

<body background="theme/grid.gif">
<table height="100%" width="100%" cellpadding="0"
   cellspacing="0" border="0">
   <tr>
   <td valign="top" height="100pt"><img hspace="0" vspace="0"
      src="theme/tradebanner1.gif" border="0" width="100%"></td>
   <td valign="top" height="100pt"><img hspace="0" vspace="0"
      src="theme/tradebanner2.gif" border="0" width="100%"></td>
   </tr>
   <tr>
   <td colspan="2" valign="top">
      <table width="90%" border="0">
      <tr>
      <td align="right" width="90%"><b> <font size="-1">
         <bean:message key="market.text.title" />
         </font> </b></td>
      <td align="right"></td>
      </tr>
      <tr>
      <td align="right"><font size="-2">
         <bean:message key="market.text.dowJones" />
   </font></td>
<td align="right" style="background: #ddeeff"><font size="-1">
   <%=tickerBean.getDowJones() %> (<%=tickerBean.getDJChange() %>)
   </font></td>
</tr>
<tr>
<td align="right"><font size="-2">
   <bean:message key="market.text.nasdaq" />
   </font></td>
<td align="right" style="background: #ddeeff"><font size="-1">
   <%=tickerBean.getNasdaq() %> (<%=tickerBean.getNasdaqChange() %>)
   </font></td>
</tr>
<tr>
<td colspan="2"><center>
<h2>Welcome to the TRADE Brokerage home page. <br>
   Enter your username and password below to Login.</h2></center>
</td>
</tr>
<tr>
<td colspan="2">
   <table width="100%">
   <tr>
   <td><html:errors /></td>
   </tr>
   <tr>
   <td>
      <table align="right">
      <html:form action="/login">
      <tr>
      <td align="right"><b>
         <bean:message key="global.field.username" /></b></td>
      <td align="right">
         <html:text property="username" size="10" value="uid:1" />
      </td>
      </tr>
      <tr>

      <td align="right"><b>
         <bean:message key="global.field.password" /> </b></td>
      <td align="right">
         <html:password property="password" size="10" value="xxx" />
      </td>
      </tr>
      <tr>
      <td colspan="2" align="right">
         <html:submit property="submit">
         <bean:message key="welcome.button.login" />
         </html:submit>
```

```
                </td>
                </tr>
                </html:form>
                </table>
             </td>
             </tr>
             </table>
          </td>
          </tr>
          <tr>
          <td colspan="2"><font size="+1"> Not a member?? <a
             href="register.jsp">Click Here</a> to register. </font></td>
          </tr>
          </table>
       </td>
       </tr>
       <tr>
       <td colspan="2"><img src="theme/WEBSPHERE_22P.GIF" align="left">
          &nbsp;&nbsp;&nbsp;&nbsp; <bean:message key="global.createdBy" />
          <bean:message key="global.copyright" /></td>
       </tr>
       </table>
   </body>
</html>
```

5

Creating a Struts-based Web application

This chapter describes how to create a Struts-based Web application using WebSphere Application Enterprise Developer. We show you how to use Enterprise Developer's wizard to create the initial Struts Web application. This chapter also describes the components that are created by the wizard.

Note: You can create a Web project at J2EE level 1.2 or 1.3. A J2EE 1.3 project can only run in a WebSphere Version 5 server, whereas a J2EE 1.2 project can run in a WebSphere Version 4 or Version 5 server.

- In this book we describe how to create the J2EE 1.3 project and run it in a WebSphere Version 5 server.
- If you choose to create a J2EE 1.2 project and a WebSphere Version 4 server, then the server configuration for the data source is somewhat different, but you should be able to find your way through that.

Creating a Struts application

In this section we use the Enterprise Developer wizard to create a Struts-based Web application, then briefly explore what the wizard has created for us.

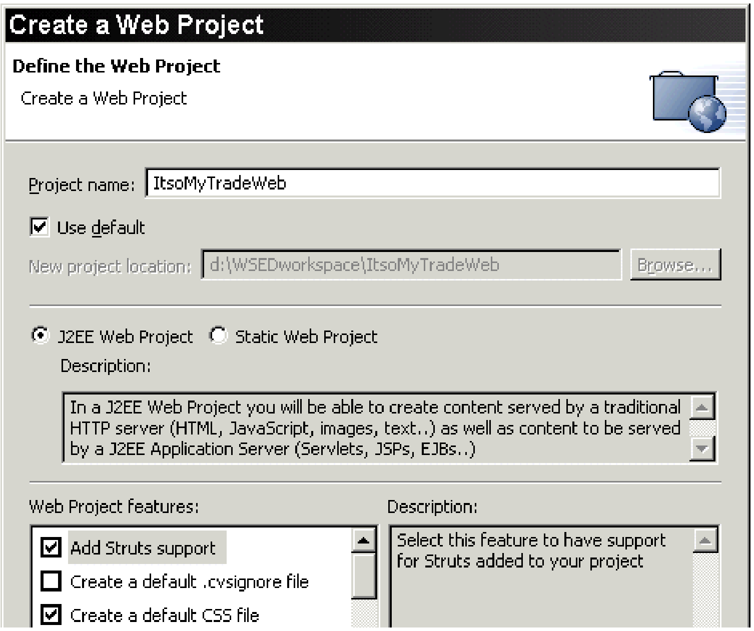
Because a Struts application is a type of Web application, we use the Web application wizard to create the Struts application.

Note: When editing files in WebSphere Studio Enterprise Developer, each file has a default editor associated with it. Double-clicking the file opens the file using the default editor. If the default editor is not your preference, you can select the file and select *Open With ->* from the context menu and then select the desired editor from the list.

Using the wizard to create a Web project

Starting from a Web perspective, you create a Struts-based application by performing these steps:

- Select the *New Wizard* icon, *Web* in the left-hand pane and *Web Project* in the right-hand pane, click *Next* (alternatively, you could select *File -> New -> Web Project*). The Define the Web Project dialog opens (Figure 5-1).



Create a Web Project

Define the Web Project

Create a Web Project

Project name: ItsoMyTradeWeb

☑ Use default

New project location: d:\WSEDworkspace\ItsoMyTradeWeb

● J2EE Web Project ○ Static Web Project

Description:

In a J2EE Web Project you will be able to create content served by a traditional HTTP server (HTML, JavaScript, images, text..) as well as content to be served by a J2EE Application Server (Servlets, JSPs, EJBs..)

Web Project features:

☑ Add Struts support

☐ Create a default .cvsignore file

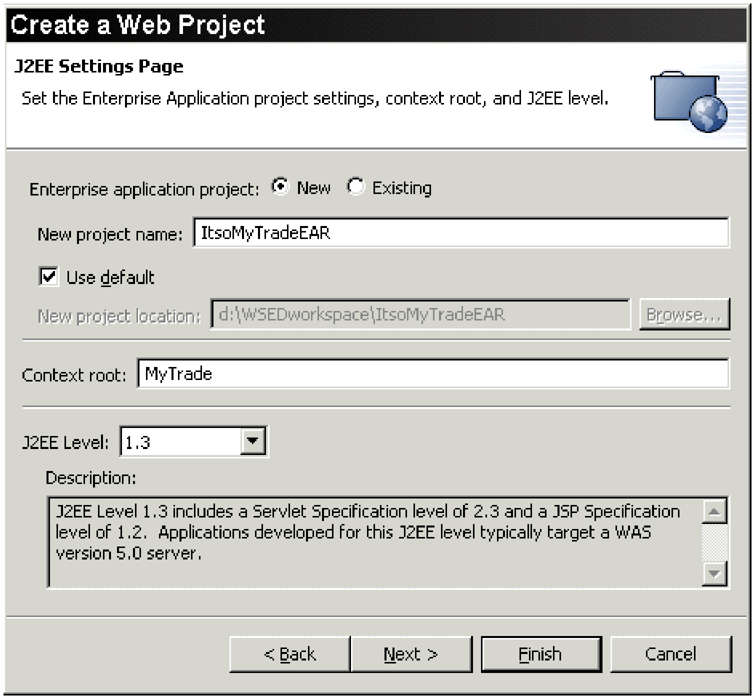
☑ Create a default CSS file

Description:

Select this feature to have support for Struts added to your project

Figure 5-1 Dialog for creating a new Web project

- There are several things to notice about this dialog:
  - You have to supply a project name for your Web application. We use `ItsoMyTradeWeb` for the trade example.
  - You can accept the default location where Enterprise Developer stores the files, or you can specify a location. We use the default location.
  - You can select a J2EE Web application or a static Web project. A static Web project consists of only items served by a traditional Web server - HTML, JavaScript, CSS files, image files, etc. In order to make a Web application that includes servlets, JSPs, or EJBs, you have to select *J2EE Web application*. This includes a Struts-based application.
  - This is where you designate to have Struts support added to your project. In order for the wizard to add Struts support, you select the *Add Struts support* check box.
- Click *Next* and the J2EE Settings Page dialog is next (Figure 5-2).



Create a Web Project

J2EE Settings Page

Set the Enterprise Application project settings, context root, and J2EE level.

Enterprise application project: ● New ○ Existing

New project name: ItsoMyTradeEAR

☑ Use default

New project location: d:\WSEDworkspace\ItsoMyTradeEAR

Context root: MyTrade

J2EE Level: 1.3

Description:

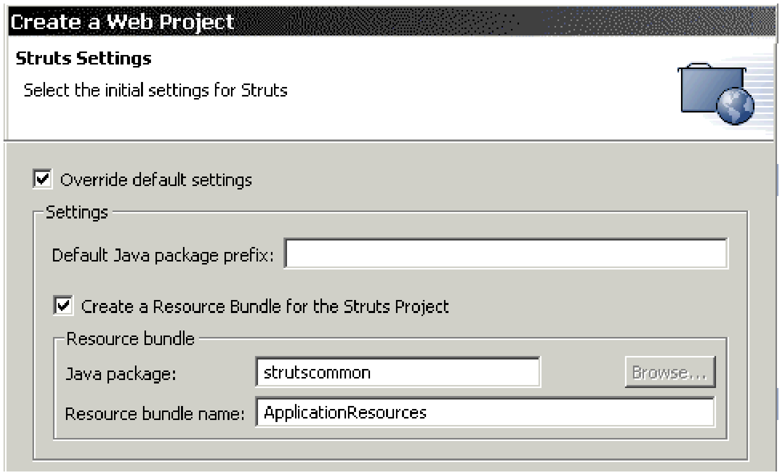
J2EE Level 1.3 includes a Servlet Specification level of 2.3 and a JSP Specification level of 1.2. Applications developed for this J2EE level typically target a WAS version 5.0 server.

Figure 5-2 Defining the J2EE settings for a Web project

  - Because we are creating a J2EE Web application, we have to create it as part of a J2EE enterprise application. You can either select an existing enterprise application or specify that a new enterprise application be created. If you choose to select an existing enterprise application, you can use the *Browse* button and select an enterprise application. We chose to have a new enterprise application created, and called it `ItsoMyTradeEAR`.

  - Specify the context root for the Web application. The default value is the name of the Web application, but you can also use a shorter alias name, such as `MyTrade`.
  - Select the J2EE specification level for the Web application. Because we are targeting WebSphere Application Server Version 5.0, we selected J2EE 1.3.

- Click *Next* and the Struts Settings dialog is displayed (Figure 5-3).



Create a Web Project

Struts Settings

Select the initial settings for Struts

☑ Override default settings

Settings

Default Java package prefix:

☑ Create a Resource Bundle for the Struts Project

Resource bundle

Java package: strutscommon

Resource bundle name: ApplicationResources

Figure 5-3 Struts Settings dialog

- The Struts Settings dialog lets you specify Struts-specific items for the Web application.

  Select the *Create Resource Bundle for the Struts Project* check box. This creates the file that contains the message definitions used by the Struts framework.

  By default, this file is called `ApplicationResources.properties`. You can specify a Java package for the properties file; we use a package name of `strutscommon`.

  > **Note:** It is good practice to use a package name rather than use the default package.

- Click *Finish* and your Struts-based Web application has been created.

J2EE Navigator view of the wizard’s output

We will now look at what the wizard generated for us. To begin, make sure you are in a Web perspective. Then look in the upper left view, called *J2EE Navigator* (Figure 5-4).

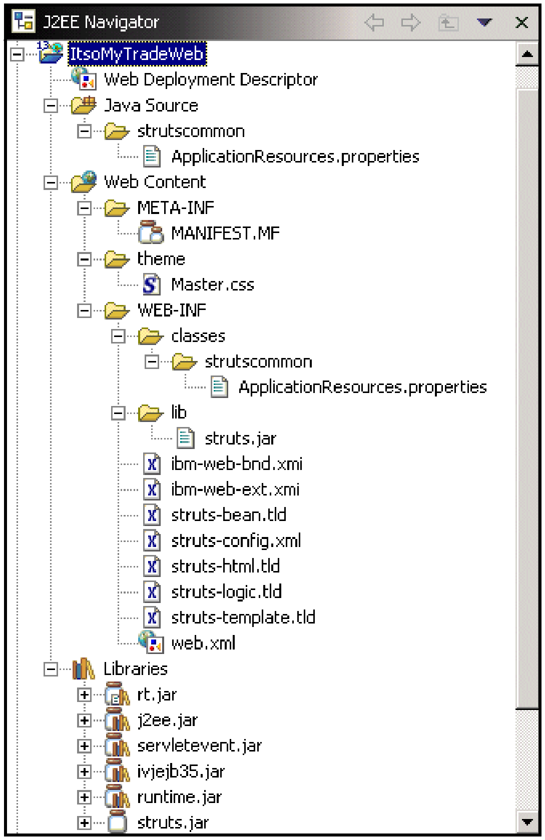


Figure 5-4 J2EE Navigator view of a Struts Web application

The J2EE Navigator view shows the Web projects you have defined in Enterprise Developer. The wizard has created the `ItsoMyTradeWeb` project. Expand the tree to see the folders and files what were created.

Deployment descriptor

The wizard creates the deployment descriptor resource, `web.xml`, which is the required deployment descriptor for a Web application, as defined by the servlet specification.

Java source files

The `Java Source` folder contains any of the Java source files you create (servlets, JavaBeans). In addition, any property files are stored in this folder.

The wizard creates the `ApplicationResources.properties` file and stores it in the package `strutscommon` within this folder. The wizard does not create any Java source files.

Web content files

The `Web Content` folder contains all HTML pages, JavaScript files, JSP, images and all other Web page resources. The wizard creates the `Web Content` folder.

META-INF

This folder contains the `MANIFEST.MF` file, where dependencies between projects are kept.

theme

You can store Web resources for your HTML pages in this subdirectory. The wizard creates the *theme* subdirectory with the default cascading style sheet `Master.css`.

WEB-INF

This is the subdirectory recommended by the servlet specification. The wizard creates the `WEB-INF` subdirectory. Inside this directory are:

- The `classes` subdirectory, which holds the servlet and utility compiled classes (including `ApplicationResources.properties`)
- The `lib` subdirectory, which holds any .jar files (including `struts.jar`)
- The `web.xml` deployment descriptor (this is the same as the Deployment Descriptor resource described above)
- The IBM bindings and extensions `.xmi` files
- The Struts configuration file (`struts-config.xml`)
- The Struts tag library files

Libraries

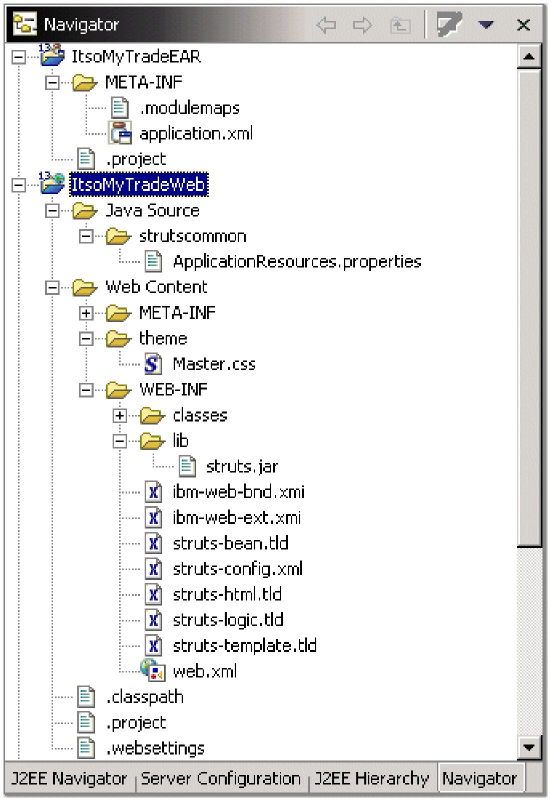
This folder contains any JAR files required by the application. It is the same as the `WEB-INF/lib` directory defined by the servlet specification.

The wizard creates the *Libraries* subdirectory and adds the `struts.jar` file `that` contains the Struts run-time classes. In addition J2EE JAR files, such as `rt.jar`, `j2ee.jar`, and so forth, are added to the Libraries folder.

Note: There is a repeat of some of the resources in the J2EE Navigator view. For example, the Deployment Descriptor resource that is listed immediately under the `ItsoMyTradeWeb` folder in Figure 5-4 on page 109 is the same as the `web.xml` file listed in the `WEB-INF` directory. Also, the Libraries resource is the same as the `Web Content/WEB-INF/lib` subdirectory. Keep in mind, then, that this view is showing you the same resources in multiple places.

Navigator view of the wizard's output

To see the actual folder and file structure, switch to the Navigator view (Figure 5-5). If the Navigator view is not open in the Web perspective, select *Window -> Show view -> Navigator* (you may have to find the view by selecting *Other -> Basic*).



```
ItsoMyTradeEAR
    META-INF
        .modulemaps
        application.xml
    .project
ItsoMyTradeWeb
    Java Source
        strutscommon
            ApplicationResources.properties
    Web Content
        META-INF
        theme
            Master.css
        WEB-INF
            classes
            lib
                struts.jar
            ibm-web-bnd.xmi
            ibm-web-ext.xmi
            struts-bean.tld
            struts-config.xml
            struts-html.tld
            struts-logic.tld
            struts-template.tld
            web.xml
    .classpath
    .project
    .websettings
```

Figure 5-5 Navigator view of the Struts project

The Navigator view shows the actual folders and files of the Web project. Java Source and Web Content are the main folders of the project. The wizard creates both these folders. The content of these folders was described in “J2EE Navigator view of the wizard’s output” on page 109.

The wizard also creates the .classpath, .project and .websettings files.

In the Navigator view, you can also see the ItsoMyTradeEAR project.

Standard Struts components

In this section we take a closer look at the Struts components that the wizard created for us.

Application resources properties file

Struts applications use a properties file for holding message text. The text can be for titles, links, buttons, error messages or any other text field. Using a resource file for the messages allows you to change the message text without having to change any of the Java code. It also facilitates translating the messages into other languages.

The wizard creates the properties file as ApplicationResources.properties; you can change the name and the package in the Struts Settings dialog, as shown in Figure 5-3 on page 108. The wizard places the file into the Java source package specified in the Settings dialog and also into the WEB-INF/classes subdirectory. The initial content of the file is minimal (Figure 5-6).

```
# Optional header and footer for <errors/> tag.
#errors.header=<ul>
#errors.footer=</ul>
```

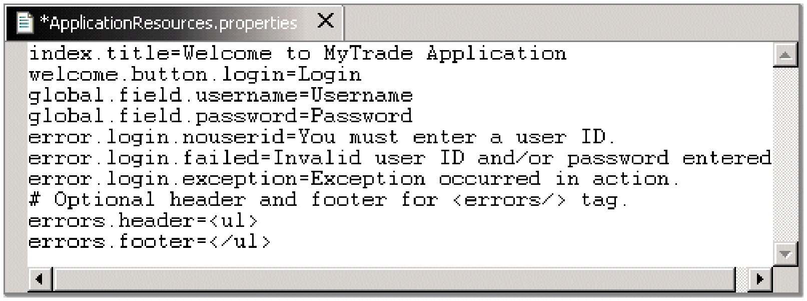
Figure 5-6 Initial application resources properties file

The name of the properties file is defined to the action servlet as its *application* initialization parameter. The file itself is full of *key=value* pairs, defining the message text for each key, for example:

```
error.login.failed=Invalid user ID and/or password entered.
```

Using a properties file, a Java class could reference error.login.failed when it determined that an unauthorized user ID/password combination was entered, rather than having to have the actual message text in the class.

Use the Properties File Editor to edit this file (Figure 5-7).



```
index.title=Welcome to MyTrade Application
welcome.button.login=Login
global.field.username=Username
global.field.password=Password
error.login.nouserid=You must enter a user ID.
error.login.failed=Invalid user ID and/or password entered
error.login.exception=Exception occurred in action.
# Optional header and footer for <errors/> tag.
errors.header=<ul>
errors.footer=</ul>
```

Figure 5-7 Properties file editor with sample application resources

Note that by removing the # signs from the header and footer you can tailor the output of error messages.

Struts run-time classes

The `struts.jar` file contains all the Struts run-time classes, as well as properties files, custom taglibs, and `.dtd` files. This is the same file that you would download as part of the Struts build from the Struts Web site if you were building a Struts application outside of Enterprise Developer.

The wizard places the `struts.jar` file in the `WEB-INF/lib` subdirectory.

Struts configuration file

The struts configuration file is used to describe the Struts application. The default name of this file is `struts-config.xml`. It is specified in the `web.xml` deployment descriptor as an initialization parameter of the action servlet.

The Struts action servlet (`ActionServlet`) reads this configuration file. This file defines two important elements, form beans and actions.

- A form bean represents the data from or to a particular form.
- The actions define action classes, including what input they use, any form beans the class uses, and the JSPs or other action classes the class can forward execution to. When you create an action class using the Struts wizard, you can specify that the action mapping be automatically added to the `struts-config.xml` file.

The wizard creates the `struts-config.xml` file in the `WEB-INF` subdirectory, but does not define any actions or form beans.

You can double-click the `struts-config.xml` file to edit it using the Struts configuration file editor. We will edit that file and cover form beans and actions in Chapter 6, “Adding JSPs and actions to the application” on page 117.

Struts taglibs

As defined in “Custom tags” on page 77, Struts includes four JSP tag libraries. The files are `struts-bean.tld`, `struts-html.tld`, `struts-logic.tld`, and `struts-template.tld`; these files are part of the Struts build.

The wizard places these files in the `WEB-INF` subdirectory.

Web deployment descriptor

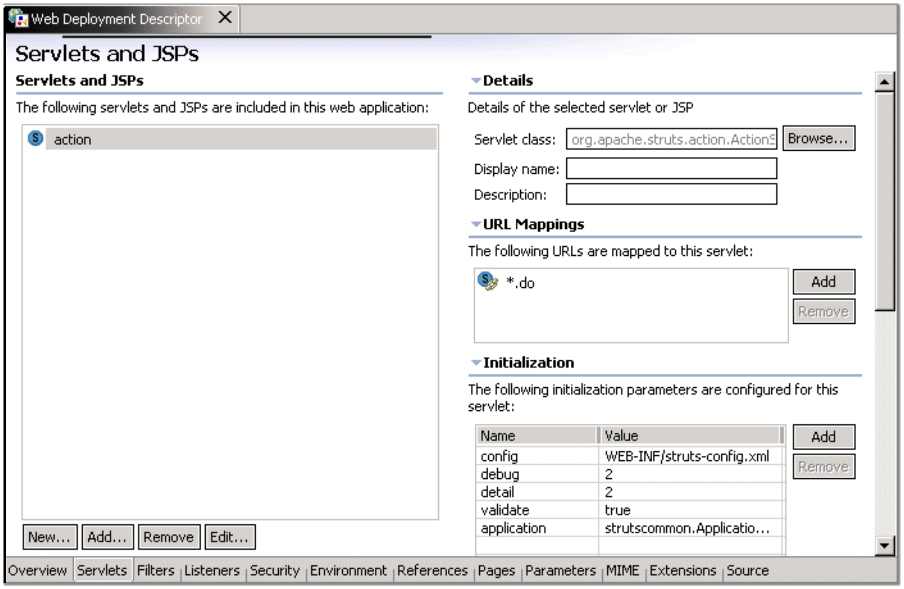
The `web.xml` file is the deployment descriptor for a Web application. It is used to define, among other things, the servlets and JSPs that make up the Web application.

Because a Struts application is a Web application, the Struts application has a `web.xml` file. The wizard creates the `web.xml` file in the `WEB-INF` directory. In addition, the wizard defines:

- `ActionServlet`, a servlet as the target of form actions
- Servlet mapping for the `ActionServlet`
- Initialization parameters for the `ActionServlet`

You can double-click the `web.xml` file to edit the file with the `web.xml` editor. Click the Servlets tab to see the servlet that was defined by the wizard (Figure 5-8):

- You can see that the servlet is called *action* and its type is `org.apache.sturts.action.ActionServlet`.
- The URL mapping defines that all URLs ending in *.do* will be mapped to this servlet.
- The initialization parameters include the configuration file and the resource file (`struts-config.xml`, `ApplicationResources`).



Servlets and JSPs

Servlets and JSPs

The following servlets and JSPs are included in this web application:

action

Details

Details of the selected servlet or JSP

Servlet class: org.apache.struts.action.ActionS

Display name:

Description:

URL Mappings

The following URLs are mapped to this servlet:

*.do

Initialization

The following initialization parameters are configured for this servlet:

Name	Value
config	WEB-INF/struts-config.xml
debug	2
detail	2
validate	true
application	strutscommon.Applicatio...

Figure 5-8 web.xml editor with the action servlet

Summary

In this chapter you used a wizard to create the structure and common components of a Struts-based Web application. You examined the components of the Web application that the wizard created for you. In the next chapter you will add actions and JSPs to the Web application.

6

Adding JSPs and actions to the application

This chapter builds upon the initial Struts Web application that the WebSphere Studio Enterprise Developer’s wizard creates. We show you how to add JSPs, action forms, actions and action mappings, and how to tie all these components together using the configuration file.

Overview

Chapter 5, “Creating a Struts-based Web application” on page 105 showed you how to create the common components of a Struts application. The Enterprise Developer wizard created the structure and components common to all Struts applications.

Recall the flow of a Struts application from Chapter 4, “Components of a Struts-based application” on page 67 (see Figure 6-1). The wizard created the action servlet. In this chapter we will add the rest of the components to complete a Struts application.

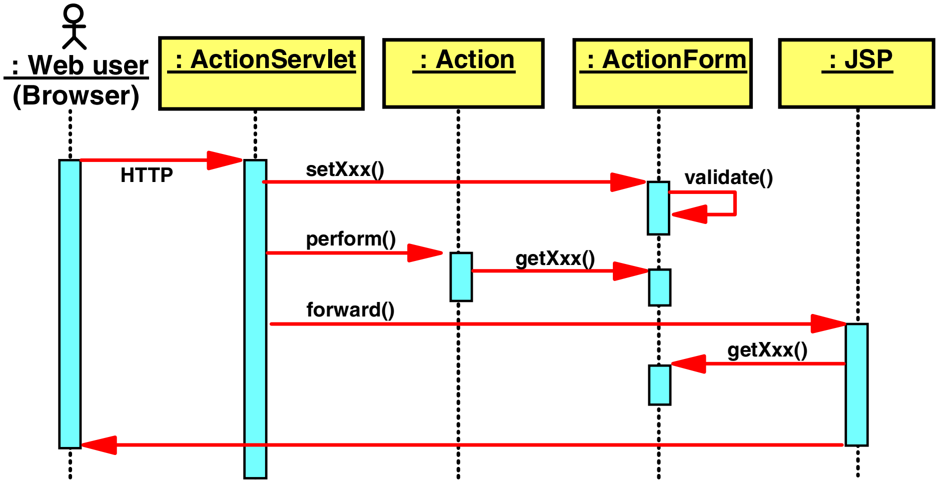


Figure 6-1 Flow of a Struts application

We address the following components and show you how they fit together to form the Struts application:

- JSPs
- `ActionForm`
- `Action`
- `ActionForward`
- `ActionMapping`
- `ActionError` and `ActionErrors`

The sample application that we will build is based on the diagram shown in Figure 6-2.

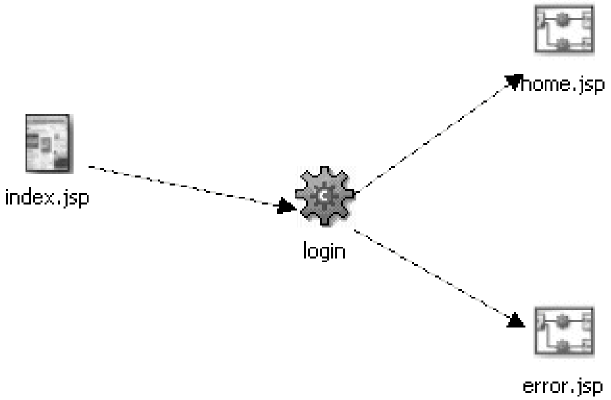


Figure 6-2 Flow of the sample Struts application

Users start with `index.jsp` and enter the user ID and password. When they submit the login form, the `ActionServlet` calls the login action. If the login is successful, control passes to `home.jsp`; otherwise, the user gets the error page (error.`jsp`), listing the errors and providing a link back to the login page.

Note: In the first implementation, we omit the `error.jsp` and route errors back to the `index.jsp` to display error messages.

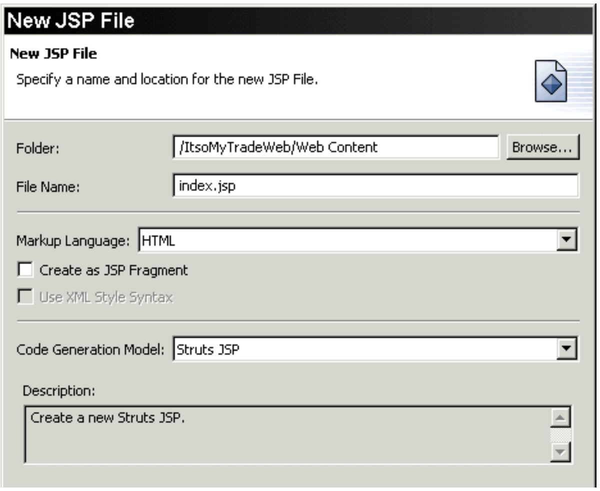
Creating JSP files

In this section we create the JSPs used in the sample application.

Creating the index.jsp

To create the `index.jsp` page, complete the following steps:

- Make sure you are in the Web perspective.
- In the J2EE Navigator view, select the `Web Content` folder in the `ItsoMyTradeWeb` project.
- Select *File -> New -> JSP File* and the dialog shown on Figure 6-3 appears:
  - Verify that the Folder field has the value `/ItsoMyTradeWeb/Web Content`.
  - In the File Name field, enter `index.jsp`.
  - Verify that the *Create as JSP Fragment* check box is not selected.
  - Verify that the Code Generation Model field selects *Struts JSP* to include the Struts taglib in the taglib directive.


New JSP File

New JSP File

Specify a name and location for the new JSP File.

Folder: /ItsoMyTradeWeb/Web Content

File Name: index.jsp

Markup Language: HTML

□ Create as JSP Fragment

□ Use XML Style Syntax

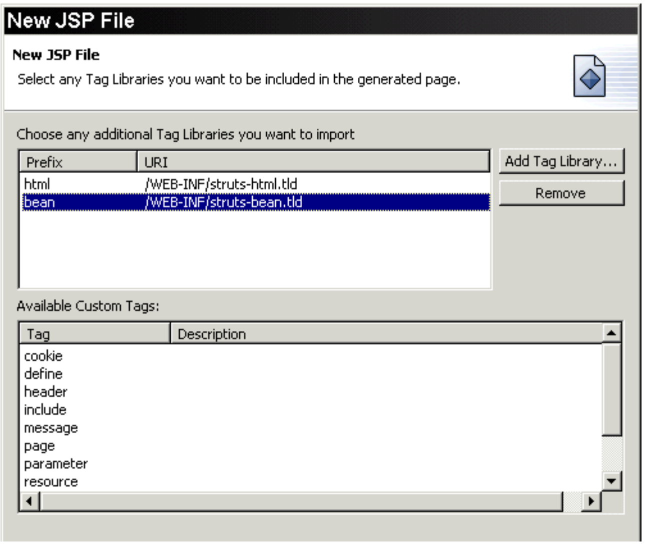
Code Generation Model: Struts JSP

Description:

Create a new Struts JSP.

Figure 6-3 Create a new JSP file

- Click *Next* and the Add Tag libraries dialog is next (Figure 6-4).


New JSP File

New JSP File

Select any Tag Libraries you want to be included in the generated page.

Choose any additional Tag Libraries you want to import

Prefix	URI
html	/WEB-INF/struts-html.tld
bean	/WEB-INF/struts-bean.tld

Available Custom Tags:

Tag	Description
cookie	
define	
header	
include	
message	
page	
parameter	
resource	

Figure 6-4 Add tag libraries

- In the top pane, `Struts-html` taglib and `Struts-bean` taglib are predefined. When the user selects a taglib in the list, the available tag and attribute list is shown in the bottom pane.

> **Note:** If you need to add additional tag libraries, you can click the *Add taglib* button and select additional tag libraries. We are not adding any beyond the HTML and bean tag libraries already selected.

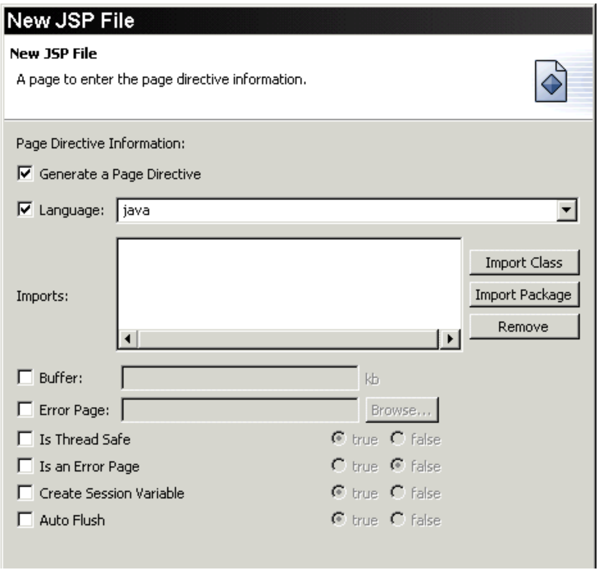
If you click *Add taglib*, perform these steps:

– Select the tag library in the taglibs list box

– At the prefix field, specify the prefix to use

– Click *OK* to define the taglib to use

- Click *Next* to proceed. The page directive information dialog is next (Figure 6-5).



New JSP File

New JSP File

A page to enter the page directive information.

Page Directive Information:

☑ Generate a Page Directive

☑ Language: java

Imports:

□ Buffer: kb

□ Error Page:

□ Is Thread Safe ● true ○ false

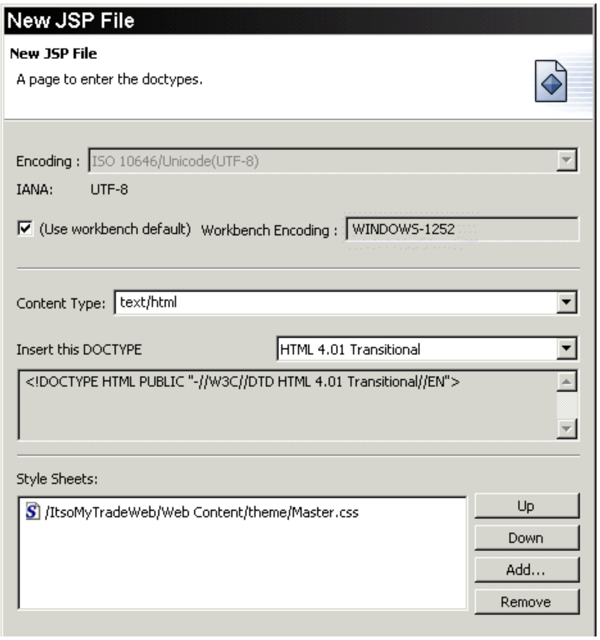
□ Is an Error Page ○ true ● false

□ Create Session Variable ● true ○ false

□ Auto Flush ● true ○ false

Figure 6-5 JSP page directive information

- In this dialog you can specify the variables for the page directive. Refer to Table 4-1 on page 92 for the meaning of each field.
- Click *Next* to proceed. The document type selection dialog is shown. In this dialog, the user can select the document type and the style sheet (Figure 6-6).


New JSP File

New JSP File

A page to enter the doctypes.

Encoding : ISO 10646/Unicode(UTF-8)

IANA: UTF-8

☑ (Use workbench default) Workbench Encoding : WINDOWS-1252

Content Type: text/html

Insert this DOCTYPE HTML 4.01 Transitional

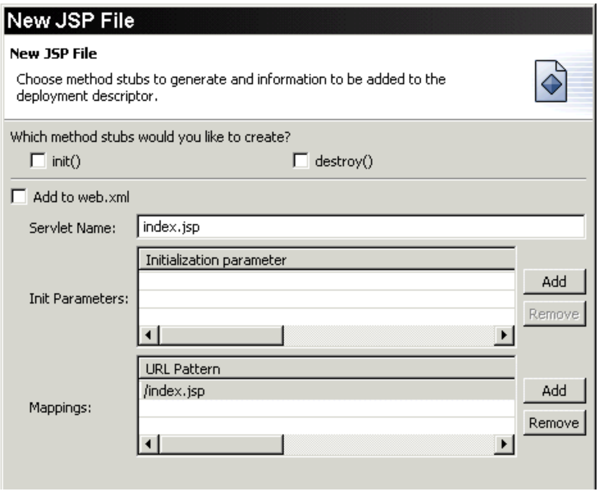
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Style Sheets:

/ItsoMyTradeWeb/Web Content/theme/Master.css

Figure 6-6 Document type selection

- Click *Next* and the method stub creation and registration to the deployment descriptor dialog is shown (Figure 6-7).


New JSP File

New JSP File

Choose method stubs to generate and information to be added to the deployment descriptor.

Which method stubs would you like to create?

□ init() □ destroy()

□ Add to web.xml

Servlet Name: index.jsp

Init Parameters:

Initialization parameter

Mappings:

URL Pattern
/index.jsp

Figure 6-7 Method stub creation and registration to the deployment descriptor

- If you want to create the `init` method or `destroy` method in the JSP page, select the appropriate check boxes at the top of the dialog. We do not create either of these methods.
- If you want to register this JSP page as a servlet, select the *Add to web.xml* check box. The default is to register the JSP in the deployment descriptor.
  - You can specify the servlet name and mapping using the corresponding fields. You can also specify the initialization parameters for the servlet in this dialog.

> **Note:** Typically you would not register a Struts JSP as a servlet.

- Click *Finish* to create the JSP.

Customizing index.jsp

The `index.jsp` opens in the page designer so that you can tailor the text and look of the JSP.

To customize `index.jsp`, replace the code between the `<BODY>` and `</BODY>` tags with the code shown in Figure 6-8. Use the Source tab to make the changes.

```
<h1 align="center"><bean:message key="index.title"/></h1>
<html:form action="/loginAction">
<html:errors/>
<p>
<table>
  <tr>
   <td><bean:message key="global.field.username"/></td>
   <td><html:text property="username" size="20" maxlength="30"/></td>
  </tr>
  <tr>
   <td><bean:message key="global.field.password"/></td>
   <td><html:password property="password" size="20" maxlength="30"/></td>
  </tr>
</table>
<p>
<html:submit><bean:message key="welcome.button.login"/>
</html:submit>
<input type="reset"><br>
</html:form>
```

Figure 6-8 Customizing index.jsp

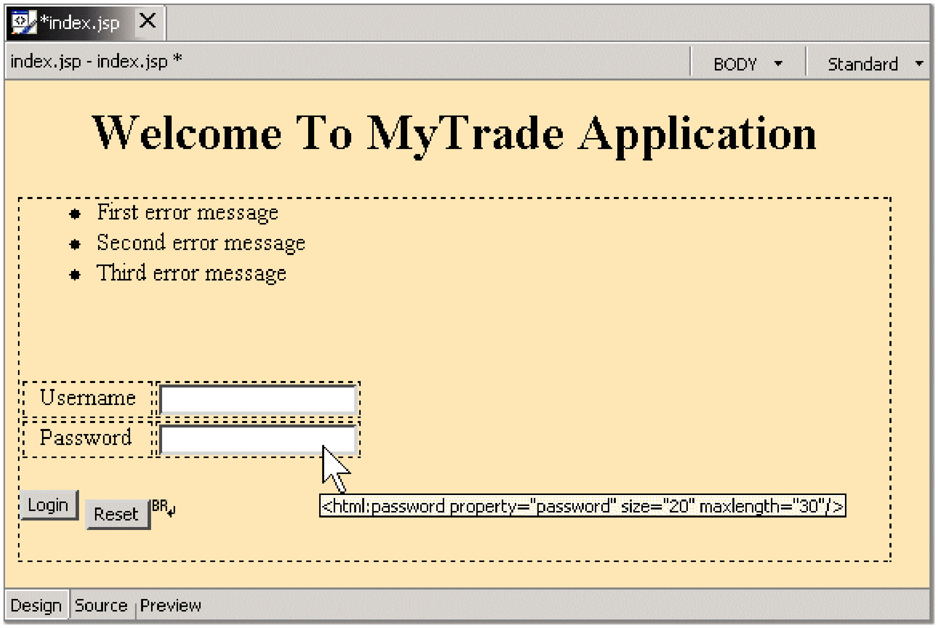
This page uses the Struts custom JSP tags to create a basic input page with two text fields (username and password) and submits the form to the action servlet specifying the `/loginAction` action.

Note: We will show how to insert custom tags in the design view of the page designer in “Tailoring the index.jsp with custom tags” on page 152.

The `<html:errors/>` tag displays any errors that are defined in an `ActionErrors` object.

Figure 6-9 shows the `index.jsp` in the design view of the page designer. Notice that the Struts custom tags have been replaced with the text from the application resource file.

When you hold the mouse over any of the custom tags, the corresponding source text is shown in the hover help.



Welcome To MyTrade Application

- First error message
- Second error message
- Third error message

```
<html:password property="password" size="20" maxlength="30"/>
```

Figure 6-9 Design view of index.jsp

Save the `index.jsp`. Note that a warning appears in the Tasks view because the form action `/LoginAction` points to an non-existing action class that we create later.

Creating home.jsp

To create the home.jsp page, use the JSP wizard in the same way you did to create index.jsp, the only difference being the name.

Customizing home.jsp

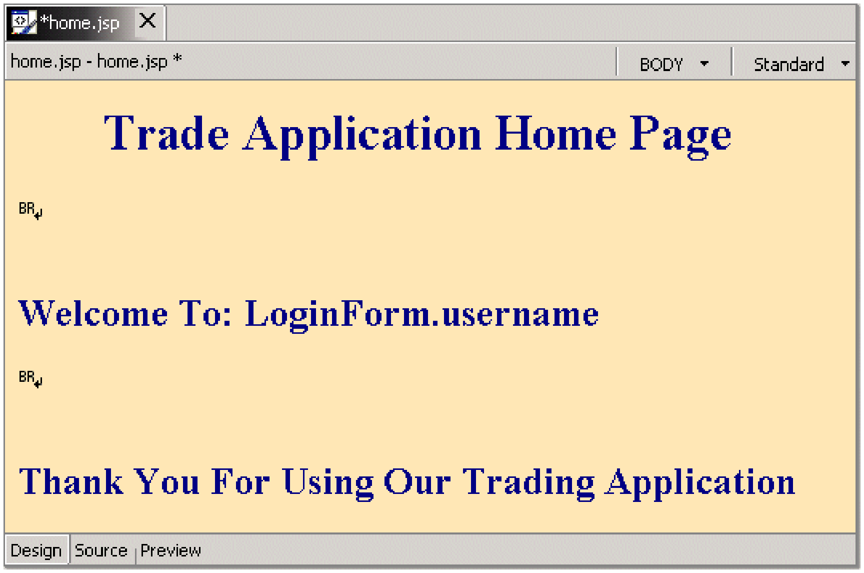
To customize home.jsp, replace the code between the <BODY> and </BODY> tags with the code shown in Figure 6-10.

```
<H1 align="center">Trade Application Home Page</H1>
<BR>
<H2>Welcome to: <bean:write name="loginForm" property="username"/></H2>
<BR>
<H2>Thank you for using our trading application</H2>
```

Figure 6-10 Customizing home.jsp

This page uses the Struts custom tags to retrieve the username from the LoginForm object. This is not really a home page and we show a *Thank you* text content instead of a page constructed with custom tags.

Figure 6-11 shows the home page in the design view of the page designer.



Trade Application Home Page

Welcome To: LoginForm.username

Thank You For Using Our Trading Application

Figure 6-11 Design view of home.jsp

Action forms

While JSPs provide the view component of a Struts application, action forms provide the state of the model that the JSPs (views) reflect. The data that the user inputs to the application through a JSP and the data the application returns to a JSP are processed using action forms.

An action form can be thought of as a JavaBean whose fields represent the data items in a corresponding JSP. Input from a JSP to the application passes through its associated action form. Similarly, application output to a JSP passes through its action form.

For example, a JSP that is used for login may contain a form that has two entry fields: username and password. The corresponding action form class for this JSP would have two fields—username and password—and getter/setter methods for each. Then, assuming successful login, the application returns a JSP that contains data retrieved from a database; this data would be passed to the JSP in the action form that is associated with the JSP.

Struts provides support for taking the parameters from the request object and storing them in the action form. You no longer have to process the parameters in your code. The action servlet takes care of this for you, before calling your action class.

Another function of the action form class is the validation of user-entered data. Your action form class can override the validate method, and you can then validate the user's data. Calling the validate method is optional, and is another function of the action servlet.

Creating the LoginForm class

An action form class extends `org.apache.struts.action.ActionForm`. For our sample application, we use a package named `strutscommon`, and a class named `LoginForm`. You can follow these steps to use the wizards to create the package and the class .

> **Note:** The method outlined here to create the package and class is just one way of doing it. There are other ways (some even quicker) to create the package and class. If you prefer another way, feel free to use it; just be sure to use the same names.

Create the package

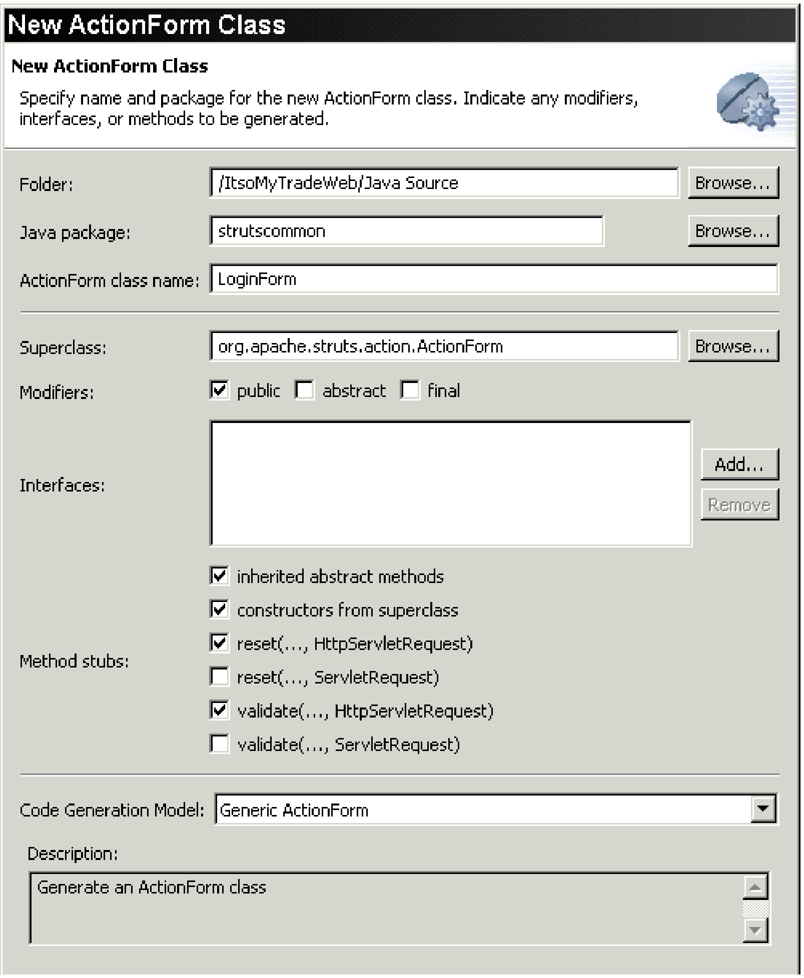
The `strutscommon` package was already created when we specified the `ApplicationResource` file.

- To create a Java package in the Web perspective you would:
  - Select the `Java Source` folder and *New -> Package,* enter the name of the package, and click *Finish*.

Create the LoginForm class

Next, to create the `LoginForm` class as an action form, perform these steps:

- Select *File -> New -> Other* (or use the *New* icon).
- In the New dialog, expand *Web*, and select *Struts* (in the left pane) and *ActionForm Class* (in the right pane), and click *Next* get the dialog shown in Figure 6-12.



New ActionForm Class

New ActionForm Class

Specify name and package for the new ActionForm class. Indicate any modifiers, interfaces, or methods to be generated.

Folder: /ItsoMyTradeWeb/Java Source

Java package: strutscommon

ActionForm class name: LoginForm

Superclass: org.apache.struts.action.ActionForm

Modifiers: ☑ public □ abstract □ final

Interfaces:

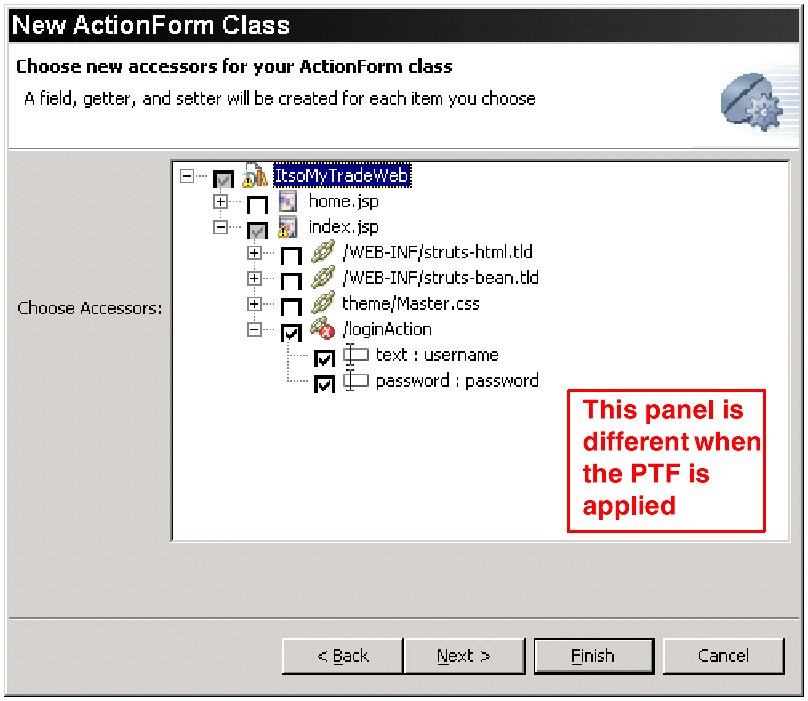
Method stubs:
☑ inherited abstract methods
☑ constructors from superclass
☑ reset(..., HttpServletRequest)
□ reset(..., ServletRequest)
☑ validate(..., HttpServletRequest)
□ validate(..., ServletRequest)

Code Generation Model: Generic ActionForm

Description:
Generate an ActionForm class

Figure 6-12 Creating the LoginForm class

- In the New ActionForm Class dialog:
  - Enter `LoginForm` as the class name.
  - Note the superclass, `org.apache.struts.action.ActionForm`.
  - Leave the default selections to *inherit abstract methods* and have *reset* and *validate* methods generated.
  - Select *constructors from superclass*.
  - Make sure the Code Generation Model is set to *Generic ActionForm*.
  - Click *Next*.
- In the Choose new accessors dialog, expand the `ItsoMyTradeWeb` project and select *username* and *password* in the `index.jsp` (Figure 6-13). Click *Next*.



New ActionForm Class

Choose new accessors for your ActionForm class

A field, getter, and setter will be created for each item you choose

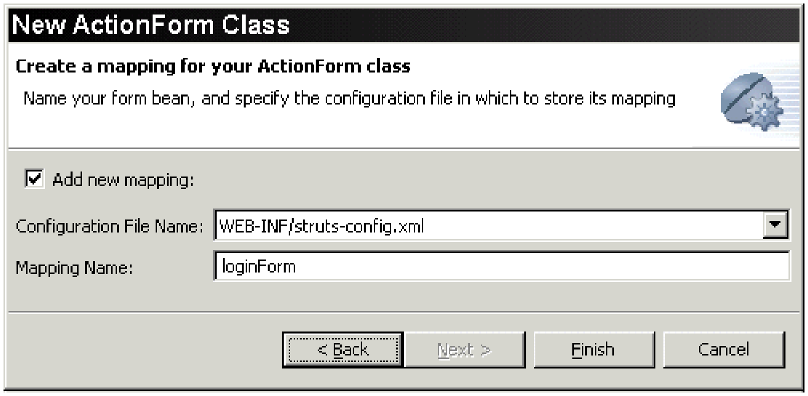
Choose Accessors:

```
⊟ ☑ ItsoMyTradeWeb
   ⊞ □ home.jsp
   ⊟ ☑ index.jsp
      ⊞ □ /WEB-INF/struts-html.tld
      ⊞ □ /WEB-INF/struts-bean.tld
      ⊞ □ theme/Master.css
      ⊟ ☑ /loginAction
           ☑ text : username
           ☑ password : password
```

This panel is different when the PTF is applied

Figure 6-13 Choose accessors for the LoginForm

- The Create new accessors dialog is next. The username and password are listed. Click *Next*.
- In the Create a mapping dialog, select *Add new mapping* (preselected), check the configuration file name, and leave the default mapping name of *loginForm* (Figure 6-14). Click *Finish* to generate the code.



New ActionForm Class

Create a mapping for your ActionForm class

Name your form bean, and specify the configuration file in which to store its mapping

☑ Add new mapping:

Configuration File Name: WEB-INF/struts-config.xml

Mapping Name: loginForm

Figure 6-14 Create the Struts mapping for the LoginForm

Customizing the LoginForm class

The `LoginForm` is open in the Java editor. The generated code is shown in Figure 6-15.

```
package strutscommon;
// import statements ...... not shown

public class LoginForm extends ActionForm {
   private java.lang.String username = null;
   private java.lang.String password = null;
   // getter and setter for username and password ...... not shown
   // constructor ...... not shown
   public void reset(ActionMapping mapping, HttpServletRequest request) {
      username = null;
      password = null;
   }
   public ActionErrors validate(ActionMapping mapping,
                                HttpServletRequest request) {
      ActionErrors errors = new ActionErrors();
      // Validate the fields in your form,
      // adding to this.errors as errors are found, e.g.
      // if ((field == null) || (field.length() == 0)) {
      //   errors.add("field", new ActionError("error.field.required"));
      // }
      return errors;
   }
}
```

Figure 6-15 Generated code for the LoginForm class (abbreviated)

Notice that the two fields from the *index.jsp* have been generated into the action form, including getter and setter methods.

The reset method

The `reset` method should be used to initialize the bean properties to their default state. This method is called before populating the properties from the HTML input form by the controller servlet.

The generated method resets both fields to `null`, so no tailoring is necessary.

The validate method

Recall that one of the functions the action form class can provide is to perform validation of the user's data. This is done in the `validate` method of the action form class. The action servlet calls the `validate` method after the action form has been processed and before the `perform` method of the action class is called.

The decision to include a validate method should be made in the bigger context of your application. Because the action form class knows nothing about the business logic for your application, it can do nothing more than simple field checking, for example:

- Are the mandatory fields filled in?
- Do fields that require all numbers contain all numbers?
- Do fields requiring alphanumeric entries contain valid alphanumeric character?

However, there will probably be additional validation done by the business logic, where additional knowledge about the fields is known. Keeping all such validation in a single place is one design consideration.

Another design consideration concerns the reusability of the business logic code. If your intent is to re-use the business logic, you would not want to put some of the basic input validation in the `validate` method, because you are essentially leaving it out of the business logic. This would require that all other modules interfacing with the business logic also implement the basic validation done in the `validate` method.

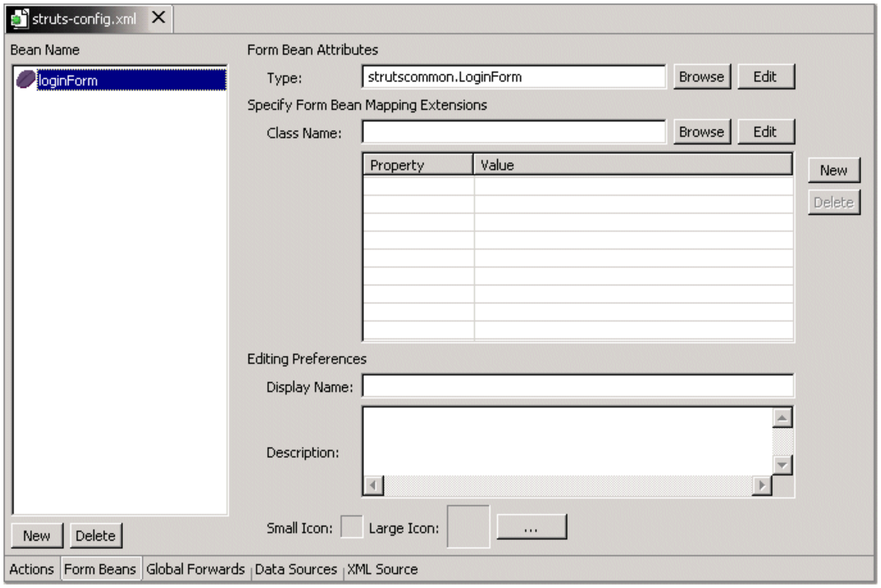
For now we will not add any validation logic and perform all validation in the business logic. Later, in "Implementing simple validation" on page 142, we will show a simple validation example.

Close the editor.

Checking the Struts configuration file

The `LoginForm` was automatically added to the Struts configuration file. Open the Struts configuration editor (double-click the `struts-config.xml` file).

Select the Form Beans tab and the `loginForm` entry (Figure 6-16).



Bean Name: loginForm

Type: strutscommon.LoginForm

Figure 6-16 LoginForm in Struts configuration editor

The important fields are:

name Unique identifier of this bean, used to reference it in corresponding action mappings. We use the `loginForm` name in `home.jsp`.

type Fully qualified Java class name of the implementation class to be used or generated.

classname Fully qualified Java class name of the `ActionFormBean` implementation class to use. It defaults to the value configured as the *formBean* initialization parameter of the Struts controller servlet.

Note: The current implementation does not create the `formBean` initialization parm in `web.xml` nor does it define the classname in `struts-config.xml` upon defining a new form bean unless you enter something for Specify Form Bean Mapping Extensions in Figure 6-16.

- Switch over to the Source tab, and in the XML file you can see the definition of the LoginForm within the <form-beans> section:

```
<!-- Form Beans -->
<form-beans>
   <form-bean name="loginForm" type="strutscommon.LoginForm">
   </form-bean>
</form-beans>
```

- Close the editor.

You have now defined a form bean (LoginForm), and associated a Java class with it (strutscommon.LoginForm).

Action forwards

The controller (ActionServlet) uses an ActionForward object to define the destination of a RequestDispatch.forward call. The business logic processing determines how the application proceeds, and passes that information back to the controller in the form of the ActionForward.

For example, when the users try to log in they may be successful or they may not. If they are successful, they may be granted access to the home page of the application. In this case, the business logic would set the destination of the ActionForward to be the home page JSP. If they are not successful, they may be sent back to the login page to try again. In this case, the business logic would set the destination of the ActionForward to be the index.jsp. The controller then passes control to the destination.

By default, the controller uses a RequestDispatch.forward to transfer control. You can specify that HttpServletResponse.sendRedirect be used instead.

You can use the ActionForward in conjunction with the ActionMapping to use named destinations. For instance, you can name a forward *success* pointing to the home.jsp and name another *failure* pointing back to the index.jsp. You then code your business logic to forward to success or failure as appropriate, and not have to worry about coding the actual JSP names in your logic. Defining forwards with ActionMapping is covered in “Action mappings” on page 136.

Action errors

Struts provides a simple way to collect errors and have messages available to any JSP. Individual errors are created as ActionError objects. A group of individual errors makes up an ActionErrors object.

There are two ways the ActionErrors object is used:

- First, ActionErrors is the return type for the ActionForm.validate method. If you implement ActionForm.validate, it must return an ActionErrors object encapsulating any errors it found. The controller (ActionServlet) determines that errors were found (by checking that the ActionErrors object is not null or empty) and forwards to the JSP specified by the input parameter of the ActionMapping.

  > **Note:** The input property specifies the JSP that should be returned when validation errors are discovered. The idea is to use a JSP similar to (if not exactly the same as) the JSP the user originally filled out. The JSP can be pre-populated with the values that were correct, and error messages will be displayed for values that failed the validation.

- Second, ActionError objects can be created by the business logic and added to an ActionErrors object. The business logic can then save the ActionErrors object using its Action.saveErrors method, which will make the ActionErrors object available to JSPs that use the Struts <html:errors/> tag.

Actions

The action class is where you implement the calls to your business logic (model) of your application. You can include all the business logic in your action class, or you can use the action class as a wrapper class for the business logic. The second approach is recommended, as it lends itself much better to code reuse of the business logic—you do not tie your business logic to a Struts class.

Action classes extend the class org.apache.struts.action.Action. The main processing is done in the perform method. This is the minimum method that you have to override. The action servlet finds or instantiates an action class, then calls its perform method. Note that only one instance of each action class is created by the action servlet, so the action classes must be thread-safe.

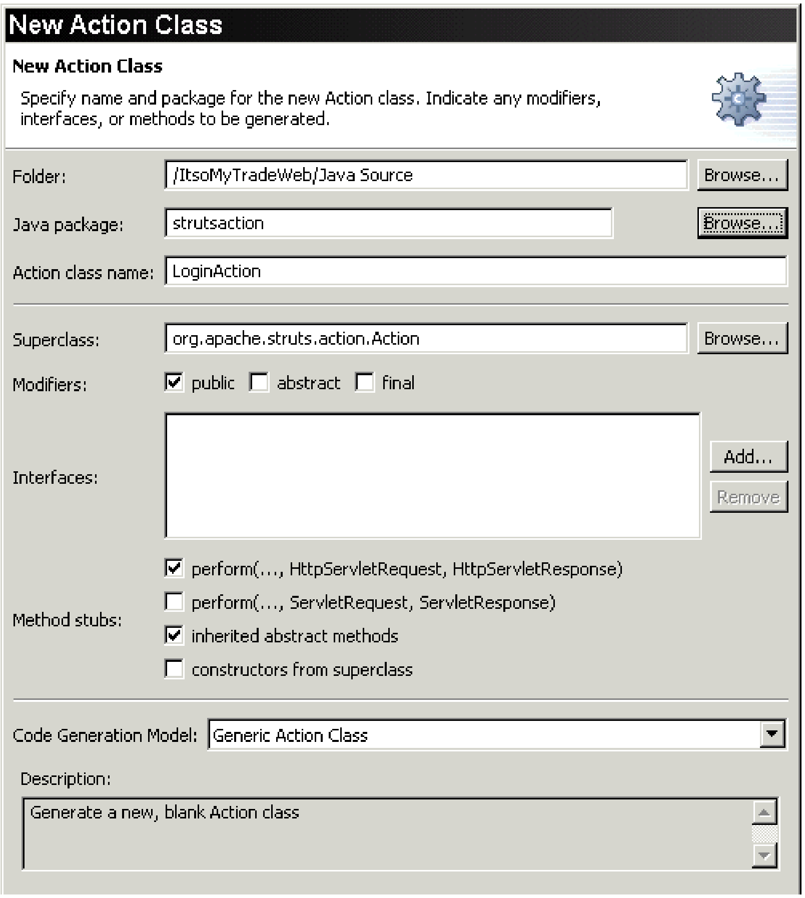
Creating the LoginAction class

You can create an action class using a wizard in the Enterprise Developer. For our sample application, we use a package named strutsaction, and for the login action a class named LoginAction. You can follow these steps to use the wizards to create the package and the class.

First, create a Java package named strutsaction in the Java Source folder.

Next, to create the action class:

- Select *File -> New -> Other*, expand *Web -> Struts* (in the left pane) and select *Action Class* (in the right pane) and click *Next*.
- The Create a generic Action class dialog opens (Figure 6-17).



New Action Class

New Action Class

Specify name and package for the new Action class. Indicate any modifiers, interfaces, or methods to be generated.

Folder: /ItsoMyTradeWeb/Java Source

Java package: strutsaction

Action class name: LoginAction

Superclass: org.apache.struts.action.Action

Modifiers: ☑ public ☐ abstract ☐ final

Interfaces:

Method stubs:
☑ perform(..., HttpServletRequest, HttpServletResponse)
☐ perform(..., ServletRequest, ServletResponse)
☑ inherited abstract methods
☐ constructors from superclass

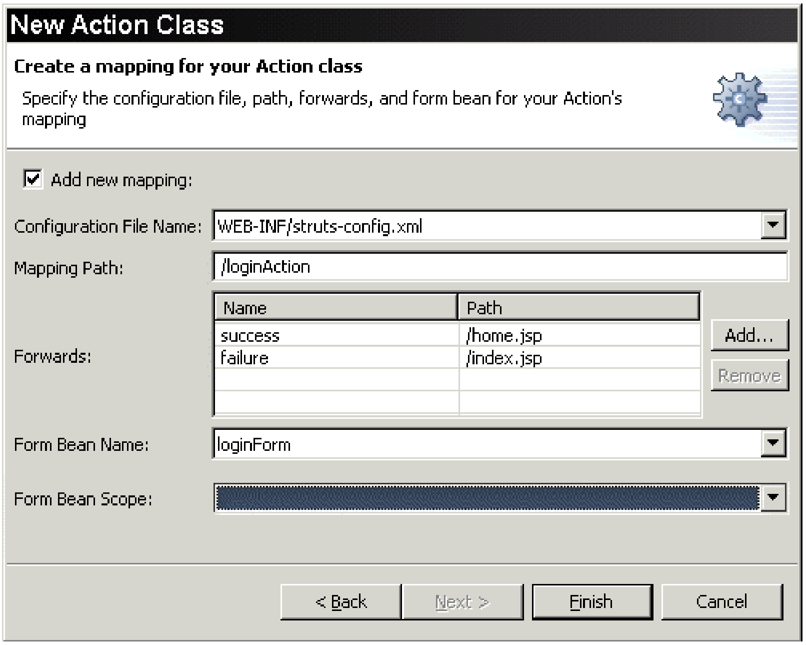
Code Generation Model: Generic Action Class

Description:
Generate a new, blank Action class

Figure 6-17 Defining the LoginAction class

- Enter a name of `LoginAction` and make sure the superclass is `org.apache.struts.action.Action`. and the package is `strutsaction`.
- Make sure the *perform(..., HttpServletRequest, HttpServletResponse)* and the *inherited abstract methods* check boxes are selected.
- Make sure the Code Generation Model is set to *Generic Action Class*.
- Click *Next*.

- In the Mapping Description dialog (Figure 6-18):
  - Set the Mapping Path to `/loginAction`.
  - Click *Add* twice for Forwards. Change the two Name fields to *success* and *failure*, and the two Path fields to `/home.jsp` and `/index.jsp`.
  - Select `loginForm` from the Form Bean Name drop-down.



New Action Class

Create a mapping for your Action class

Specify the configuration file, path, forwards, and form bean for your Action's mapping

☑ Add new mapping:

Configuration File Name: WEB-INF/struts-config.xml

Mapping Path: /loginAction

Forwards:

Name	Path
success	/home.jsp
failure	/index.jsp

Form Bean Name: loginForm

Form Bean Scope:

Figure 6-18 Mapping for the Login action

- Click *Finish*.

Customizing the LoginAction class

The `LoginAction` file opens in the editor. The signature of the `perform` method is shown in Figure 6-19.

```
public ActionForward perform(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
                       throws IOException, ServletException
```

Figure 6-19 Signature of the perform method

Note that the action class has access to the action mapping (discussed in “Action mappings” on page 136), the action form (which gives the action access to the data the user entered) and the request and response objects. The action class must return an action forward object, which is used to tell the action servlet where to pass control to next.

Add the code shown in Figure 6-20 to the `try/catch` block inside the `perform` method.

```
try {
   // do something here
   String userID = loginForm.getUsername();
   if (!userID.equals("userid")) {
         errors.add("login", new ActionError("error.login.failed"));
   }
} catch (Exception e) {
   // Report the error using the appropriate name and ID.
   errors.add("login", new ActionError("error.login.exception"));
}
```

Figure 6-20 Customizing the LoginAction class

This code checks that the user ID is the string `userid`, and if not causes an `ActionError` to be created.

Save the code and close the editor.

Action mappings

So far we have added JSPs, action forms, and actions to our basic Struts application. We have added action forwards and action errors to our action. We will now tie all those pieces together using action mappings to complete the login portion of our application.

See “Action mappings” on page 82 for a list of the action mapping properties.

The action mappings are defined in the `struts-config.xml` file. Each of these properties can be defined in the `struts-config.xml` file using the Enterprise Developer’s Struts configuration file editor.

Editing struts-config.xml

To edit the `struts-config.xml` file, start in a Web perspective and Web or Navigator view. The file is in the `WEB-INF` subdirectory. To verify that the Struts configuration file editor is the default editor, select the file, right-click it and select *Open With -> Struts Configuration File Editor*.

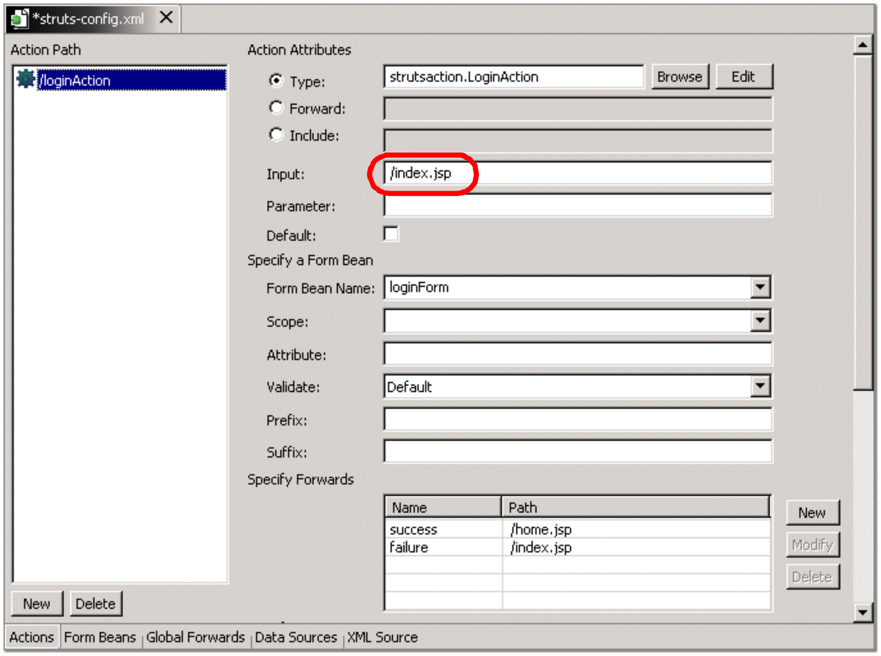
The editor opens. Notice the tabs at the bottom of the editor. There are tabs for Actions, Form Beans, Global Forwards and Data Sources. These are the four areas defined by this configuration file.

Completing the login action

The login action is visible in the editor (Figure 6-18); it was added when the `LoginAction` class was defined.

All we have to add to this mapping is `/index.jsp` as the input. Save your action mapping.

You should examine the XML source of the configuration file by clicking the XML Source tab. You will be able to see the form beans and actions that you have defined.



*struts-config.xml

Action Path

/loginAction

Action Attributes

● Type: strutsaction.LoginAction

○ Forward:

○ Include:

Input: /index.jsp

Parameter:

Default: □

Specify a Form Bean

Form Bean Name: loginForm

Scope:

Attribute:

Validate: Default

Prefix:

Suffix:

Specify Forwards

Name	Path
success	/home.jsp
failure	/index.jsp

Figure 6-21 Struts configuration file editor

Testing the Struts application

With the setup of the basic Struts application completed, we can test the application in the built-in WebSphere Application Server.

Define a server project

To test Web and EJB applications we have to define test servers. Such definitions are best stored in a server project.

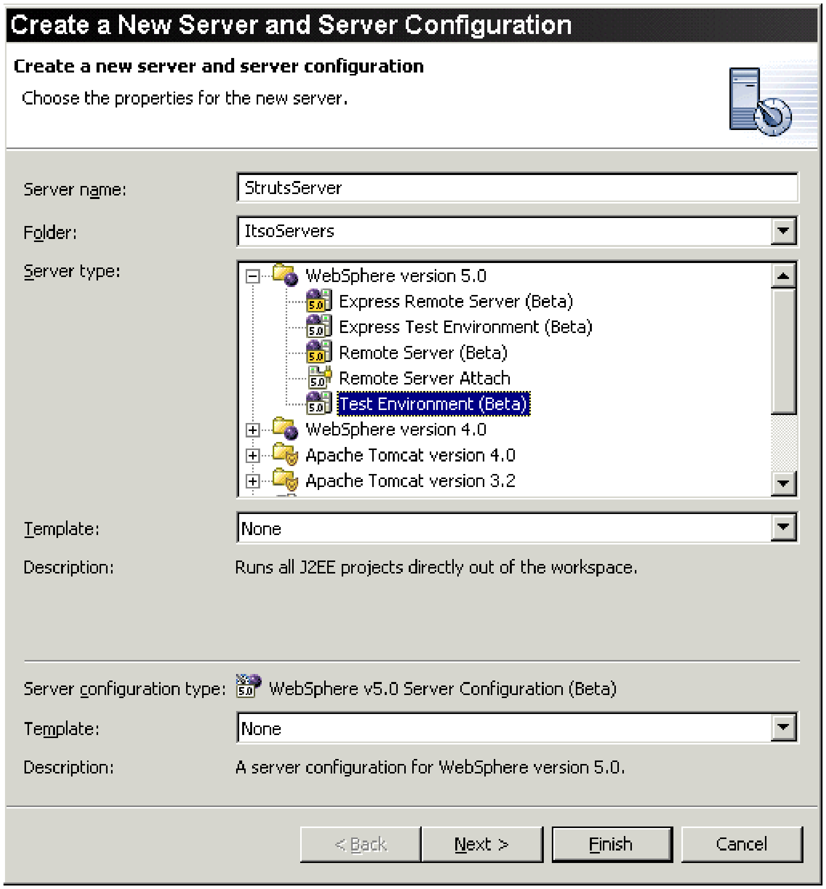
To define a server project:

- Open the Server perspective.
- In the New dialog (click *File -> New -> Other*), select *Server* in the left pane and *Server Project* in the right pane and click *Next*.
- Enter `ItsoServers` as the project name and click *Finish*.

Define a WebSphere test server

To define a test server, we need a server and a server configuration:

- Click the *Create Server and Server Configuration* icon in the tool bar (or in the New dialog select *Server -> Server and Server Configuration*).
- In the dialog (Figure 6-22), enter `StrutsServer` as the name, expand *WebSphere Version 5.0* and select *Test Environment*.
- Optionally click *Next* and see the default port number (9080).
- Click *Finish*.



Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: StrutsServer

Folder: ItsoServers

Server type:

- WebSphere version 5.0
  - Express Remote Server (Beta)
  - Express Test Environment (Beta)
  - Remote Server (Beta)
  - Remote Server Attach
  - Test Environment (Beta)
- WebSphere version 4.0
- Apache Tomcat version 4.0
- Apache Tomcat version 3.2

Template: None

Description: Runs all J2EE projects directly out of the workspace.

Server configuration type: WebSphere v5.0 Server Configuration (Beta)

Template: None

Description: A server configuration for WebSphere version 5.0.

Figure 6-22 Define the StrutsServer

The `StrutsServer` appears in the Servers view, and the server and configuration appear in the Server Configuration view.

Add the enterprise application project to the server configuration:

- In the Server Configuration view, expand Server Configurations.
- Select the `StrutsServer` and *Add -> ItsoMyTradeEAR* from the context menu.
- Figure 6-23 shows the Server Configuration view with the `StrutsServer` and the EAR project.

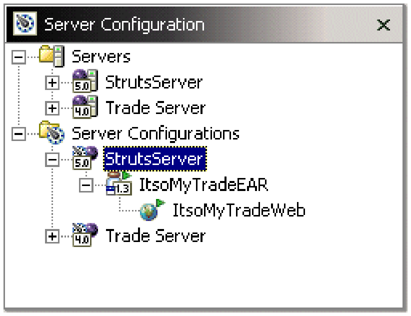


Figure 6-23 StrutsServer configuration with project

Running the Struts application

We can start the `StrutsServer` in two ways:

- Explicit start from the Servers view.
- Implicit start by selecting an HTML or JSP file (or a Web project) and selecting *Run on Server* or *Debug on Server* from the context menu.

Start the server

Expand the `ItsoMyTradeWeb` application, select the `index.jsp` file and *Run on Server* from the context menu.

In the Select a Server dialog, select the `StrutsServer`, select *Set this server as the preferred server* (to bypass this dialog in the future), and click *Finish*.

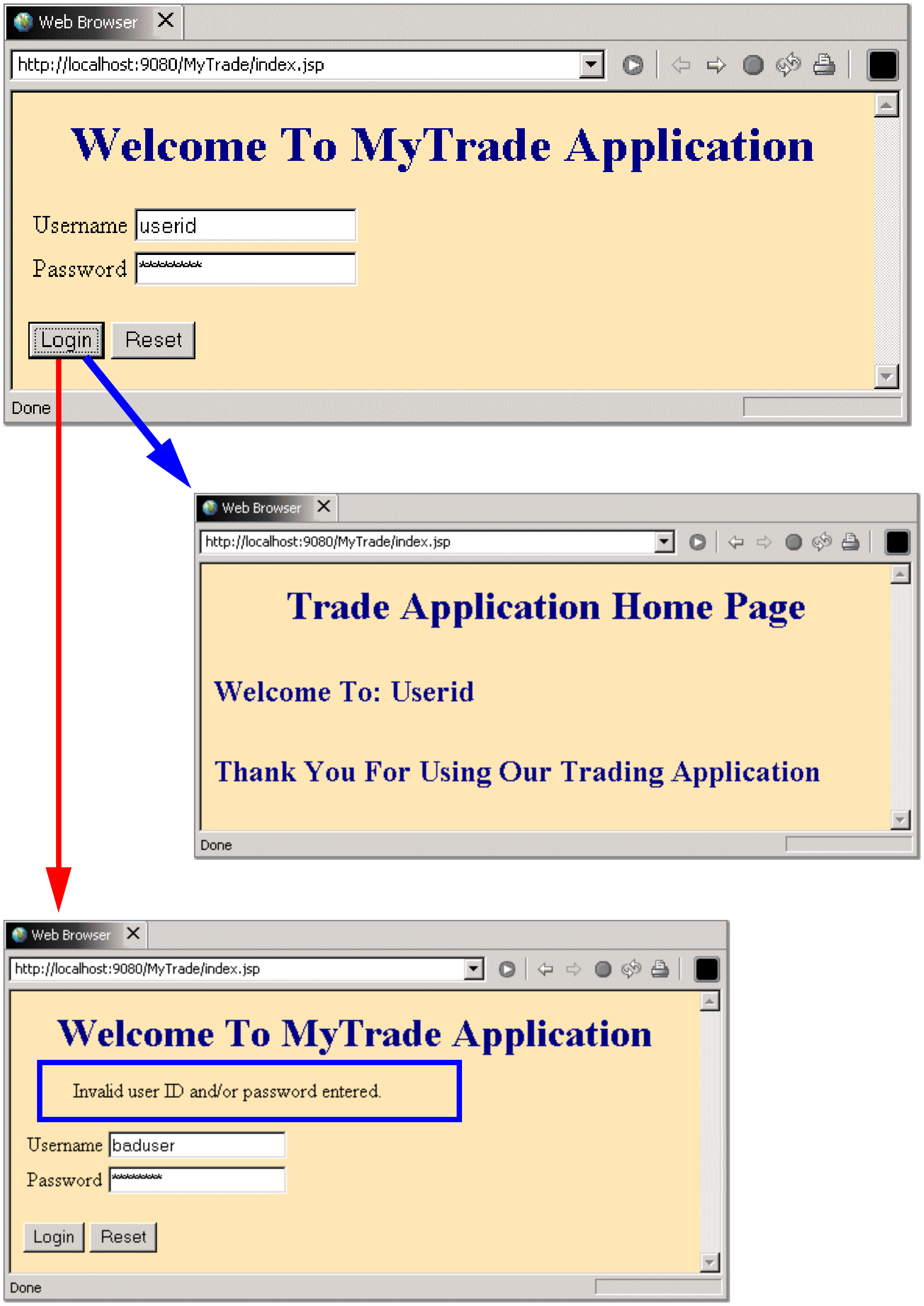
The application is published to the server and the server starts. Watch the Console view for messages. The server is ready when the message *Server open for e-business* appears.

Test the application

The `index.jsp` is displayed (Figure 6-24).

- Enter *userid* for the username field and any password, then click *Login*. This action should display the home JSP.
- Enter any other user ID and the `index.jsp` is redisplayed with an error message:

```
Invalid user ID and/or password entered.
```



http://localhost:9080/MyTrade/index.jsp

Welcome To MyTrade Application

Username userid

Password ********

http://localhost:9080/MyTrade/index.jsp

Trade Application Home Page

Welcome To: Userid

Thank You For Using Our Trading Application

http://localhost:9080/MyTrade/index.jsp

Welcome To MyTrade Application

Invalid user ID and/or password entered.

Username baduser

Password ********

Figure 6-24 Struts application test run

Implementing simple validation

Now let's try a simple validation. Open the LoginForm bean (in strutscommon), add an import statement, and add validation logic to the validate method (Figure 6-25).

```
import org.apache.struts.action.ActionError;
......
public ActionErrors validate(
   ActionMapping mapping,
   HttpServletRequest request) {
   org.apache.struts.action.ActionErrors errors =
      new org.apache.struts.action.ActionErrors();
   // validation logic ......
   if (username.trim().equals(""))
      errors.add("login", new ActionError("error.login.nouserid"));
   return errors;
}
```

Figure 6-25   Adding simple validation

Restart the server and retest the application. If you leave the user ID field empty, the error message from the ApplicationResources file is displayed (Figure 6-26).

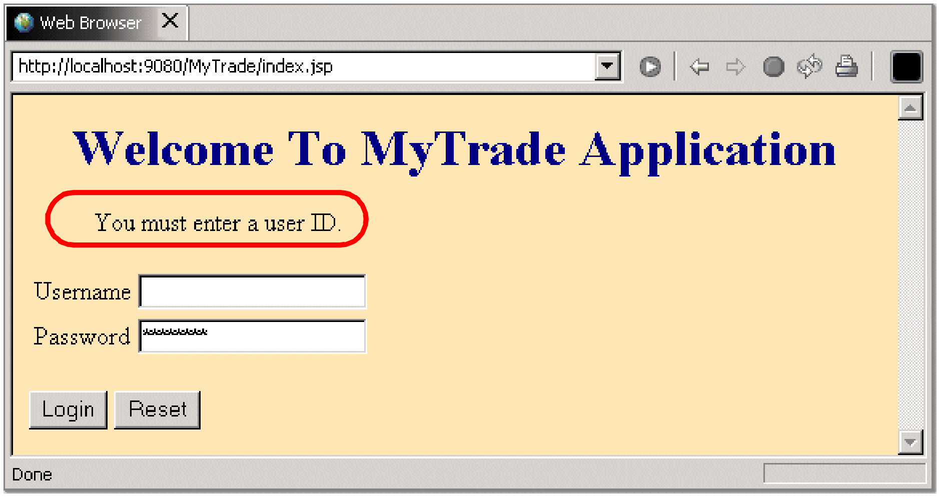

Web Browser
http://localhost:9080/MyTrade/index.jsp
Welcome To MyTrade Application
You must enter a user ID.
Username
Password
Login
Reset
Done

Figure 6-26   Validation error message

Stop the server.

7

Struts application diagram editor

In this chapter, we create a simple Web application to show the usage of the Struts application diagram editor.

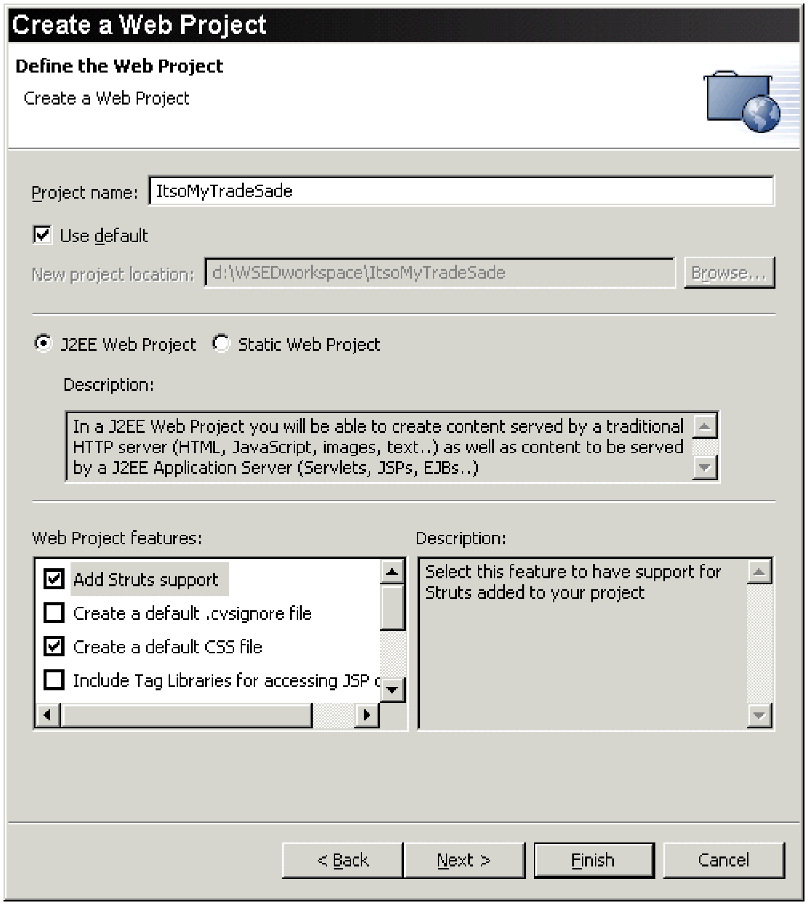
Our sample Web application has a simple flow:

- A welcome page is displayed initially.
- A user can enter a user ID and a password on the welcome page and click *Submit*.
- The server invokes a Struts action class to verify the user ID and password.
- If the authentication is successful, the user can proceed to the home page of the application.
- If the authentication fails, an error page is shown and the user can go back to the welcome page.

To implement this Web application, we use the Struts application diagram editor, from which we can implement the JSPs and the action.

Create a Web project for the Struts application

In the Web view of the Web perspective we create a Web project named `ItsoMyTradeSade` for our Struts application (Figure 7-1).



Create a Web Project

Define the Web Project

Create a Web Project

Project name: ItsoMyTradeSade

☑ Use default

New project location: d:\WSEDworkspace\ItsoMyTradeSade

● J2EE Web Project ○ Static Web Project

Description:

In a J2EE Web Project you will be able to create content served by a traditional HTTP server (HTML, JavaScript, images, text..) as well as content to be served by a J2EE Application Server (Servlets, JSPs, EJBs..)

Web Project features:

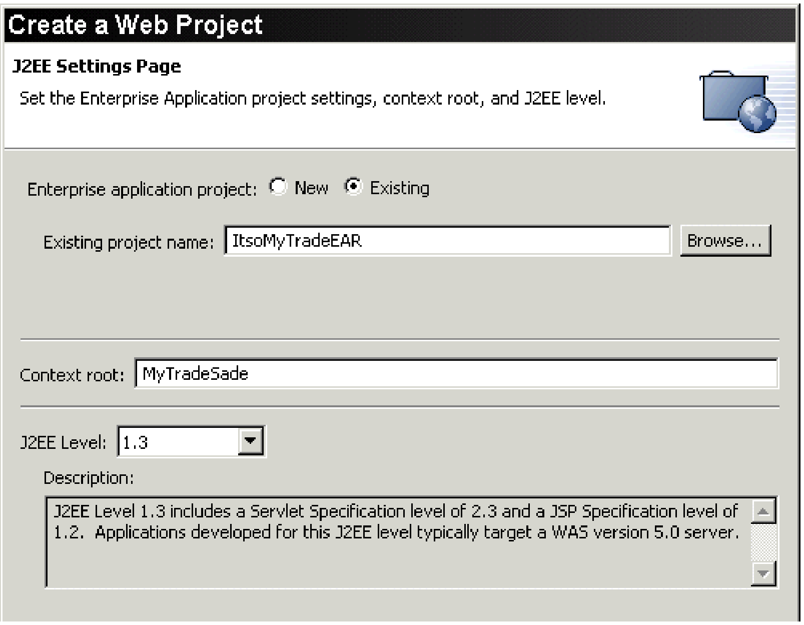
☑ Add Struts support
□ Create a default .cvsignore file
☑ Create a default CSS file
□ Include Tag Libraries for accessing JSP c

Description:

Select this feature to have support for Struts added to your project

Figure 7-1 Create Web project for Struts diagram editor application

- Make sure that *Add Struts support* is selected.
- Click *Next* and enter the J2EE settings (Figure 7-2).



Create a Web Project

J2EE Settings Page

Set the Enterprise Application project settings, context root, and J2EE level.

Enterprise application project: ○ New ● Existing

Existing project name: ItsoMyTradeEAR

Context root: MyTradeSade

J2EE Level: 1.3

Description:

J2EE Level 1.3 includes a Servlet Specification level of 2.3 and a JSP Specification level of 1.2. Applications developed for this J2EE level typically target a WAS version 5.0 server.

Figure 7-2 J2EE settings of the Struts diagram editor project

- Select *Existing* for the Enterprise Application Project radio button and select `ItsoMyTradeEAR` as the EAR project name.
- Enter `MyTradeSade` as the context root.
- Click *Finish* to create the Web project.

We skipped the other panels. Therefore the `ApplicationResources.properties` file is created in the default package (directly in the Java Source folder).

Important: You get a prompt to repair the server configuration and add the new Web project to the `StrutsServer`, because it belongs to the `ItsoMyTradeEAR` enterprise application that is attached to the server. Click *OK*.

To design the Struts application in the Workbench, make these views active in the Web perspective:

- J2EE Navigator in the top left pane.
- Select the Web Structure view in the bottom left pane.
- Select the Properties view in the bottom right pane (use *Window -> Show View -> Other -> Basic -> Property* if the view is not open).

Create the application resources

Edit the `ApplicationResources` file and add the same content to the file as for the `ItsoMyTradeWeb` application (Figure 5-3 on page 108).

Figure 7-3 shows the complete `ApplicationResources` file with two extra error message lines and the error header and footers shown in boldface.

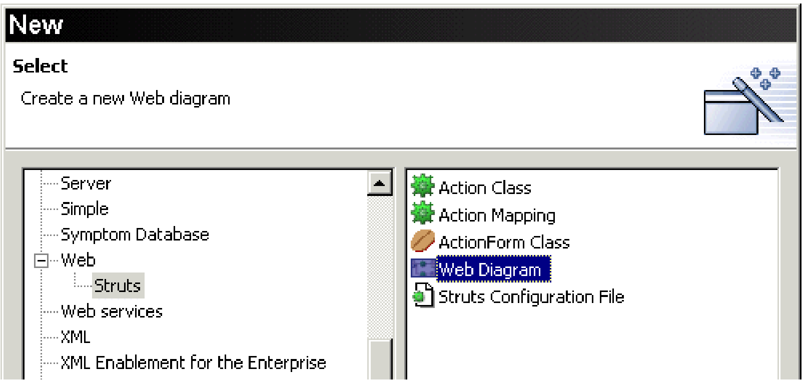
```
index.title=Welcome to MyTrade Application
welcome.button.login=Login
global.field.username=Username
global.field.password=Password
error.login.nouserid=You must enter a user ID.
error.login.failed=Invalid user ID and/or password entered.
error.login.exception=Exception occurred in action.
error.invalidUsername=User ID is invalid.
error.invalidPassword=Password is invalid
# Optional header and footer for <errors/> tag.
errors.header=<font color="red"><ul>
errors.footer=</ul></font>
```

Figure 7-3 Application resources file

Create a Struts application diagram file

To create a Struts application diagram file, follow these steps:

- Select the *Web Content/WEB-INF* folder in the *Web* view and *New -> Other*. In the New dialog, select *Web -> Struts* in the left pane, *Web Diagram* in the right pane, and click *Next* (Figure 7-4).



New

Select

Create a new Web diagram

Figure 7-4 Create new Struts application diagram

- Specify *Login* for the file name and click *Finish.*

The Struts application diagram file `Login.gph` is created in the Workbench and the editor opens (Figure 7-5).

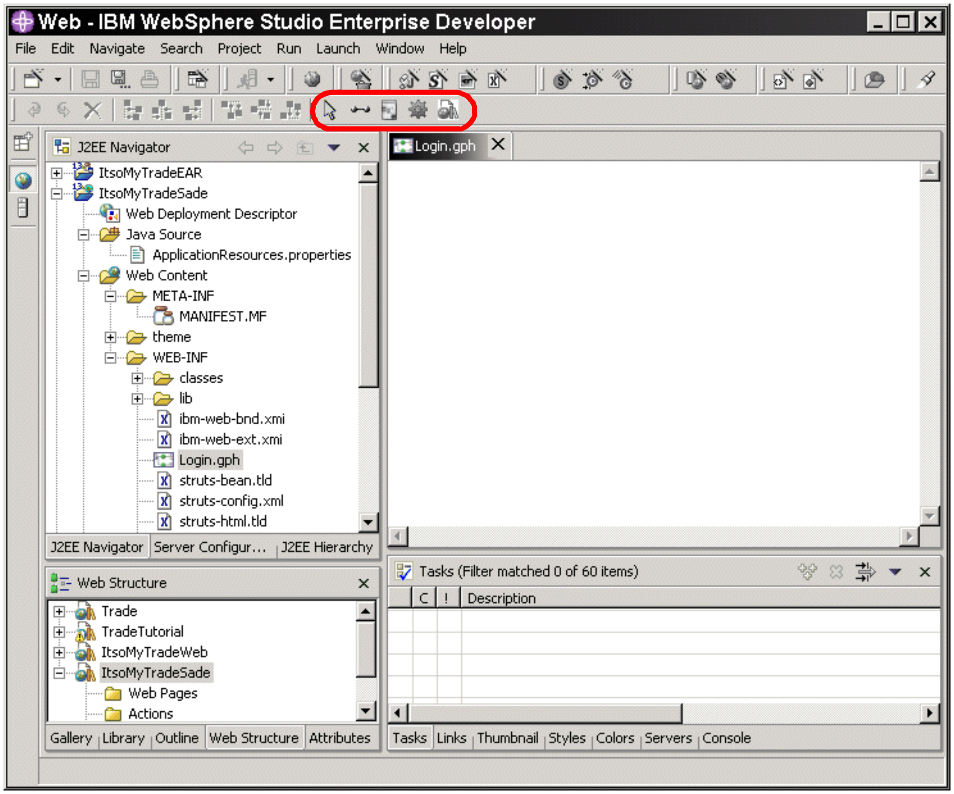
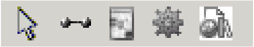

Web - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Run Launch Window Help
J2EE Navigator
ItsoMyTradeEAR
ItsoMyTradeSade
Web Deployment Descriptor
Java Source
ApplicationResources.properties
Web Content
META-INF
MANIFEST.MF
theme
WEB-INF
classes
lib
ibm-web-bnd.xmi
ibm-web-ext.xmi
Login.gph
struts-bean.tld
struts-config.xml
struts-html.tld
J2EE Navigator | Server Configur... | J2EE Hierarchy
Login.gph
Web Structure
Trade
TradeTutorial
ItsoMyTradeWeb
ItsoMyTradeSade
Web Pages
Actions
Gallery | Library | Outline | Web Structure | Attributes
Tasks (Filter matched 0 of 60 items)
C ! Description
Tasks | Links | Thumbnail | Styles | Colors | Servers | Console

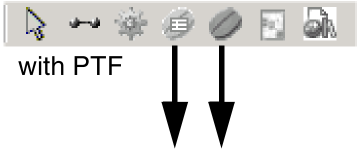
Figure 7-5 Initial Struts application dialog editor

- The editor pane is empty for now.
- A number of icons are available in the toolbar to create and connect objects in the Struts application diagram editor:



early availability

- – Select an Object
- – Connect Two Nodes
- – New Web Page Node
- – New Action Mapping Node
- – New Web Application Node



with PTF

- – New Form Bean Node
- – New Java Bean Node

Design the Struts application using the diagram editor

Initially, the diagram editor is used during the design phase of a project to lay out important aspects of the desired application. With this editor, users can quickly design the flow of the Web application, organizing pages and actions based on the application requirements, setting aside some of the issues associated with implementation.

In this section, we design the flow of the login sample application as described in the chapter introduction.

Creating the Web page objects

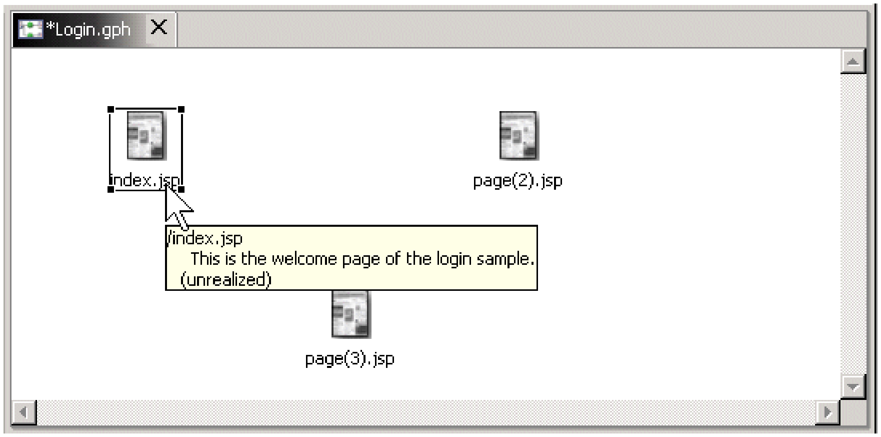
.We place three Web pages in the editor for the welcome page, home page, and error page:

- Click the *New Web Page Node* icon in the tool bar.
- Without holding down a mouse button, move the cursor to the diagram area.
- Click where you want the page to be.
- Overtype the default name `/page.jsp` with the name of your Web page, or change the name later using *Change Path* in the context menu.

These icons can be moved by dragging the icon. We put the welcome page at the top left of the editor, the home page at the top right, and the error page at the bottom.

- Right-click the icon for the welcome page and select *Change Path* in the context menu.
- Enter `/index.jsp` for the welcome page context path and press *Enter*.
- Right-click the icon and select *Change Description* to enter a description. Enter any text to describe the page, for example, *This is the welcome page of the login sample*.

Figure 7-6 shows the initial layout of the Web pages with the hover help on the welcome page. The name and description are also visible in the Properties view.



/index.jsp
This is the welcome page of the login sample.
(unrealized)

Figure 7-6 Web pages layout with hover help

Notice that all the icons are grey, indicating that they have not been realized, that is the JSPs have not been implemented. When an object is implemented, the grey icons turn into colored icons.

Complete the initial diagram by setting the path names and description for the `/home.jsp` (right) and the `/error.jsp` (bottom).

Creating an action object

For the verification for the user ID and password, we place an action mapping into the diagram editor:

- Click the *New Action Mapping Node* icon in the toolbar.
- Move the cursor to the diagram area and drop an action mapping object into the center area.
- Select the action mapping object and *Edit Path i*n the context menu. Enter `/login` for the context path. Optionally add a description.

Figure 7-7 shows the layout of the Web pages and the login action.

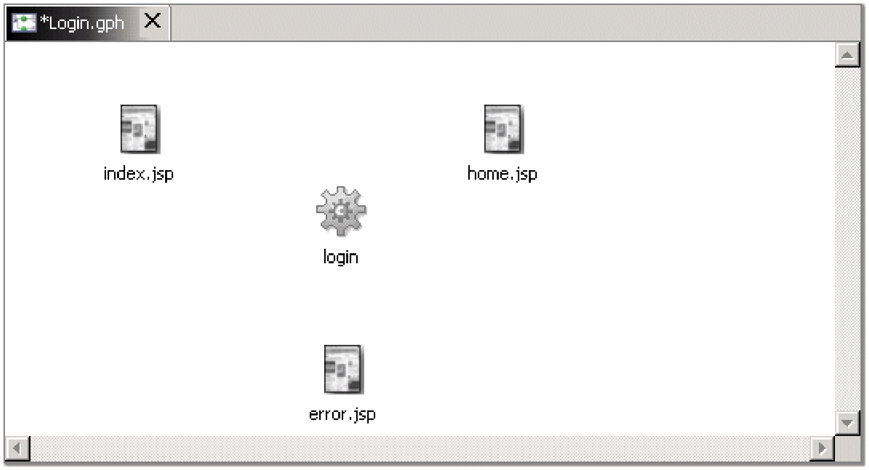


Figure 7-7 Web pages and action layout

Creating connections

We draw these connections between the objects

- From the welcome page to the action mapping
- From the action mapping to the home page
- From the action mapping to the error page
- From the error page to the welcome page

To create a connection:

- Click the C*onnect Two Nodes* icon in the toolbar.
- Click the source object in the editor and then click the destination object.

Alternatively, click an object and select *Connection* from the context menu, and then click the destination object.

You can enter the forward name on the connections from the action mapping icon by overtyping the `<new>` that is displayed. Alternatively:

- Right-click the connection between the login action mapping icon and `home.jsp` icon and select *Edit the forward name* in the context menu. Enter `valid` for the forward name.
- Repeat this for the connection to the error page and enter `invalid` for the forward name.

Figure 7-8 shows the diagram with the connections.

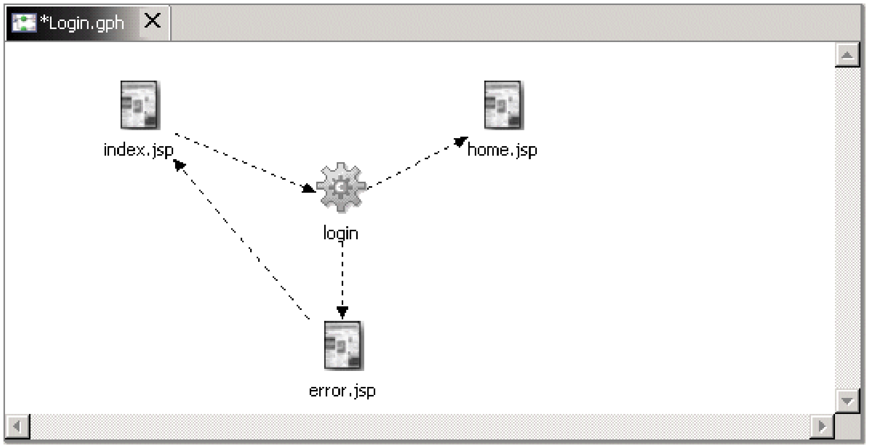


Figure 7-8 Login sample completed at the design phase

Save the current state of the diagram.

Implement the Struts application

The diagram editor can also be used to access the underlying resource files that compose the application. Here is the procedure for implementing the login sample:

- Implement Web pages
- Implement the form bean class
- Implement the action class
- Implement the action mapping

Implement Web pages

We are going to implement three Web pages:

`index.jsp` The welcome page defined in the deployment descriptor (`web.xml`).

`home.jsp` The home page to proceed to when the verification of user ID and password is successful.

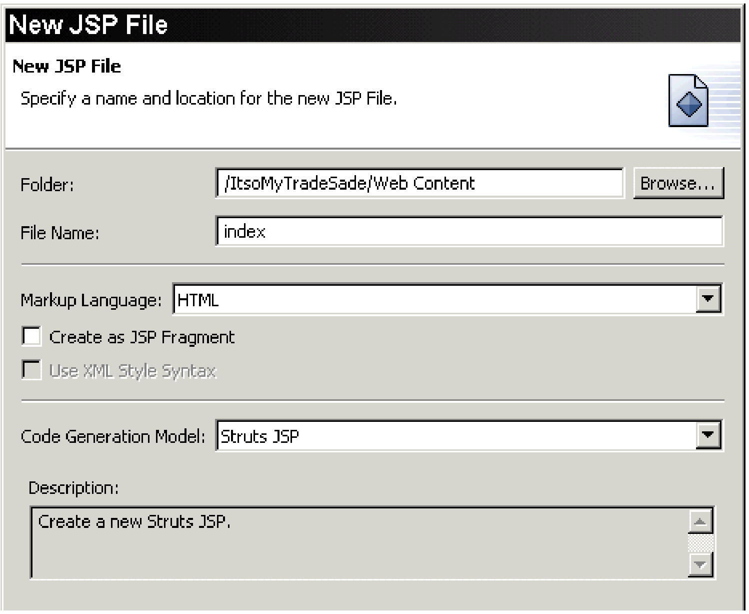
`error.jsp` The error page that shows the reason of the verification failure. This page contains a link to the `index.jsp` to return to the welcome page.

We will use the Struts custom tags to lay out the Web pages. All the text constants must be defined in the `ApplicationResources` file, as defined in Figure 7-3 on page 146.

Welcome page

To implement the welcome Web page (`index.jsp`):

- Open the Struts application diagram (`Login.gph`).
- Double-click the `index.jsp` page icon in the diagram to open the wizard for a new JSP.
- The file name is prefilled with `index`.
- You can go through all the panels or just click *Finish* to create the page.



New JSP File

New JSP File

Specify a name and location for the new JSP File.

Folder: /ItsoMyTradeSade/Web Content

File Name: index

Markup Language: HTML

☐ Create as JSP Fragment

☐ Use XML Style Syntax

Code Generation Model: Struts JSP

Description:

Create a new Struts JSP.

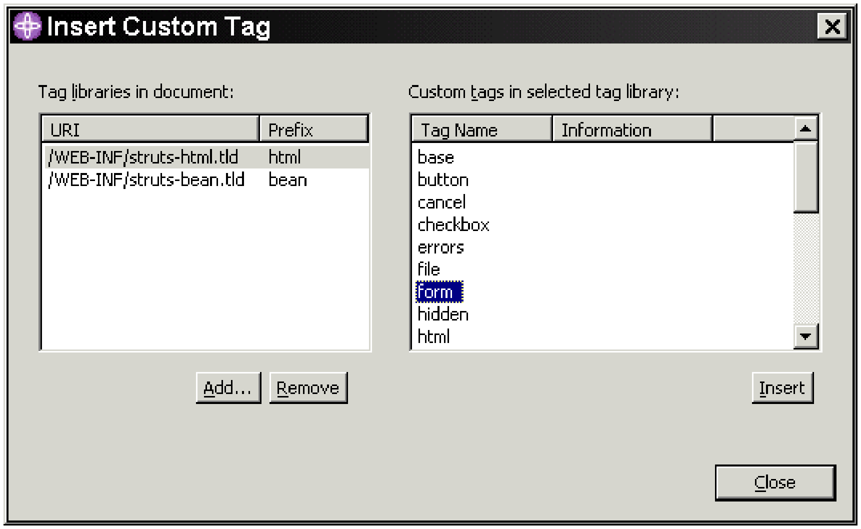
Figure 7-9 Creating index.jsp

The `index.jsp` file is created and the page designer opens to edit the content.

Tailoring the index.jsp with custom tags

Note that after installing the PTF, a form has already been added.

- Delete the text *Place index.jsp's content here*.
- Add a Struts form by selecting *JSP -> Insert Custom*. Select *html* and *form* in the Insert Custom Tag dialog and click *Insert* and *Close* (Figure 7-10).



Tag libraries in document:

URI	Prefix
/WEB-INF/struts-html.tld	html
/WEB-INF/struts-bean.tld	bean

Custom tags in selected tag library:

Tag Name	Information
base	
button	
cancel	
checkbox	
errors	
file	
form	
hidden	
html	

Figure 7-10 Inserting JSP custom tags

- Select the form tag and in the Properties view enter the value `/login` for the *action* property.
- Inside the form, add two message beans (*JSP -> Insert Custom -> bean -> message*.

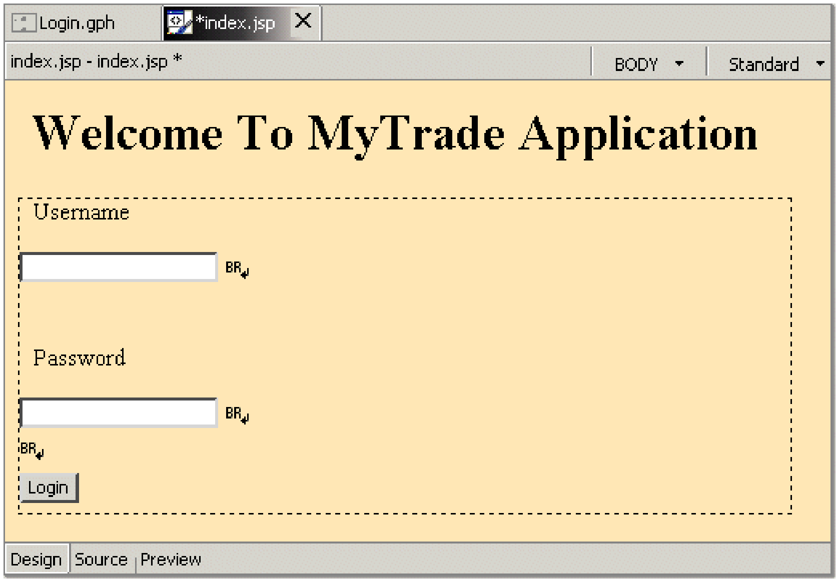
  Switch to the Source tab (or stay in the Design tab). Select each message tag and in the Properties view enter `global.field.username` and `global.field.password` for the *key* property.
- After the username message, add a custom tag *html -> text*. Select the text and in the Properties view enter `username` for the *property* property.
- After the password message, add a custom tag *html -> password*. Select the password tag and in the Properties view enter `password` for the *property* property.
- Below user ID and password, add a custom tag *html -> submit*. Inside this tag, add a message bean and set the *key* property to `welcome.button.login`.
- Above the form, add a heading (select *Insert -> Paragraph -> Heading 1*) and inside the heading add a custom tag for the heading text (*bean -> message*) with `index.title` as key.
- Add some line breaks for nice formatting.

Switch to the Source tab and check that the `<body>` content is similar to Figure 7-11.

```
<BODY>
<H1><bean:message key="index.title" /></H1>
<html:form action="/login">
   <bean:message key="global.field.username" />
   <html:text property="username"></html:text>
   <br>
   <bean:message key="global.field.password" />
   <html:password property="password"></html:password>
   <br><br>
   <html:submit>
      <bean:message key="welcome.button.login" />
   </html:submit>
</html:form>
</BODY>
```

Figure 7-11 Login JSP HTML code

Figure 7-12 shows the index.jsp in the Design tab of the page designer.



Welcome To MyTrade Application

Username

Password

Figure 7-12 Login JSP design view

- Save and exit the page designer.

In the Tasks view, the message *Action /login does not exist* is shown. We will solve this problem later by defining an action mapping.

Home page

Next, we create the `home.jsp`. Follow the sequence of the welcome page to create the home page:

- In the JSP wizard, make sure the file name is set to *home*.
- In the page designer, delete the text *Place `home.jsp`'s content here* and add these lines as an initial home page layout:

```
<h1><bean:message key="index.title"/></h1>
<h2>Home Page</h2>                                  <=== should use bean:message
```

- Save and exit the page designer.

Error page

Finally, we create `error.jsp` in the same way:

- In the JSP wizard, make sure the file name is set to *error*.
- In the page designer, delete the text *Place `error.jsp`'s content here* and add these tags to create an HTML link to the welcome page:

```
<h1><bean:message key="index.title"/></h1>
<h2>Error Page</h2>                                 <=== should use bean:message
<p>
<html:errors/>
<p>
<a href="index.jsp"><b>Welcome Page</b></a>
```

- Save and exit the page designer.

Switch to the diagram editor. We can see the pages realized with colored icons on Figure 7-13.

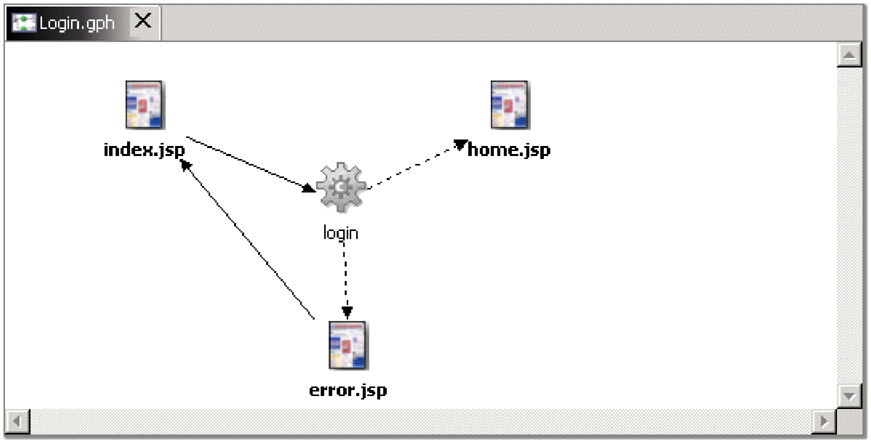


Figure 7-13 Diagram editor with JSPs realized

Implement the form bean class

WIth the early availability release, we have to create the form bean class by hand. With a PTF applied, we can use the diagram editor.

Creating the form bean by hand

We create an action form to handle the input of the `index.jsp`. This process is identical to "Creating the LoginForm class" on page 126:

- Create a package named `struts` in the `Java Source` folder.
- Create an action form class (*New* -> *Web* -> *Struts* -> *ActionForm Class*). Use the `struts` package and enter `LoginForm` as the name. Click *Next*.
- For accessors, expand the `ItsoMyTradeSade` project and select *username* and *password* of the `/login` action in the `index.jsp`.
- Skip the other pages and click *Finish*.
- The `LoginForm` opens in the editor. The code is complete and you can close the editor.
- Open the `struts-config.xml` file and you can see the `loginForm` on the Form Beans tab. Close the editor.

Creating the form bean using the diagram editor

Select the form bean icon and drop a form bean into the diagram. Enter `loginForm` as name. Then use the connector icon to connect the form bean to the action (Figure 7-14).

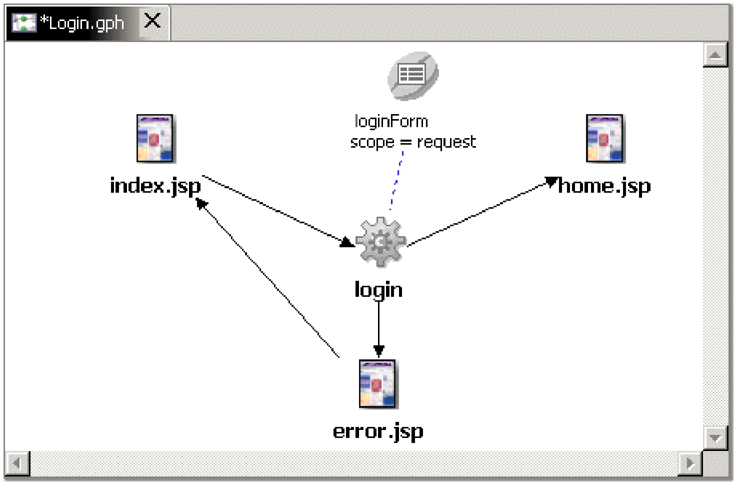


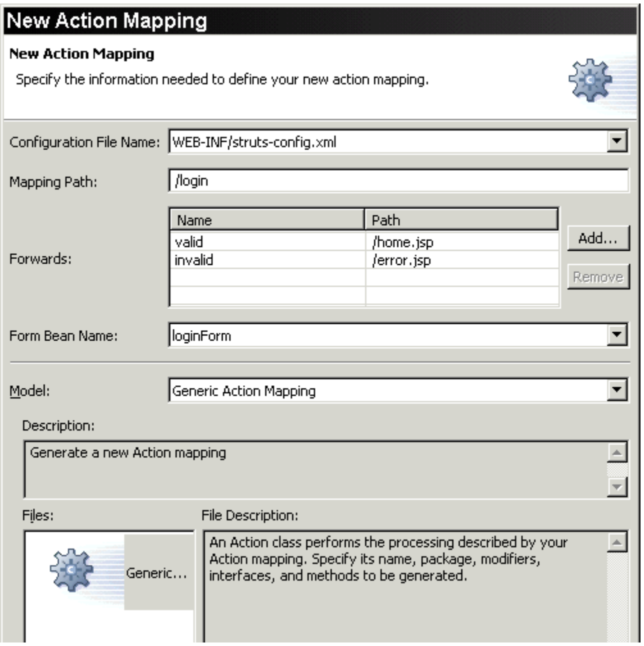
Figure 7-14 Diagram editor with new form bean

Double-click the form bean and the form bean wizard opens. Complete the information in the same way as for “Creating the form bean by hand” on page 156.

Implement the action mapping and action class

We implement the action mapping for the `LoginAction` from the diagram editor:

- Double-click the login action mapping icon in the Struts diagram editor.
- The New Action Mapping dialog opens on the default `struts-config.xml` file (Figure 7-15):
  - The Mapping Path is set to `/login`.
  - The Forwards `valid` and `invalid` are shown and they point to `/home.jsp` and `/error.jsp` respectively.
  - Select the `loginForm` from the Form Bean name pull-down.
  - Select *Generic Action Mapping* for the Model pull-down.
  - Click *Next*.



New Action Mapping

New Action Mapping

Specify the information needed to define your new action mapping.

Configuration File Name: WEB-INF/struts-config.xml

Mapping Path: /login

Forwards:

Name	Path
valid	/home.jsp
invalid	/error.jsp

Form Bean Name: loginForm

Model: Generic Action Mapping

Description:

Generate a new Action mapping

Files:

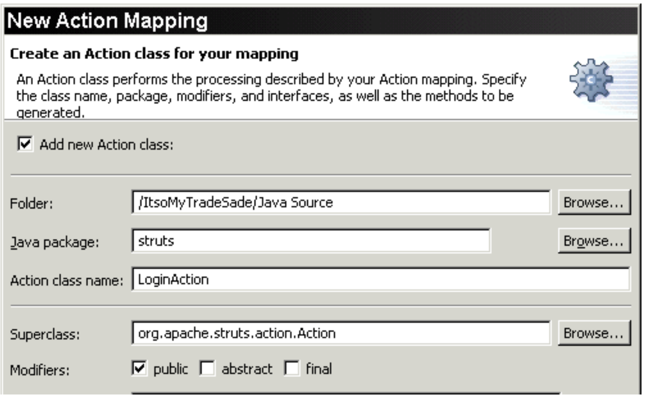
File Description:

An Action class performs the processing described by your Action mapping. Specify its name, package, modifiers, interfaces, and methods to be generated.

Figure 7-15 Creating a new action mapping

- In the Create an Action class dialog (Figure 7-16):
  - Select *Add new Action class*.

- Select the `struts` package and enter `LoginAction` as class name.
- Leave all other fields as defaults.
- Click *Finish*.



New Action Mapping

Create an Action class for your mapping

An Action class performs the processing described by your Action mapping. Specify the class name, package, modifiers, and interfaces, as well as the methods to be generated.

☑ Add new Action class:

Folder: /ItsoMyTradeSade/Java Source

Java package: struts

Action class name: LoginAction

Superclass: org.apache.struts.action.Action

Modifiers: ☑ public □ abstract □ final

Figure 7-16 Creating the action class for the action mapping

Complete the perform method in the action class

The Java editor opens and you can see the generated code in the editor.

Add the validation logic to the `perform` method (Figure 7-17).

```
try {
   // do something here
   if (!loginForm.getUsername().equals("userid"))
      errors.add("login", new ActionError("error.invalidUsername"));
   else if (!loginForm.getPassword().equals("password"))
      errors.add("login", new ActionError("error.invalidPassword"));
} catch (Exception e) {
   // Report the error using the appropriate name and ID.
   //errors.add("name", new org.apache.struts.action.ActionError("id"));
   errors.add("login", new ActionError("error.login.exception"));
}
```

Figure 7-17 Action class user ID and password validation

The valid user ID and password are *userid* and *password*. Note that the errors point to the text constants defined in the application resources file.

Note: The generated action class does not use the forward names we specified in the dialog—hopefully this will be fixed.

Although we entered `valid` and `invalid` as forward names, the generated code uses *success* and *failure* as default forward names. Change the names to *valid* and *invalid* as shown in Figure 7-18.

Save the `LoginAction` class and close the editor.

```
if (!errors.empty()) {
   saveErrors(request, errors);
   // Forward control to the appropriate 'failure' URI (change name ...)
   forward = mapping.findForward("invalid");
} else {
   // Forward control to the appropriate 'success' URI (change name ...)
   forward = mapping.findForward("valid");
}
```

Figure 7-18 Success and failure in the action class

Complete the Struts configuration file

Open the `struts-config.xml` file in the editor. Select the `/login` action on the Actions tab:

- Enter `/index.jsp` in the Input field.
- Check that the Forwards are set to `valid` and `invalid`.
- Check that the `loginForm` in the Form Bean Name is selected.
- Select the *Form Beans* tab at the bottom of the editor. The `LoginForm` was added already with the name `loginForm`.
- Save the changes.

You can confirm the updates by switching to the Source tab where you can view the XML source of the configuration file as shown in Figure 7-19.

```
<!-- Form Beans -->
<form-beans>
   <form-bean name="loginForm" type="struts.LoginForm">
   </form-bean>
</form-beans>

<!-- Action Mappings -->
<action-mappings>
   <action name="loginForm"> path="/login" type="struts.LoginAction"
            input="/index.jsp"
      <forward name="valid" path="/home.jsp">
      </forward>
      <forward name="invalid" path="/error.jsp">
      </forward>
   </action>
</action-mappings>
```

Figure 7-19 Source of Struts configuration file (extract)

Complete Struts application diagram

You can see the implemented login sample in Figure 7-20. The icons have all turned into colored icons and the connections are represented using solid lines.

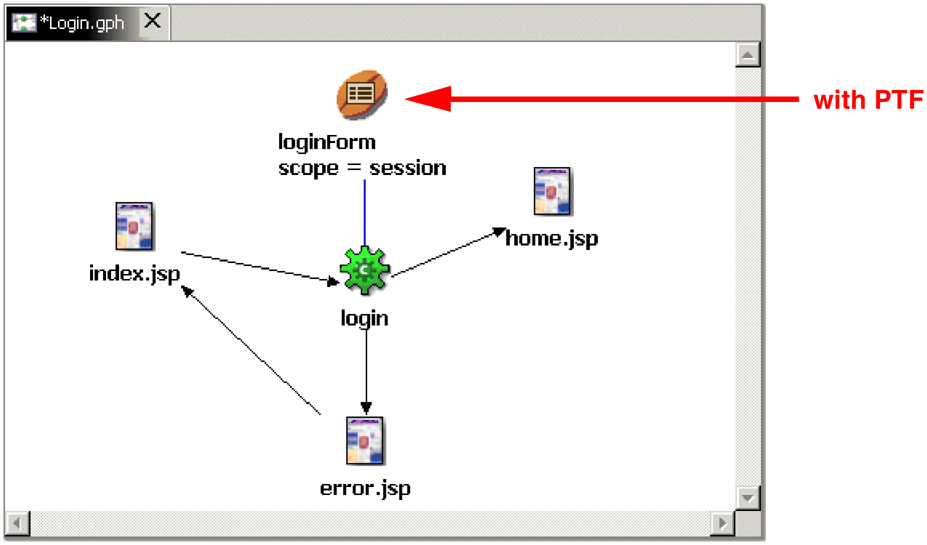


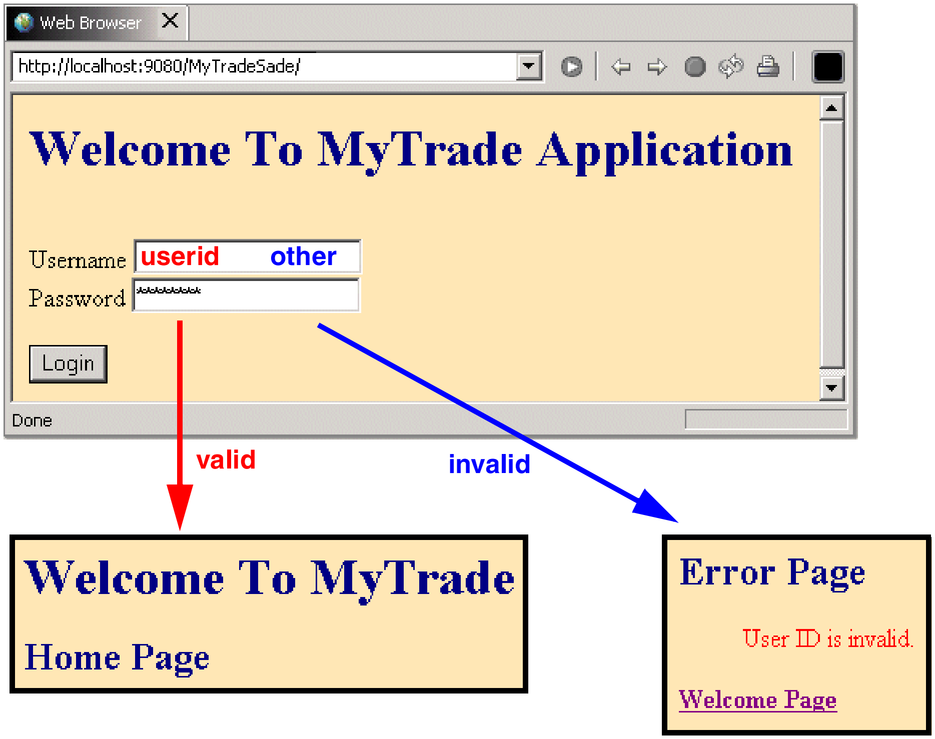
Figure 7-20 Login sample implemented

Testing the Struts application

To test the Struts application we can use the existing `StrutsServer` because the `ItsoMyTradeSade` Web application is part of the `ItsoMyTradeEAR` enterprise application that is attached to the server:

- Start the `StrutsServer`.
- Select the `ItsoMyTradeSade` project and *Run on Server* from the context menu.
- You can confirm the run-time behavior of the login sample by trying out different user ID and password combinations. The only valid values are *userid* and *password*.
- Stop the `StrutsServer`.

Figure 7-21 shows the Web browser for a sample run of the application.



http://localhost:9080/MyTradeSade/

Welcome To MyTrade Application

Username: userid / other

Password: ********

Welcome To MyTrade

Home Page

Error Page

User ID is invalid.

Welcome Page

Figure 7-21 Struts application run

Analyze a Struts application

Using the diagram editor, a developer can draw a diagram of the application flow from an existing Struts application.

In this section, the flow analysis capabilities of the diagram editor are described. We use the trade sample to show a sample usage.

Install the trade sample application

The trade sample is bundled as a sample application with the Application Developer.

If you installed the trade sample application already, then skip this section. Otherwise, follow the instructions in “Installing the trade sample application” on page 56:

- Create the application using the New wizard.
- This process installs the `TradeSample` enterprise application project, the `Trade` Web project, and the `TradeEJBs` EJB project, and creates a DB2 database named `TRADEDB`. Creating the database is not required to complete this chapter; however, the database is required for the EGL chapters.

Drawing the application flow

The diagram editor has a capability of drawing the application flow of an existing Struts application.

First we have to create a new Struts application diagram:

- Select the *Trade -> Web Content -> WEB-INF* folder in the Web view and *New -> Other* from the context menu.
- Select *Web -> Struts* in the left pane and *Web Diagram* in the right pane. Click *Next* to proceed. Enter *Trade* as the file name and click *Finish*.

The diagram editor opens. We initialize the diagram with the `index.jsp` file:

- Drag the `index.jsp` and drop it in the top left of the diagram editor.
- Select the `index.jsp` icon in the diagram editor and click *Draw -> Draw All* in the context menu.
- The flow of the trade application is automatically rendered in the editor as shown in Figure 7-22.

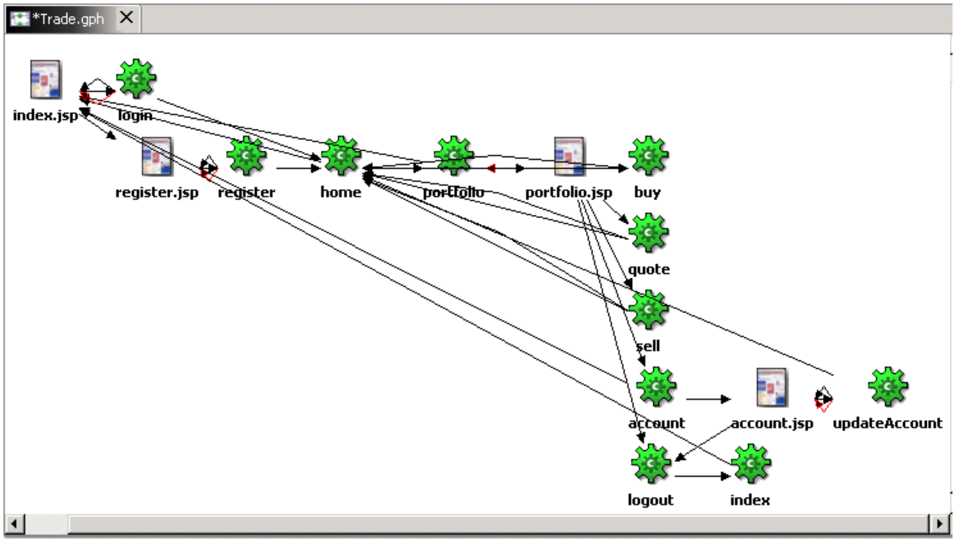


Figure 7-22 Application flow rendered by the diagram editor

With the PTF applied, the form beans are also added to the diagram (Figure 7-23).

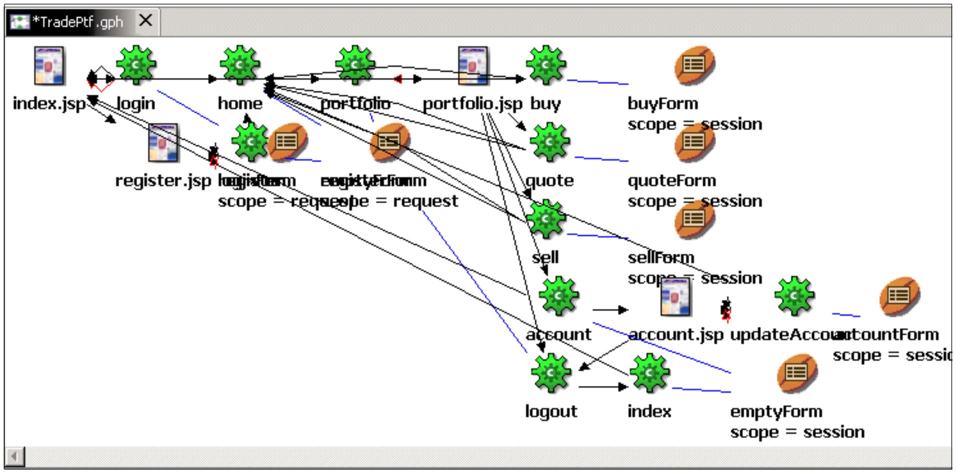


Figure 7-23 Application flow rendered by the diagram editor (with PTF)

The icons and connections can be repositioned by dragging the mouse to create a better visual appearance. The result is shown in Figure 7-24.

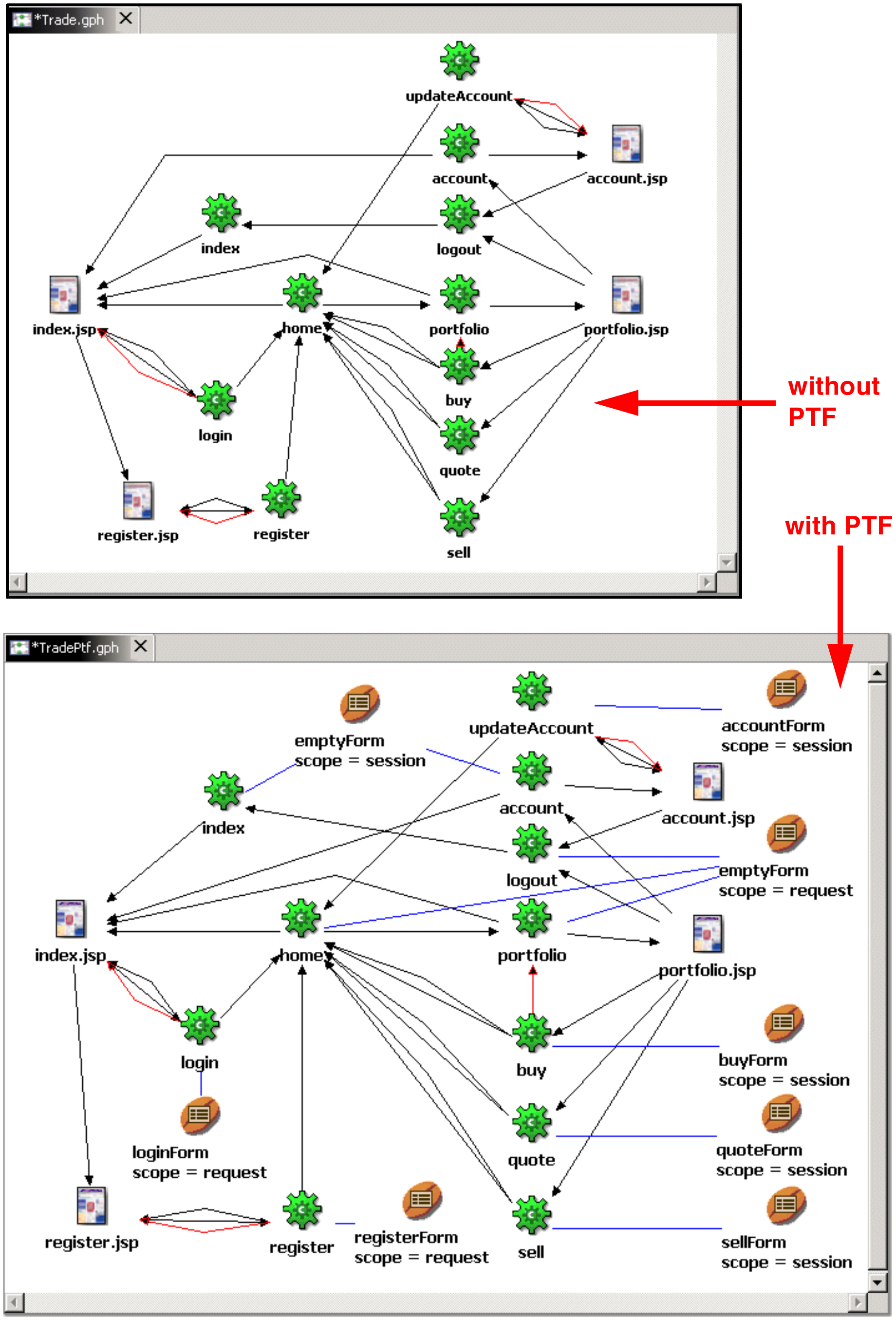


Figure 7-24 Tailored application flow, without and with PTF

To understand the diagram, you can:

- Select icons and connections and see the definition in the Properties view.
- Double-click Web pages to open the page designer.
- Double-click action icons to open that action in the Struts configuration editor.
- Double-click form beans to open that form bean in the Struts configuration editor.
- Double-click connections to open the Struts configuration editor for that success or failure link.
- Hold the mouse over an icon or connection to see the hover help with the name and description.
- Red arrows indicate the input JSP.

Drawing the main path of the flow

The diagram shown in Figure 7-24 is a busy diagram. When the application gets more complicated, it is difficult to grasp the overview of the application flow. In this section we take a different approach to draw parts of the diagram.

While the complete flow of the trade sample is composed of many paths including error handling, we are creating a diagram for the normal path in order to analyze the application flow:

- We create a new diagram named `TradeMain`.
- Place the welcome page of the trade sample on the diagram editor. (Drag the `index.jsp` into the editor.)
- We will inspect the application flow using the *Draw All From* function step by step.

Note: We only show this process without the PTF, that is the form beans are not shown.

index.jsp

Right-click the `index.jsp` icon and select *Draw -> Draw All From* in the context menu.

The action mapping icon named `Login` and the page icon named `register.jsp` are shown in the diagram editor (Figure 7-25).

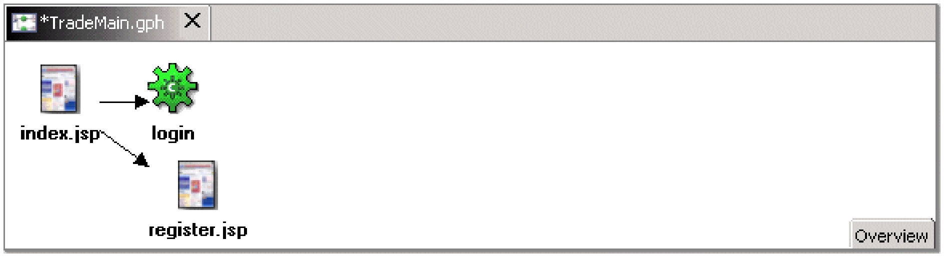


Figure 7-25 Drawing the main path (1)

register.jsp

The `register.jsp` is used for user registration. We remove the icon from the diagram.

- Select the `register.jsp` and *Delete* from the context menu.
- You are prompted whether you want to *Delete underlying resource?* Click *No*. Select the No action in all future delete requests.

Login action mapping

We continue the analysis with the login icon:

- Select the `login` icon and *Draw -> Draw All From* in the context menu (Figure 7-26).

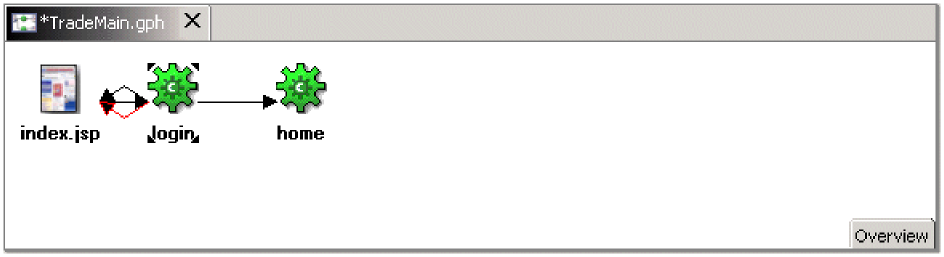


Figure 7-26 Drawing the main path (2)

You can see the *home* action mapping icon on the diagram. Two links from the `login` icon to the `index.jsp` are shown.

When you move the cursor on the red link from `login` to `index.jsp`, the hover help displays `<input>--><index.jsp>`, meaning the error handling path for the validation in the `FormBean` class. We remove this link from the dialog. Select the red link and *Delete* from the context menu (do not delete the underlying resource).

The hover help shows `failure-->/index.jsp` on the black link from `login` to `index.jsp`. The `LoginAction` class returns *failure* if the authentication fails. We also delete this link from the dialog.

Home action mapping

We expand the home action mapping with *Draw All From* (Figure 7-27).

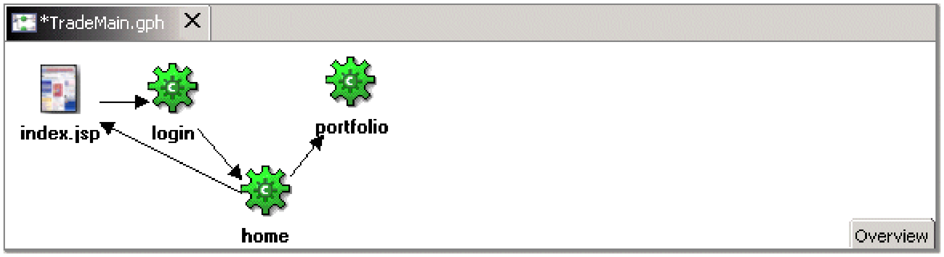


Figure 7-27 Drawing the main path (3)

An action mapping of *portfolio* is added to the diagram. You can also see the link from `home` to `index.jsp`. This is the *failure* forward of the home action. We delete this link from the diagram.

Portfolio action mapping

Next, we inspect the portfolio action mapping with *Draw All From* (Figure 7-28).

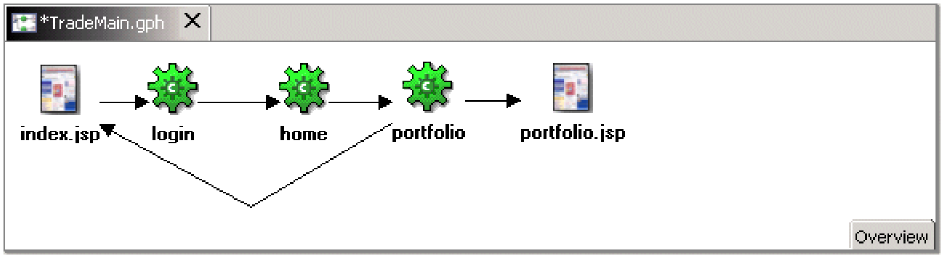


Figure 7-28 Drawing the main path (4)

We delete the *failure* link from `portfolio` to `index.jsp`.

portfolio.jsp

We contine the analysis on the `portfolio.jsp` with *Draw All From* (Figure 7-29).

On the diagram the action mappings `account`, `buy`, `logout`, `quote` and `sell` are added.

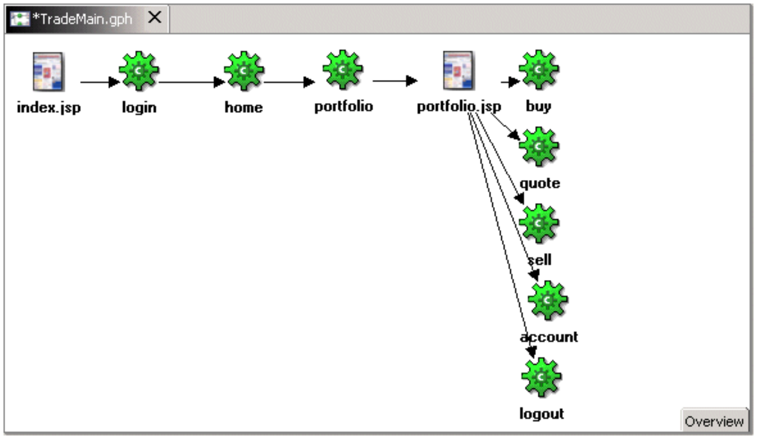


Figure 7-29 Drawing the main path (5)

Iteration for all action mappings

For each action mapping, we iterate the *Draw All From* operation. For each added object, we iterate again. Finally we delete all the failure connections. The result, after some rearranging, is shown in Figure 7-30.

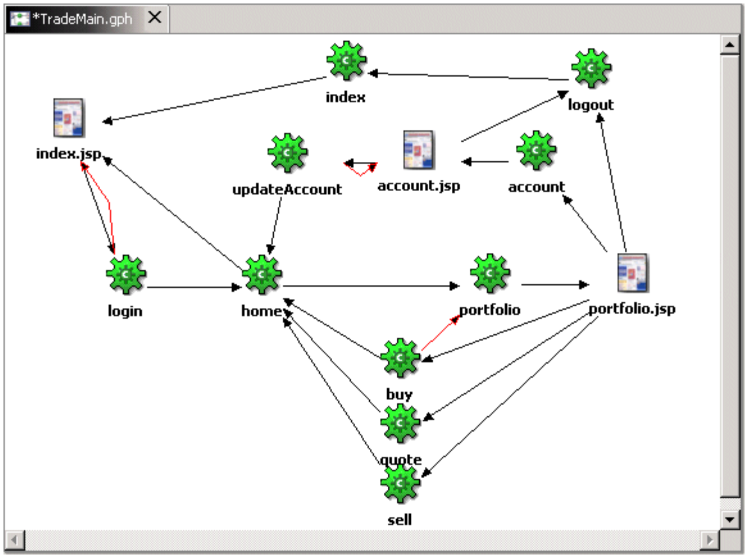


Figure 7-30 Drawing the main path (6)

Using this technique, users can prepare diagrams for each sub-scenario of the application flow, which is very effective for more complicated application flows.

Part 3

Enterprise generation language

Enterprise generation language (EGL) is a new language for creating complex business applications running on a variety of platforms.

EGL is a high-level language from which Java and COBOL code is generated.

In Part 3 we describe how EGL can be used to generate Java programs running on Windows and COBOL programs running on z/OS. We also describe how such programs can be accessed as Struts action programs.

8

Implementing EGL actions

In this chapter, the following topics are described:

- Enterprise generation language (EGL) fundamentals
- Building business applications using EGL
- Generating Java and COBOL from EGL
- Incorporating EGL-generated code into Struts action classes

We use the `ItsoMyTradeWeb` project to implement EGL actions.

Accessing EGL programs

Enterprise generation language (EGL) will replace VisualAge Generator as the tool for creating complex business applications using a high-level programming specification, in the next version of the product, when all features existing today in VisualAge Generator will be implemented.

The current release of EGL generates Java source code (for Windows or z/OS UNIX System Services) and COBOL (for CICS/MVS). To access the generated EGL code, the generation process can also generate Java wrappers that can be included in Java programs that have to access EGL-generated code such as Struts action classes. The Java wrappers can be wrappers for both Java and COBOL code generated by the EGL build process.

When the EGL generator generates COBOL code, it can also generate a Java wrapper class that uses a J2EE Connector resource adapter (J2C Connectors) to access the COBOL code through a CICS transaction gateway.

Figure 8-1 shows the process of how business logic defined using EGL is generated and applied at run time.

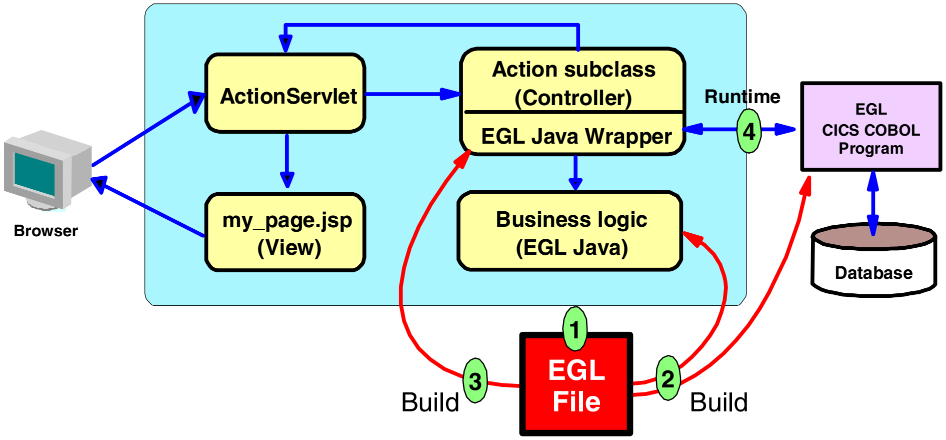


Figure 8-1 Accessing EGL-generated components

1. Enterprise Developer is used to define the business logic and the database records in EGL.
2. The developer can then generate source code in either Java or COBOL that implements this business logic.
3. The developer can also choose to create a Java wrapper for the EGL-generated source code. In this example, the Java wrapper class is used by a Struts action class to access EGL-generated code.

4. If COBOL source is generated, then a Java wrapper is generated to access a J2EE connector resource adapter (J2C) for the CICS/MVS system.

Enterprise Developer also includes a test run-time environment and debugger.

In the remainder of this chapter, we will discuss EGL in more detail. We will show how to create an EGL program and how to generate Java and COBOL source code from the EGL program. We will also discuss how an EGL-generated component can be tested and debugged and how EGL-generated programs can be accessed through Struts.

Creating and generating EGL programs

In this section we discuss the architecture of EGL and how it is used to create programs and generate source code.

In general, the development process for creating programs with EGL is as follows:

- Iteratively develop and test
  - Create EGL parts and write EGL scripts
  - Validate the EGL to verify a correct specification
  - Generate source code in Java or COBOL
  - Debug with built-in debugger
- Deploy
  - Generate 3GL source code from EGL specifications (Java or COBOL)
  - Prepare run-time executable by compiling the generated 3GL source code
- Run
  - In z/OS UNIX System Services, Windows 2000 or Windows NT (this executes in either tier 2 or 3 of the typical application topology)
  - CICS for MVS (this executes in tier 3 of the typical application topology)

EGL editing in Enterprise Developer

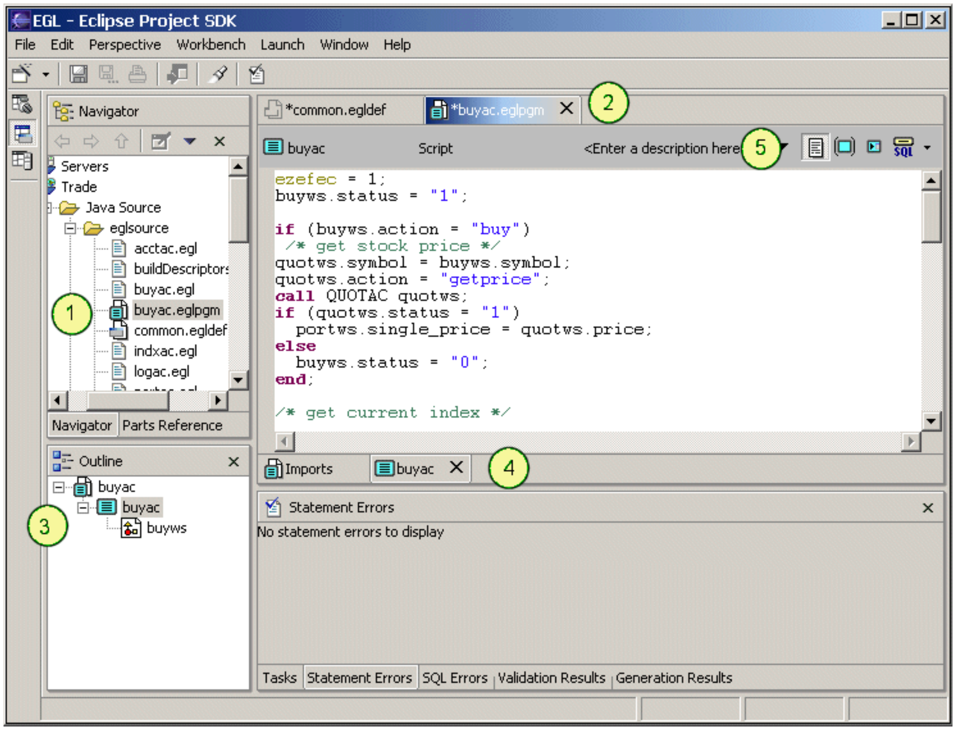
When you open an EGL file in Enterprise Developer (a file with extension of `eglpgm`, `egldef` or `eglbld`), the EGL editor opens, if it is the default.

EGL editors

The Enterprise Developer provides two EGL editors—the EGL Part Editor and the EGL Source Editor.

The default editor for EGL files is set by selecting *Window -> Preferences*, *Workbench -> File Associations.* We suggest that you set the default for `.egldef`, `.eglpgm`, and `.eglbld` to the EGL Part Editor.

An example of the EGL Part Editor is shown in Figure 8-2.



```
ezefec = 1;
buyws.status = "1";

if (buyws.action = "buy")
 /* get stock price */
quotws.symbol = buyws.symbol;
quotws.action = "getprice";
call QUOTAC quotws;
if (quotws.status = "1")
  portws.single_price = quotws.price;
else
  buyws.status = "0";
end;

/* get current index */
```

No statement errors to display

Figure 8-2 EGL Part Editor

These are the various views of the EGL perspective that are involved when editing EGL parts:

1. The Navigator view shows the project resources. EGL files are designated by three icons:

   This icon denotes EGL definition files that contain common definitions of parts that can be referred to by other parts in other EGL files.

   This icon denotes EGL program files that contain EGL logic parts and any other parts particular to the program (data parts).

   This icon denotes EGL build files that contain EGL control parts to define the build process.

The default behavior of opening one of these resources is to open the EGL editor and EGL Outline view. The first part defined in the file is opened in the EGL editor.

2. The top tab of the editor pane shows all the currently open EGL files.
3. The Outline view shows the parts contained in the EGL file that is currently in focus. You can open a part by selecting it and pressing the *Enter* key, or by double-clicking the part.
4. The bottom tab of the EGL Part Editor shows the parts of the EGL file that have been opened from the Outline view. The tab with the “X” icon is the currently active part. You can close any part by selecting the tab and clicking the “X” icon when it appears. Also, the imported files are shown when the icon is clicked.
5. Each part can have multiple pages and different types of parts have a set of different pages. For example, only logic parts have a script page. The list below shows all the icons that can appear in the various part editors and which page of the editor they open. The first sentence in the description is the hover help label that appears for that icon if the mouse is held over the icon.

   Show Script—The script editor is used to edit program and function code.

   Show Signature—Clicking this icon allows you to specify input parameters for a program.

   Show Variables—This action opens the variable declaration page for a program part.

   Show Signature— This action allows you to specify the input parameters and the return value for a function.

   Show Variables—This action opens the variable declaration page for a function part.

   SQL Show SQL Statement or Show SQL Default Select Conditions—This part of the editor is used to edit SQL statements in programs, functions, and SQL record parts.

   Show Structure— For a record part, this action shows the record structure.

   Show Organization Properties— For some record parts, this action shows the properties that can be specified for the part. An SQL record, for example, will have a table name specified here, while an MQ record will have the queue definitions specified here.

   SQL Show SQL Item Properties—This icon appears for an SQL record.

EGL files and parts

EGL is made up of special programming parts and a procedural scripting language. The intent is to allow developers to create complex business applications using a high-level procedural scripting language without having to know the details of the implementing technologies. The final implementation of the business logic is generated as Java or COBOL source code by Enterprise Developer from the EGL script and parts.

EGL parts

An EGL program is structured as *parts* that developers use to implement the different concepts within an EGL program. The overall EGL program uses a set of different types of parts to implement its business logic.

EGL parts can be categorized as one of three types:

Logic parts Logic parts are program or function parts that implement business logic. The file name extension for program parts is `eglpgm`. A program is the only top-level part allowed in an `eglpgm` file; functions and data parts can be nested under the program. Functions can also be defined in files with extension `egldef`.

The definition of program and function used here have the same meaning as in VisualAge Generator.

Data parts Data parts provide access to the state of the application or to data in persistent storage tables or working storage. You can place data parts into files with the extension `egldef`, or you can nest them under programs in an `eglpgm` file.

The definition of data parts used here has the same meaning as in VisualAge Generator (with the exception of the new structure part).

Control parts Control parts define how the source code is generated from the EGL parts and scripts. Control parts are build descriptors, linkage options, resource associations, link edit, and bind control. The file name extension for control parts is `eglbld`.

In VisualAge Generator, these parts were called generation options, linkage table, resource associations, bind control, and link edit.

Data parts

Data parts define data items and structures that are accessed by logic parts. There are three different types of data parts:

Data item A data item is a specification of the format and structure of data. Data items can be defined and used in logic parts to allocate memory to contain data of the type defined by the data part. A data item can be passed as a parameter to a logic part.

Structure A structure is a collection of memory areas called structure items. SQL item properties may also be included in a structure. A structure cannot be used to perform I/O operations.

Record A record is also a collection of memory areas that is internal to a program and has a certain organization type. Record parts can be used to manipulate data that is internal to a program and some organization types can be used to transfer data to and from persistent storage and memory.

Like a structure, an SQL record can also contain SQL items. However, an SQL record can also be used to perform I/O operations and can reference a structure as a type definition.

Records

A record data part is further specified by how it is used to access data:

Indexed An indexed record is used to declare a file that is accessed through a key value that represents the logical position of the record in persistent storage (for example, `VSAM KSDS`).

Relative A relative record is used to define a record of fixed length within a data set. The record is accessed by specifying its (numeric) sequential position in the data set (for example, `VSAM RRDS`).

Serial A serial record is associated with a file or data set. The file is read sequentially and a write adds a record to the end of the file.

MQ An MQ record is used to work with messages from WebSphere MQ.

SQL SQL records are used to read and write to relational tables through SQL statements.

Working storage Working storage is used to hold data for internal processing within a program.

Placement of EGL source files

Within a Web application, you can place EGL source folders in three locations:

- Under `Java Source`
- Under `Web Content`
- Directly under the project

Table 8-1 shows the pros and cons of each selection.

Table 8-1 Placement of EGL source folders

Placement	Pros	Cons
Java Source	EGL source are source files, so they belong into the source category. Export of the project into a WAR file includes the EGL source folders.	By being placed under `Java Source`, the EGL files are copied to `Web Content\WEB-INF\classes`. This creates duplicates that show up when selecting a build descriptor to generate Java or COBOL code.
Web Content	There will be no duplicates. Export of the project into a WAR file includes the EGL source folders.	EGL source files do not belong under `Web Content`; their purpose is very different.
In Web project	A separate EGL source folder makes most sense because EGL is not Java code and not Web content. There will be no duplicates.	Export of the project into a WAR file does **not** copy EGL source folders.

For the time being we will place EGL source folders under **Java Source**, although EGL is not Java code. **Note that we could use a completely separate simple project for the EGL code, and generate the Java code into the Web project.**

Creating EGL files and parts

EGL parts are usually created in the EGL perspective of Enterprise Developer. We will be creating record parts for the trade sample application in a new folder under the `Java Source` folder of the `ItsoMyTradeWeb` project.

> **Tip:** To remove all the warnings from the trade sample projects from the Tasks view, click the *Filter* icon and in the filter dialog select *On any resource in same project*.

Create a folder

We create a separate folder for EGL parts to distinguish them from Java:

- Open an EGL perspective (select *Window -> Perspective*).
- In the Navigator view, select the `Java Source` folder and *New -> Other -> Simple -> Folder* and click *Next*.
- In the New Folder dialog, make sure that `Java Source` is selected. Specify `eglsource` as the folder name and click *Finish.*

Create an EGL file

To create an EGL file in the `eglsource` folder:

- Select the `eglsource` folder and *New -> Other -> EGL -> EGL Definitions File* from the context menu.
- In the Create EGL Definitions File dialog, make sure that the `eglsource` folder is selected.
- In the File name field, enter *common* as the name for the EGL file and click *Finish.*

You should now have a `common.egldef` file in the `eglsource` folder and it should also be open in the EGL editor and visible in the Outline view (Figure 8-3).

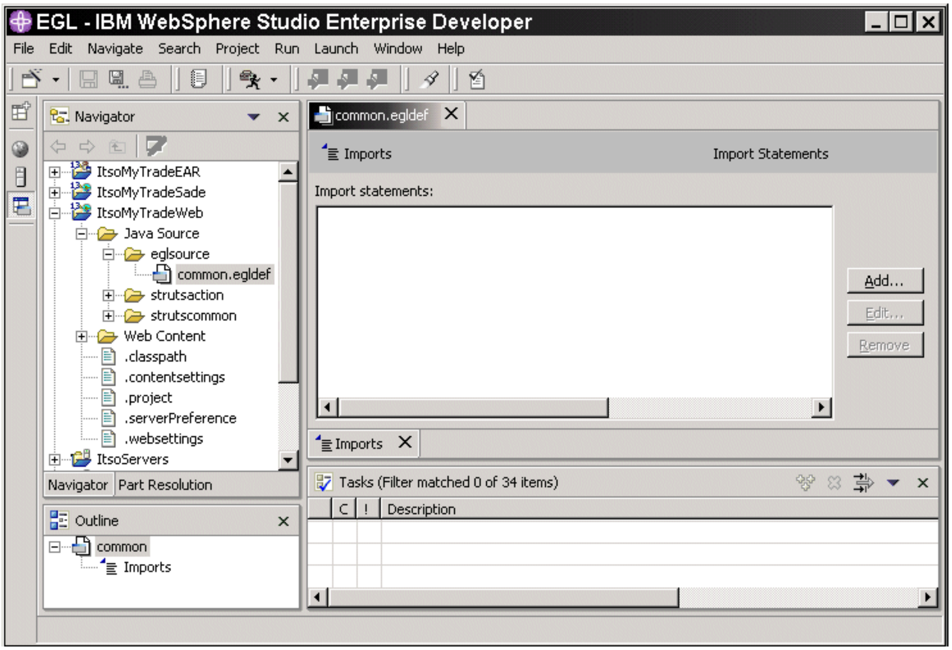


Figure 8-3 EGL definition file common.egldef

Important: There is an EGL Part Editor and an EGL Source Editor. You may have to close the file and reopen it in the EGL Part Editor to get this view. Select the file and *Open With -> EGL Part Editor.*

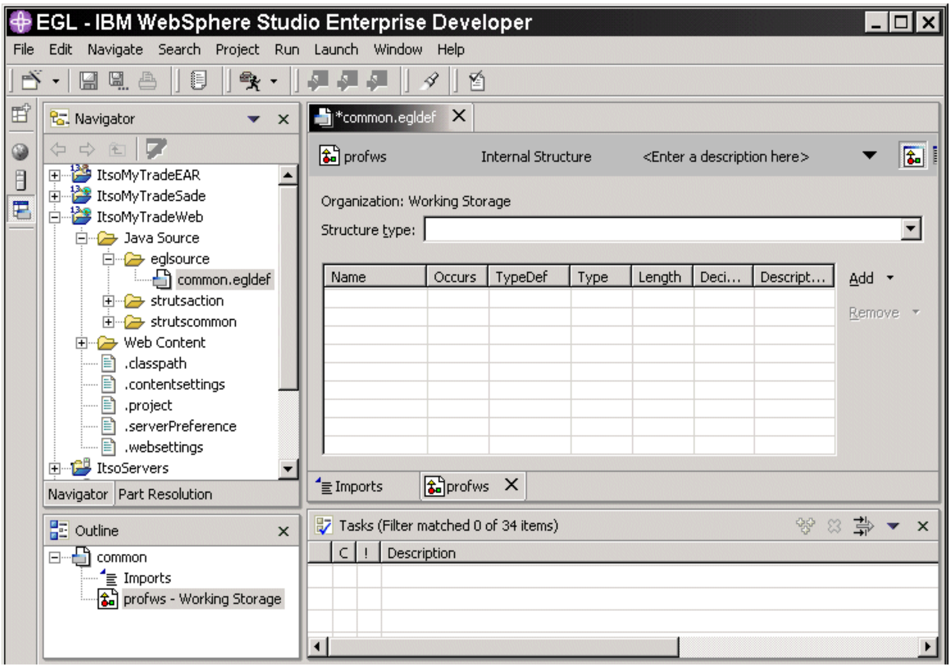
We can now start adding data parts to the EGL definitions file `common.egldef`. The parts in this file will be referenced by other EGL parts that we create later.

Create an EGL part

To add a working storage record:

- Select `common` in the Outline view and *Add Part...* from the context menu.
- For Select Type of EGL Part to Add, select *Record* and click *Next.*
- In the Name field, enter `profws`. We create the working storage record to hold profile information of the site user. Click *Next.*
- In the Select Record Organization page, select *Working Storage Record*.
- Click *Finish.*

You should now have an empty working storage part in the `common.egldef` EGL definitions file, and the record editor for `profws` should be open (Figure 8-4).



Organization: Working Storage

Structure type:

Name	Occurs	TypeDef	Type	Length	Deci...	Descript...

Figure 8-4 New working storage record part

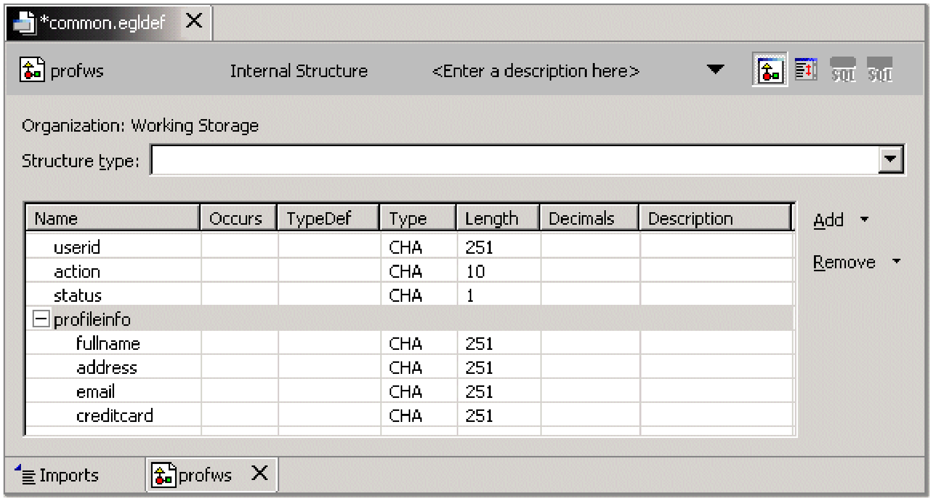
To build the structure of the record part, use the *Add* and *Remove* drop-downs to the right of the structure table.

- Select *Add -> Add After*. This adds the first item in the structure with a name of `NewName`.
- The new item should be selected. Select the Name cell to edit it and enter a name of `userid`.
- Select the Type cell to put it in edit mode. In the drop-down, select *CHA*. This specifies the item as being a character type. Press *Enter*.
- Select the Length cell to put it in edit mode. Enter `251`. This specifies the item as being 251 characters in length. Press *Enter*.

You can also add sub-structures to the record. Insert an item by clicking *Add -> Add Before* or *Add After*. Select the item and then add a child by clicking *Add -> Add Child*.

An existing item can also be made into a new sub-structure. Select the item you want to make part of a new sub-structure. Click *Add -> Add Parent*. This creates a parent part with the originally selected part as a child within it.

Follow the procedure above to enter the rest of the items for the `profws` record part as shown in Figure 8-5.



Organization: Working Storage

Structure type:

Name	Occurs	TypeDef	Type	Length	Decimals	Description
userid			CHA	251		
action			CHA	10		
status			CHA	1		
⊟ profileinfo						
fullname			CHA	251		
address			CHA	251		
email			CHA	251		
creditcard			CHA	251		

Figure 8-5 Definition of the profws part

Save the changes to `common.egldef`.

Create the working storage records and data items shown in Table 8-2 into the `common.egldef` file.

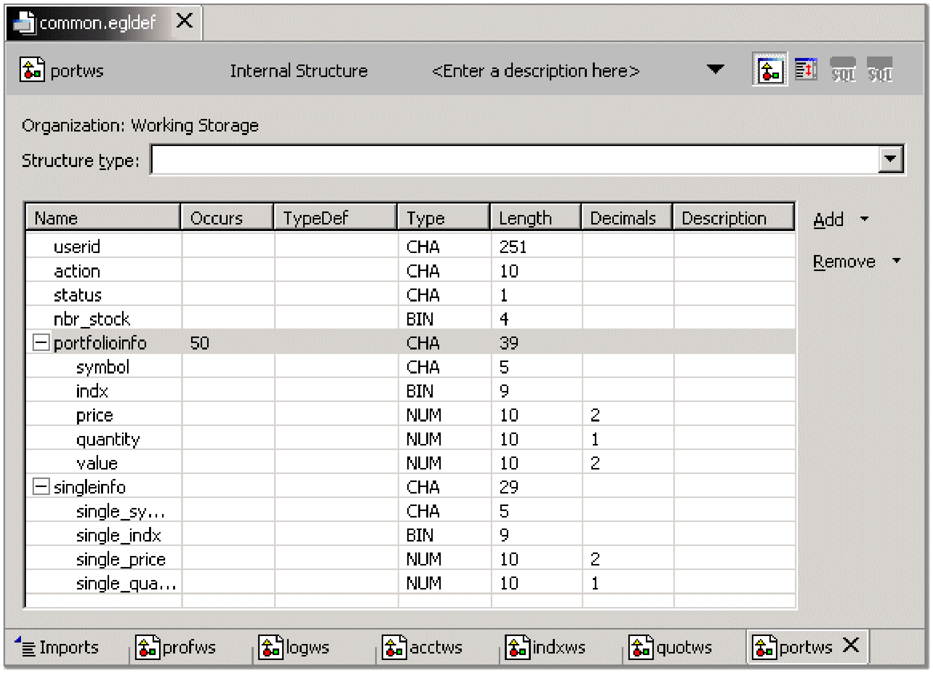
Important: For the small sample application described in this chapter, only the **logws** record is required.

Table 8-2 Working storage items

Working storage record name	Item name	Type	Length	Decimal
logws	userid	CHA	251	
	password	CHA	251	
	action	CHA	10	
	status	CHA	1	
acctws	userid	CHA	251	
	current_balance	PACK	11	2
	balance_change	PACK	11	2
	action	CHA	10	
	status	CHA	1	
indxws	indx	BIN	9	
	action	CHA	10	
	status	CHA	1	
quotws	action	CHA	10	
	status	CHA	1	
	symbol	CHA	5	
	price	NUM	10	2
	details	CHA	20	
portws	userid	CHA	251	
	action	CHA	10	
	status	CHA	1	
	nbr_stock	BIN	4	

Working storage record name	Item name	Type	Length	Decimal
portws (continued)	portfolioinfo: **occurs 50**	CHA	39	
	symbol indx price quantity value	CHA BIN NUM NUM NUM	5 9 10 10 10	 2 1 2
	singleinfo	CHA	29	
	single_symbol single_indx single_price single_quantity	CHA BIN NUM NUM	5 9 10 10	 2 1

The `common.egldef` file in the EGL Part Editor with all the parts open is shown in Figure 8-6.



Organization: Working Storage

Structure type:

Name	Occurs	TypeDef	Type	Length	Decimals	Description
userid			CHA	251		
action			CHA	10		
status			CHA	1		
nbr_stock			BIN	4		
⊟ portfolioinfo	50		CHA	39		
symbol			CHA	5		
indx			BIN	9		
price			NUM	10	2	
quantity			NUM	10	1	
value			NUM	10	2	
⊟ singleinfo			CHA	29		
single_sy...			CHA	5		
single_indx			BIN	9		
single_price			NUM	10	2	
single_qua...			NUM	10	1	

Figure 8-6 Complete common.egldef file with all EGL parts

A new concept for part definitions: typeDef

One major improvement in part definitions compared to VisualAge Generator is the concept of type definition (`typeDef`).

`TypeDef` is used as a model format for parts for these reasons:

- To identify the characteristics of a variable
- To reuse part declarations
- To enforce formatting conventions
- To clarify the meaning of data

In general a `typeDef` is used to identify an abstract grouping. You can declare a structure part named *address*, for example, and divide the information into *streetAddress1*, *streetAddress2*, and *city*. If a personnel record includes the structure items *workAddress* and *homeAddress*, each of those structure items can point to the format of the structure part named *address*. This use of `typeDef` ensures that the address formats are the same.

A `typeDef` also can be used to declare a variable that is more complex than a data item. For instance, if you declare a variable named **`myRecord`** and point to the format of a part named **`myRecord`**, EGL models the declared variable on that part. If you point instead to the format of a part named **`myRecord02`**, however, the variable is called **`myRecord`**, but has all characteristics of the part named **`myRecord02`**.

The table and sections that follow give details on `typeDefs` in different contexts.

EGL source code

If you open the `common.egldef` file with the EGL Source Editor (close the file, then select the file and *Open With -> EGL Source Editor*) then you can see how the information is stored in the file system. An extract of the EGL source is shown in Figure 8-7.

You can use the EGL Source Editor to make changes to EGL parts. You must have an understanding of the format used to store EGL files.

When using the EGL Source Editor, the content assist feature of the editor comes in very handy to help you with the syntax.

```
Record profws
   10 userid CHA(251);
   10 action CHA(10);
   10 status CHA(1);
   10 profileinfo;
   15 fullname CHA(251);
   15 address CHA(251);
   15 email CHA(251);
   15 creditcard CHA(251);
end
Record logws
   10 userid CHA(251);
   10 password CHA(251);
   10 action CHA(10);
   10 status CHA(1);
end
...
```

Figure 8-7 EGL source (extract)

EGL scripting language

Logic parts are programs and functions that have their implementation specified using the EGL scripting language. An EGL program part is the main logical unit and can be composed of other EGL data and function parts. An EGL function part is a logic unit that is called from another program or function. A function can accept parameters and return a value. A program can be separately generated, but a function cannot.

The definition of *program* and *function* used here have the same meaning as in VisualAge Generator.

This discussion does not provide details on the scripting language except to explain the code that is presented herein. To fully understand the language, please read the Enterprise Developer online documentation.

Evolution from VisualAge Generator language

EGL is based on the 4GL technology used in VisualAge Generator. This language provides a number of benefits:

- You can quickly implement business logic by using the EGL procedural language and an EGL-based debugger.

- You can focus on the problem your code is addressing rather than on the technical complexities of systems, such as CICS, MQSeries, and SQL. For example, you can use similar I/O statements to access different types of external data stores.
- You can code in response to current platform requirements without worrying about future migration. An EGL program written for one target platform can be converted easily for use on another.
- You can produce multiple parts of an application system from the same source. After developing an EGL program, for example, you can generate a Java wrapper, an Enterprise JavaBean (EJB) session bean, and a tier 3 program. This increased efficiency comes into play when you develop software to give users access to a tier 2 servlet, which in turn passes data to a generated Java wrapper, which in turn accesses either a generated program on tier 3 or an EJB server.
- You avoid having to configure a CICS connector when you deploy a generated program on CICS for MVS. A generated Java wrapper on tier 2 reformats the data to be passed between tier 2 and tier 3.
- Because Java or COBOL code is generated from the EGL language, developers do not have to migrate when new versions of Java or COBOL are released. A regeneration creates the updated Java or COBOL code. Developers can leave the complexity of migration to the generation engine.

EGL code generation

From EGL you can generate Java and COBOL code. Java is generated for tier 2 and 3 platforms, Windows 2000, Windows NT, and z/OS UNIX System Services (also known as USS). COBOL programs are generated for the tier 3 platform CICS for MVS.

The code generation process works as follows:

- The source code and necessary components are produced and optionally Enterprise Developer sends each generated part to a target platform.
- Enterprise Developer oversees a preparation step to compile Java programs, to translate, compile, and link CICS COBOL programs, and to bind load modules to a DB2 database. This is performed by a build processor on the target platform.
- The build processor returns a confirmation message and, when sent to z/OS, also returns files with the results of the translation, compilation, link edit, and bind.
- Java code can be generated into an Enterprise Developer project and then no build processor is involved.

EGL language

Table 8-3 shows an extract of the EGL language. For details, refer to the Enterprise Developer Help. Statements can span multiple lines and must be terminated by a semicolon (except for the end statement). Keywords can be entered in uppercase or lowercase.

Table 8-3 EGL elements

Element	Description and syntax

Assignment	Assign a value or expression to a data item:
	`target = expression; // blanks around = sign`
	`aRecord.anItem = a * (b + c);`

if, else	Conditional statement, with optional else clause:
	`if (expression)`
	`   // other statements;`
	`else`
	`   // statement;`
	`end`
	`if (anItem IS BLANKS) ...`
	`if (anItem NOT NUMERIC) ...`
	`if (aRecord IS ERR) ...`

while	Executes statements in a loop:
	`while (expression)`
	`   // other statements;`
	`end`

set	Initialize a record or structure or set an SQL item to null:
	`set aRecord empty; // blank (char) or zero (numeric)`
	`set sqlRecord.anItem null;`

select	Multiple sets of statements where at most one set is executed:
	`select (item or expression)`
	`   case value1:`
	`      // statements;`
	`   case value2, value3:`
	`      // statements;`
	`   default:`
	`      // statements;`
	`end`

call	Call another program. Arguments are passed as reference, that is the called program can change the values of the calling program:
	`call progA(arg1, arg2);`
	`   on exception                    // optional`
	`      // statements;`
	`   end`

Element	Description and syntax

functions	Functions can be called like programs or they can return a value:
	`functA(arg1, arg2);`
	`functB();`
	`anItem = functC(b,c);`
	A function that returns a value must use EZERTN:
	`// statement;`
	`ezertn(result);`

I/O statements	

add	Put a record into a file, message queue, or database:
	`add aRecord;`
	`add aSQLrecord`
	`   on exception ..... // optional on all i/o statements`

inquiry	Read single record from file or database: `inquiry aSQLrecord;`
replace	Replace current record in file or database: `replace aSQLrecord;`
delete	Delete current record in file or database: `delete aSQLrecord;`

update	Read and lock a record in file or database, followed by replace or delete:
	`update aSQLrecord;`
	`// statement to change content;`
	`replace aSQLrecord;`

setinq
setupd
scan
close
	Select a set of rows from a database for retrieval with scan.
	Select a set of rows for retrieval followed by replace/delete.
	Read the next row (also read records in a file).
	Close setinq/setupd, or close a file.
	`setupd aSQLrecord;`
	`   while (....)`
	`      scan aSQLrecord;`
	`      if (....) // change content;`
	`      replace aSQLrecord;`
	`   end`
	`close aSQLrecord;`

Eze words

Eze words are special function words that you can use in your programs. These words provide access to many system-provided values (such as the date and time, or the environment in which the program is running) and useful functions (such as mathematical and string operations).

For a full description of all eze words, refer to the help of the Enterprise Developer.

Here is an extract of some very useful eze words:

`ezefec`	Controls continuation after hard errors. If set to 1 and an error routine is specified, execution continues.
`ezesys`	Identifies environment: `WIN`, `USS`, `MVSCICS`.
`ezecomit`	Function to commit resources: `ezecomit();`
`ezerollb`	Function to roll back resources: `ezerollb();`
`ezesize`	Function to return size of an array: `ezesize(item);`
`ezeclos`	Function to end the current program: `ezeclos();`
`ezertn`	Return a value from a function: `ezertn(value);`
`ezesqcod`	Item with SQL return code of last SQL statement.
`ezesqlca`	Item with entire SQL communication area.
`ezeconct`	Function to connect to a database: `ezeconct(userid, password, jdbc/DataSourceName);`

Mathematical functions:

> `ezeabs` (absolute value), `ezeceil` (round up), `ezefloor` (round down), `ezemin` (minimum), `ezemax` (maximum)

String manipulation functions:

> `ezescmpr` (compare substrings), `ezescnt` (concatenate), `ezescopy` (copy substring), `ezesfind` (find in substring)

Java access functions:

> `ezeJava` (invoke method), `ezeJavaGetField` (retrieve property value), `ezeJavaSetField` (set property value), `ezeJavaIsNull` (check for null object)

Writing an application in EGL

We create a program to handle the login action from the trade sample Struts application. We start by creating an EGL logic part (`logac.eglpgm`) and then start adding other parts that perform various functions to validate the user and to add a new user. All of this work is done in the Enterprise Developer EGL perspective.

The code for the login program uses the `logws` working storage record (defined in the `common.egldef` file) as the input parameter and an SQL record as the data part to perform the SQL select and insert statements against the `traderegistrybean` table based on the value in `logws.action`.

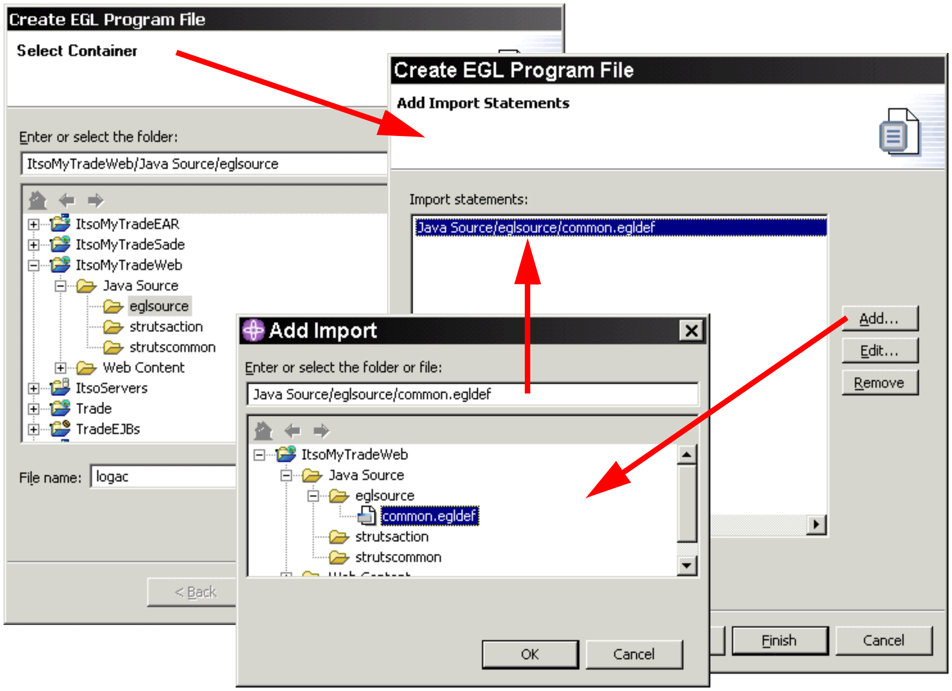
The `logac.eglpgm` file is made up of several parts: one program part, two function parts, and one SQL record part.

Creating an EGL logic part

In the EGL perspective, select the `eglsource` folder in the `ItsoMyTrade` project in the Navigator view and *New -> EGL Program File* from the context menu.

The Create EGL Program File wizard opens (Figure 8-8):

- For the folder select the `eglsource` folder you created earlier.
- Enter `logac` as the filename and click *Next*.
- In the import statements field, insert the `common.egldef` file by selecting *Add* and *Browse* for the file in the project.
- Click *OK* to add the file as an import statement.


Create EGL Program File

Select Container

Enter or select the folder: ItsoMyTradeWeb/Java Source/eglsource

File name: logac

Create EGL Program File

Add Import Statements

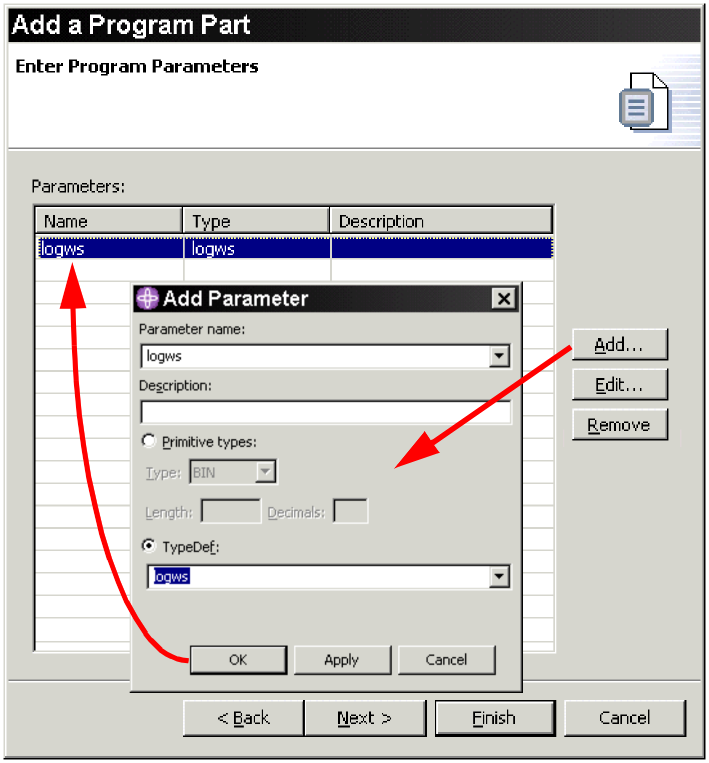
Import statements: Java Source/eglsource/common.egldef

Add Import

Enter or select the folder or file: Java Source/eglsource/common.egldef

Figure 8-8 Create EGL program file

- Click *Next* and enter `logac` as the external name for the program.
- Click *Next*. In the Enter Program Parameters page, we can select a record part from the included `common.egldef` file (Figure 8-9). Click *Add*. In the Add Parameter dialog box, select `logws` from the Parameter name and TypeDef drop-down lists and click *OK*.
- Click *Finish.*



Add a Program Part

Enter Program Parameters

Parameters:

Name	Type	Description
logws	logws	

Add Parameter

Parameter name: logws

Description:

○ Primitive types:

Type: BIN

Length:

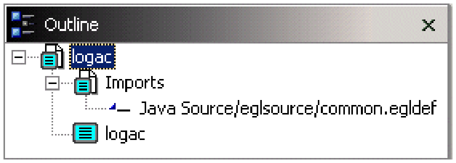
Decimals:

● TypeDef:

logws

Figure 8-9 Using a working storage record as parameter

The `logac` part is added to the Outline view (Figure 8-10) and the logac program is open in the EGL part editor. If the EGL Source Editor is open, close the file and use *Open With -> EGL Part Editor.*



Outline

logac
Imports
Java Source/eglsource/common.egldef
logac

Figure 8-10 logac program in the Outline view

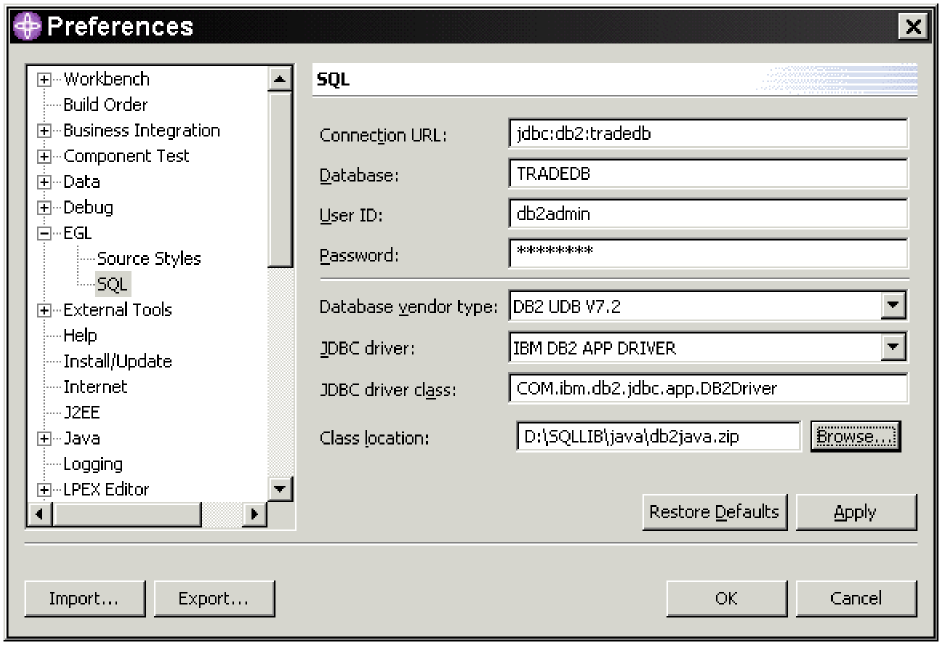
Creating the SQL record

The `logac` program part should already be open now; otherwise, double-click the `logac.eglpgm` file in the `Trade` project `eglsource` folder. This opens the EGL file in the Outline view and the first part defined in the file in the EGL part editor. To open other parts in the EGL file, expand the parts hierarchy in the Outline view and double-click the part you want to open.

SQL preferences

Before we can populate the data items of the SQL record from the database catalog, we have to set the SQL preferences for the EGL plug-in.

- From the Enterprise Developer main menu, select *Window -> Preferences.*
- In the preferences tree on the left-hand side, open the *EGL* section and select *SQL* (Figure 8-11).



Connection URL: jdbc:db2:tradedb

Database: TRADEDB

User ID: db2admin

Password: *********

Database vendor type: DB2 UDB V7.2

JDBC driver: IBM DB2 APP DRIVER

JDBC driver class: COM.ibm.db2.jdbc.app.DB2Driver

Class location: D:\SQLLIB\java\db2java.zip

Figure 8-11 SQL preferences

- For the connection URL, enter `jdbc:db2:tradedb`.
- For database, enter `TRADEDB`.
- Enter a valid user ID and password.
- If you are using DB2 UDB Version 7.2 as the database system, then complete the page with this information:

  - Database vendor type is DB2 UDB V7.2.
  - JDBC driver is IBM DB2 APP DRIVER.
  - JDBC driver class is COM.ibm.db2.jdbc.app.DB2Driver.
  - For the class location enter <SQLLIB HOME>java\db2java.zip where <SQLLIB_HOME> is the location where DB2 is installed.
- Click *OK*.

Add SQL record to EGL program part

We can now proceed to add the SQL record named registry.

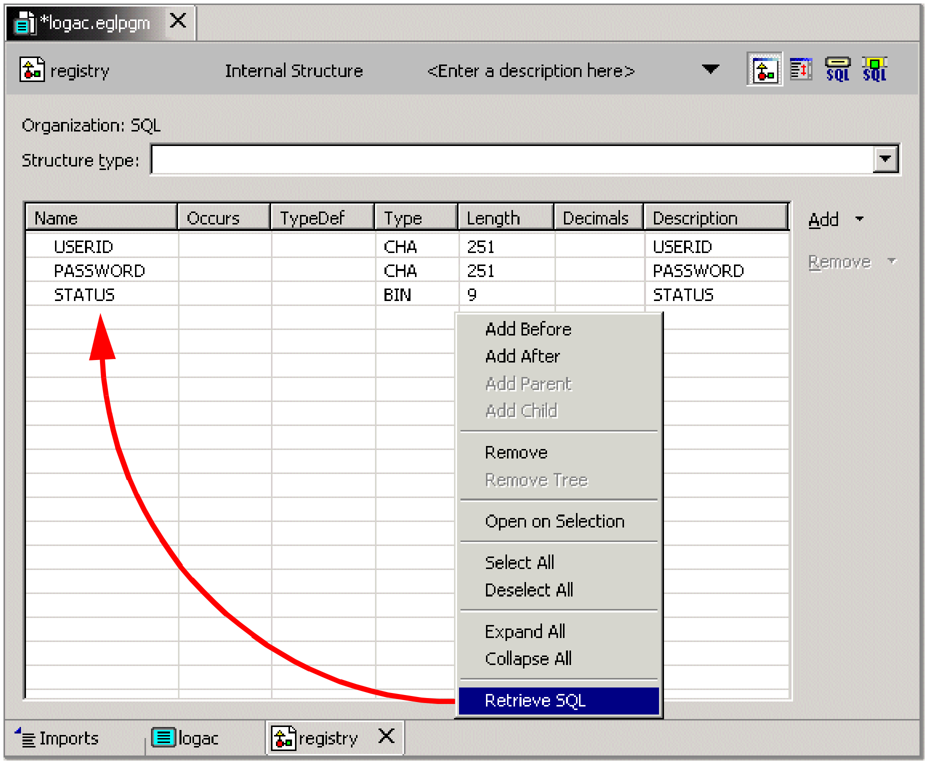
- Select the logac program part in the Outline view and *Add Part* from the context menu.
- In the Add EGL Part wizard, select the *Record* radio button and click *Next.*
- Enter registry as the part name and click *Next.*
- In the Select Record Organization page, select the *SQL Record* radio button and click *Next*.
- In the Enter SQL Properties page, we have to specify the tables that are used for this SQL record. This record interacts with the traderegistrybean table in the tradedb database:
  - Click the *Add* button. In the Name column for the new entry, type traderegistrybean and press the *Enter* key to commit the changes. Leave the label as T1.
  - Click *Finish*.

The registry SQL record part is added to the Outline view and opened in the part editor in the right-hand pane. The part editor is opened to the structure page where the structure of the SQL record can be entered. Because we already have the table defined in the tradedb database, it is easier to import the table structure from the database into the EGL part.

Import the table structure

Place the cursor inside the structure table and select *Retrieve SQL* from the context menu and wait for the request to complete.

The editor obtains the SQL structure from the TRADEREGISTRYBEAN table in the TRADEDB database. It knows the table name because it was defined when we created the registry record. We also defined the database connection properties in the SQL preferences. The SQL structure for the table appears in the SQL structure view shortly (Figure 8-12).



Organization: SQL

Structure type:

Name	Occurs	TypeDef	Type	Length	Decimals	Description
USERID			CHA	251		USERID
PASSWORD			CHA	251		PASSWORD
STATUS			BIN	9		STATUS

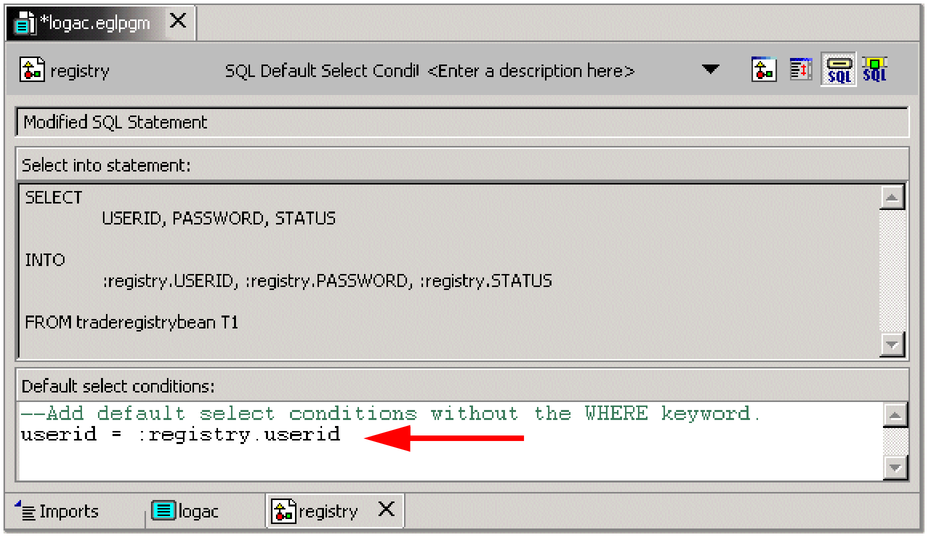
Figure 8-12 SQL record structure for the registry

Open the SQL Select Definitions page by clicking the SQL icon labeled *Show SQL Select Definitions* at the top-right of the part editor (hold the mouse pointer over the icon to see the label appear in hover help). Note that a default select statement for the table has been generated for you.

In the *Default select conditions* white space area, the user can enter the condition that would be used in the SQL where clause for this table. For the `registry` SQL record, enter the default select condition as:

```
userid = :registry.userid
```

Figure 8-13 shows the SQL record with the where clause.



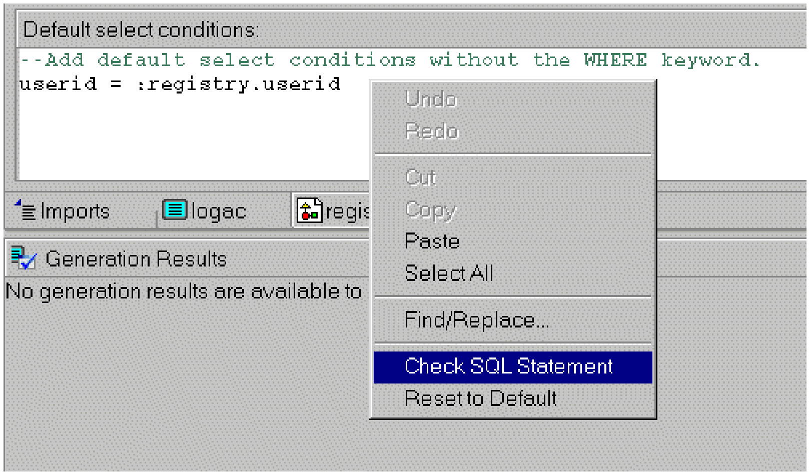
```
SELECT
          USERID, PASSWORD, STATUS

INTO
          :registry.USERID, :registry.PASSWORD, :registry.STATUS

FROM traderegistrybean T1
```

```
--Add default select conditions without the WHERE keyword.
userid = :registry.userid
```

Figure 8-13 Default SQL and select condition for the registry SQL record

SQL syntax check

It is a good idea to check that the SQL select statements are correct. This can be done before closing the dialog. Place the cursor inside the Default select conditions pane, select *Check SQL Statement* from the context menu, and wait for the request to complete (Figure 8-14). If the SQL statement is correct, a message box displays *Check SQL statement successful*.



```
--Add default select conditions without the WHERE keyword.
userid = :registry.userid
```

No generation results are available to

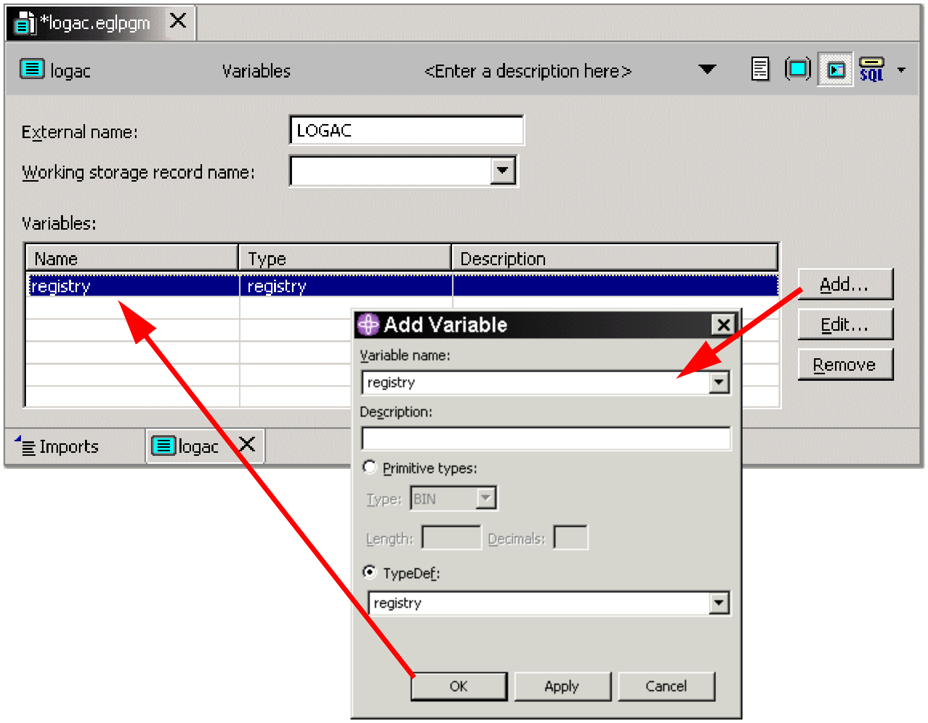
Figure 8-14 SQL statement syntax check

Save the file and close the `registry` SQL record part by clicking the ‘X’ on the `registry` tab. This completes defining the SQL record part and we can now refer to it in the program and functions that we add to the `logac.eglpgm` file.

Declare SQL record as program variable

The registry record part must be declared to the logac program as a variable:

- Make sure the `logac` program part is in focus in the part editor by selecting the `logac` tab. If the `logac` tab does not exist, then open the `logac` part by double-clicking it in the Outline view for the `logac.eglpgm` file.
- Click the *Show Variables* icon .
- Click *Add*.
- In the Add Variable dialog box, select `registry` from both the variable name and TypeDef drop-down lists and click *OK*.
- The record variable is added to the program (Figure 8-15).



*logac.eglpgm

logac Variables <Enter a description here>

External name: LOGAC

Working storage record name:

Variables:

Name	Type	Description
registry	registry	

Add Variable

Variable name: registry

Description:

○ Primitive types:

Type: BIN

Length: Decimals:

● TypeDef:

registry

Figure 8-15 SQL record as variable

Creating the program logic part

Make sure the `logac` program part is in focus in the part editor by selecting the `logac` tab. Open the script page for the `logac` program part by clicking the *Script* icon ▤. In the script pane, enter the code shown in Figure 8-16.

```
ezefec = 1;
registry.userid = logws.userid;
registry.password = logws.password;
registry.status = -1;
logws.status = "1";

if (logws.action = "inquire")
  registry-select();
  if (registry is nrf)
    logws.userid = " ";
    logws.status = "0";
  end
  if (logws.password != registry.password)
    logws.status = "0";
  end
else
  if (logws.action = "add")
    registry-add();
    if (registry is err)
      logws.status = "0";
    end
  else
    logws.status = "0";
  end
end
ezefec = 1; // better would be ezertn()
```

Figure 8-16 Logac program part script

While entering the code, you can also get help with program statements and variables by pressing *Ctrl+spacebar* to activate content/code assist:

- The content assist feature of the EGL editor enables you to automatically complete part names, eze words, insert statement templates, and more.
- Given the context of the cursor position, content assist provides a list of possible additions to the current statement based on the position of the cursor within the script.
- Content assist proposals are based on EGL visibility and scoping rules, and the part definitions in effect at the time the feature is invoked. Experiment with using content assist as you enter the code.

When you have completed entering the code, save the changes. The Tasks list shows many errors because we have not yet defined the underlying functions.

At the top of the program, the `ezefec` special variable is set to 1. This determines how EGL handles hard I/O errors encountered while running the program:

- Setting `ezefec` to 1 causes EGL to run a specified exception clause if one is present.
- If `ezefec` is set to 0, then a system error results and the program stops running.
- Exception clauses are shown in the script for the `registry-select` and `registry-add` functions in Figure 8-17 on page 199 and Figure 8-18 on page 200.

The program then initializes the `registry` SQL record item with the user ID and password from the `logws` input parameter. The requested action is stored in the `logws.action` field and can be either `inquire` or `add`.

For an inquire request, the `registry-select` function is called:

- At the completion of the `registry-select` function, the error state of the registry record is tested. The value *`nrf`* is a soft I/O error meaning "no record found". If no record is found, then the status is set to 0 and the user ID set to an empty string.
- For a successful retrieve the passwords are compared and the status is set to 0 if they do not match.

For an add request, the `registry-add` function is called:

- This function attempts to add the user ID and password to the `traderegistrybean` table. After the add function, the registry record is tested for a hard I/O error called *err*. This error signifies that a nonzero return code was received from the I/O operation.

The final statement (`ezefec = 1`) is there to be able to set a breakpoint. A better statement at this point would be `ezertn()` to return to the caller.

Note: The `logws.status` variable indicates success or failure of the `logac` program: 1 is success, 0 is failure.

Creating function logic parts for SQL access

The main program calls two functions, `registry-select` and `registry-add`, that use SQL to retrieve a row from or insert a row into the table.

Create function to retrieve SQL record

To add a function part:

- Click the `logac` program part in the Outline view and select *Add part* from the context menu.
- In the Add EGL Part wizard, make sure the *Function* radio button is selected and click *Next*.
- Specify `registry-select` as the function name and click *Next*.
- This function does not return a value and does not require any parameters, so click *Next*.
- This function does not use any local variables, so click *Finish*.

The `registry-select` function part is added to the Outline view as a child of the logac program part. The part editor for the function is opened in the right-hand pane. In the script editor, enter the code shown in Figure 8-17.

```
inquiry registry statementID=registry-select on exception
  logws.status = "0";
  ezertn();
end
```

Figure 8-17 registry-select function

`inquiry` is an EGL keyword for SQL statements that execute a select statement on a row record. In this case the select acts on the `registry` record and an SQL statement labeled `registry-select` is executed. Because `ezefec` has been set to 1 in the `logac` program part (see “Creating the program logic part” on page 197), the `on exception` clause causes the statements before the `end` keyword to be executed if there is a hard I/O error as a result of the SQL statement.

We now have to define the SQL statement `registry-select`. The SQL statement view is opened by clicking the *Show SQL Statement* icon and selecting *inquire registry (registry-select)* from the drop-down list.

A default SQL statement has already been generated by the EGL editor. Review the SQL statement. There should be no need to change the statement:

```
SELECT
   USERID, PASSWORD, STATUS
INTO
   :registry.USERID, :registry.PASSWORD, :registry.STATUS
FROM traderegistrybean T1
WHERE
      userid = :registry.userid
```

Save the file and click the *X* button on the `registry-select` tab to close the function part.

Create function to add SQL record

Repeat the above procedure to add a new function called `registry-add`. In the script editor, enter the code shown in Figure 8-18.

```
add registry statementID=registry-add on exception
  logws.status = "0";
  ezertn();
end
```

Figure 8-18 registry-add function

The SQL record that is generated for the add function is:

```
INSERT INTO traderegistrybean
   (USERID, PASSWORD, STATUS)
VALUES
   (:registry.USERID, :registry.PASSWORD, :registry.STATUS)
```

After saving this part, all errors disappear from the Tasks list and you can close the `logac.eglpgm` editor.

EGL visibility and scoping rules

For validation of data declarations and for references between programs and functions, there are two sets of rules: visibility rules and scoping rules. This is a new feature of the EGL that was not present in VisualAge Generator.

Visibility rules

Visibility rules enforce which *defined* data parts can be declared in a logic part and which logic parts (programs and functions) can be called from another part.

Each part is visible to:

- The parent part
- Other descendants of its parent (to any depth)

Parts defined at the top level are visible to all parts in that file.

Visibility rules example

In the example shown in Figure 8-19, the indentation of a name indicates that the named part is declared inside another part.

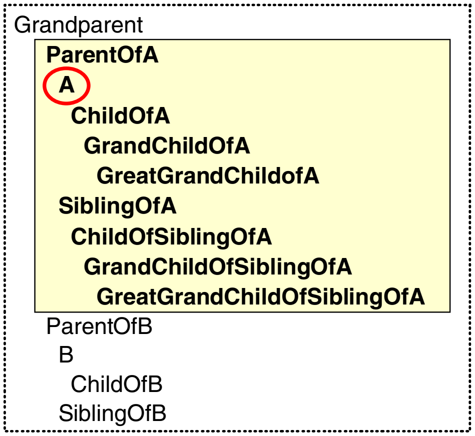


Figure 8-19 Example of visibility rules

Assume that you are working on part **`A`**. The part `A` is protected code and is visible only to the parts in boldface and inside of the box. Part `A` is not visible to the `Grandparent` or to any part subordinate to the `Grandparent` other than the line of descent that starts with `ParentOfA`.

Resolving an EGL part reference

The parts hierarchy is used to find a part referenced by another part. The search order is:

- Children of the part (in the listed order in the file)
- Siblings (in listed order)
- Parent and siblings of the parent (in listed order)
- Repeat the last step for grandparents and their siblings, and so forth
- Top-level parts of imported files

Import

An EGL file can import other files of the same project or other required projects. This is mainly used in program files (`.eglpgm`) to import data parts from definition files (`.egldef`).

In “Creating an EGL logic part” on page 190, the `logac.eglpgm` program imports the data definitions from `common.egldef`.

Scoping rules

Scoping rules enforce which *declared* data parts are accessible by a logic part. Data parts can be in global or local scope.

- Data parts are in *global* scope if they are program parameters or declared parts (variables) in a program, such as input/output records. These parts are accessible to the program and all functions that are invoked.
- Data parts are in *local* scope if they are parameters or declared parts (variables) in a function. These parts are only accessible to that function.

Generating source code from EGL

Generation of Java or COBOL source code is controlled by special EGL control part. Control parts control the EGL generation process for generating Java and COBOL source code and for generating Java wrapper code to be used to call EGL programs from hand-coded Java classes such as Struts action classes and J2EE client applications. This section discusses how control parts are used to generate COBOL and Java code from the EGL program.

EGL control parts

EGL control parts exist in the build file, which has a file extension of `eglbld`.

There are many control parts that impact different aspects of the generation process, and their settings depend on the target generation language and run-time system. The following is a list of the control parts:

Build descriptor A build descriptor controls the overall generation process. The properties in a build descriptor specify how to generate source code from EGL programs and prepare generated output for execution. Some of the properties in this part also define how other control parts are used.

Linkage options A linkage options part specifies how a generated program interacts with other programs. This applies for calls made from generated Java programs or wrappers to other EGL-generated code, generated COBOL program calls to other generated code, and for accessing files on a remote CICS region.

Resource associations An EGL record part has a logical name for a physical file to which it is associated. The resource association control part specifies a file that maps the logical file name to a physical location. A different physical file can be specified for each target system in which a logical file is used. The correct association is used during the generation process.

Link edit Link edit parts apply to COBOL generation where the run-time environment is MVS only. This part specifies how to combine COBOL programs into a load module.

Bind control Bind control parts apply to COBOL generation where the program accesses SQL tables on DB2 and the target environment is MVS only. The bind control part specifies which database request modules (DBRMs) to include in the DB2 plan that is created at preparation time.

Build server

The build server allows generated source code to be compiled on a different machine from the generation machine. Enterprise Developer comes with a build server for each target platform. The build server for Windows and z/OS USS invokes the Java compiler. The build server for MVS invokes the CICS translator, the COBOL compiler, and the DB2 preprocessor if necessary.

Figure 8-20 describes the interactions between Enterprise Developer and the build server.

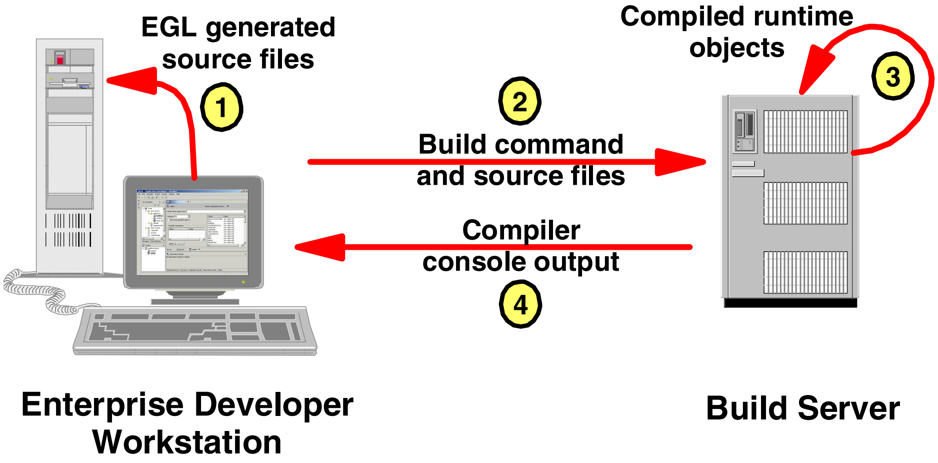

EGL generated source files
1
Compiled runtime objects
3
2
Build command and source files
Compiler console output
4
Enterprise Developer Workstation
Build Server

Figure 8-20 Build server interactions

1. EGL code written in the Enterprise Developer IDE is used to generate source code in either COBOL or Java using an appropriately configured build descriptor part. The generated source code is stored on the Enterprise Developer machine in a location specified in the build descriptor.
2. The generation process sends a build request to the build server along with the source code. The build request is either COBOL or Java, depending on the settings in the build descriptor.
3. The build server stores the source files and performs a build using the options passed to it by the Enterprise Developer workstation. If the build is successful, the built run-time objects are stored on the build server machine in a location specified in the build descriptor.
4. The output of the build process (a success message or compiler errors) is sent back to Enterprise Developer. The output is stored in local files.

The process is the same regardless of whether COBOL or Java source code is compiled.

z/OS build server

We will describe the build server on z/OS in Chapter 10, “Generating COBOL for z/OS from EGL” on page 265.

Windows build server

> **Note:** The Windows build server is not required for builds on a local machine. We recommend that developers build Java code inside the Enterprise Developer Workbench into a project.

A Windows machine can be set up as a build server. This may be an approach where large builds are required and you do not want to tie up the developers' machine.

The build server executable for Windows is named `ccublds.exe` in:

```
<WSED Home>\wstools\eclipse\plugins\com.ibm.etools.egl.distributedbuild_5.0.0\bin
```

where `<WSED Home>` is the file system location where Enterprise Developer is installed.

Accessing the build server from Enterprise Developer

Before the build server can function properly, the run-time environment on the host machine must be properly configured to compile EGL-generated Java source code. This means setting the `path` system variable such that `javac.exe` is available and setting the `CLASSPATH` system variable such that all class referenced in the Java source can be found.

When Enterprise Developer performs a remote build with the build descriptor option `PREP` set to `YES`, it makes a call to the build server program executable to pass it the source code for compiling.

For this to occur, Enterprise Developer requires access to the executable `ccubldc.exe`, which resides in the directory:

```
<WSED Home>\wstools\eclipse\plugins\com.ibm.etools.egl.distributedbuild\bin
```

When you install the Enterprise Developer, the system `PATH` is updated with the directory of the build server and a JAR file is added to the `CLASSPATH` environment variable.

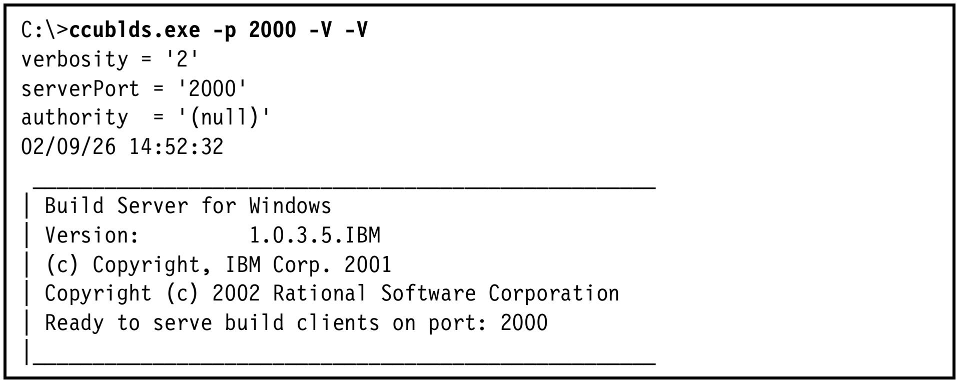
Starting the Windows build server

Once the `PATH` and `CLASSPATH` environment variables have been configured on the build server machine, start the build server by executing this command:

```
ccublds.exe -p 2000 -V -V
```

The -p flag sets the TCP/IP port that the build server will listen on. Make sure the port number is not already in use. The `-V` flag sets the build server console output verbosity level. This can be set up to three times for maximum verbosity. This does not affect the output from the Java compiler.

The console output appears as shown in Figure 8-21.



```
C:\>ccublds.exe -p 2000 -V -V
verbosity = '2'
serverPort = '2000'
authority  = '(null)'
02/09/26 14:52:32
 ____________________________________________________
| Build Server for Windows
| Version:        1.0.3.5.IBM
| (c) Copyright, IBM Corp. 2001
| Copyright (c) 2002 Rational Software Corporation
| Ready to serve build clients on port: 2000
|____________________________________________________
```

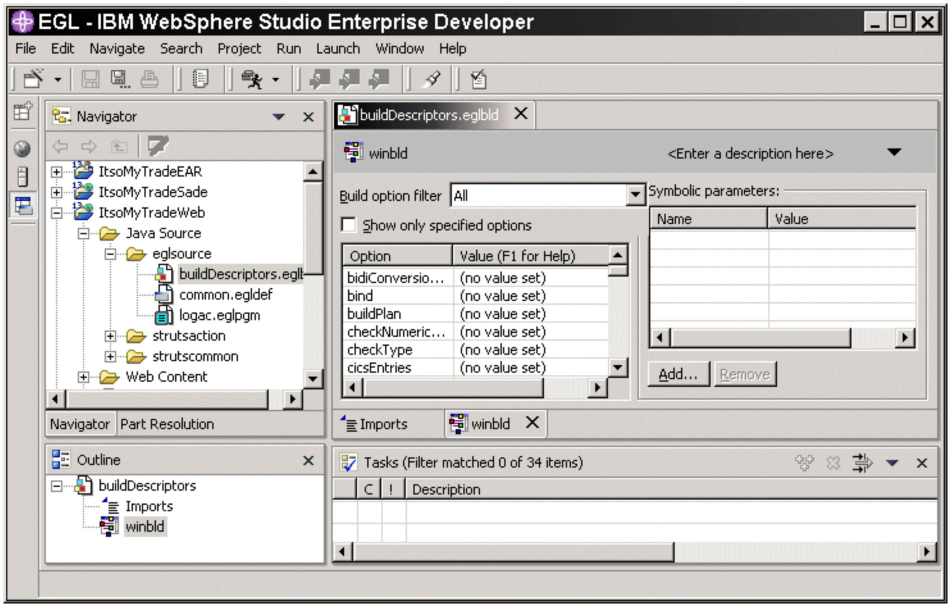
Figure 8-21 Windows build server console output

Creating a build descriptor

We are creating a build descriptor for the trade sample application that can be used to generate Java programs with an EJB session bean wrapper and COBOL programs with a standard JavaBean wrapper.

- In the EGL perspective, select the `eglsource` folder of the sample application and *New -> EGL Build File* from the context menu.
- The Create EGL build file wizard opens. In the Select container page Folder name, select the `eglsource` folder.
- Enter `buildDescriptors` as the file name.
- Click *Next*.
- We do not have to import any other files, so click *Next*.
- In Select type of EGL part to add, select the *Build Descriptor* radio button and click *Next*.
- Enter `winbld` as the name for the build descriptor part and click *Finish*.

Figure 8-22 shows the resulting Enterprise Developer view with the Outline view showing the contents of the build descriptor file and the `winbld` build descriptor part shown in the EGL part editor.



Build option filter: All

☐ Show only specified options

Option	Value (F1 for Help)
bidiConversio...	(no value set)
bind	(no value set)
buildPlan	(no value set)
checkNumeric...	(no value set)
checkType	(no value set)
cicsEntries	(no value set)

Symbolic parameters:

Name	Value

Figure 8-22 Enterprise Developer view with build descriptor

Configuring control parts for Java generation

The build descriptor part editor shows all the options by default. The option list can be narrowed down using the *Category* drop-down list.

Our first build descriptor part generates Java code, so in the *Category* drop-down list, select *Java Target System (Basic)*.

Enter the following values for the options:

genProject `ItsoMyTradeWeb`—The name of the project (with Java support) to store the generated source code into. If the project does not exist in the workspace, it is created during generation.

genProperties `YES`—Specifies that you want to generate environment files that contain settings derived from the build descriptors, linkage options, and resource associations when you generate the Java server program.

packagename `tradeEGL.genned`—Specifies the name of a Java package in which to group related classes and interfaces. If the package does not exist in the project, it is created.

sqlDB `jdbc/tradedb`—Specifies the name of the database accessed by the application. In a J2EE environment, this specifies the data source.

sqlID `db2admin`—The user ID used when installing the trade sample application (see “Loading the trade sample” on page 57).

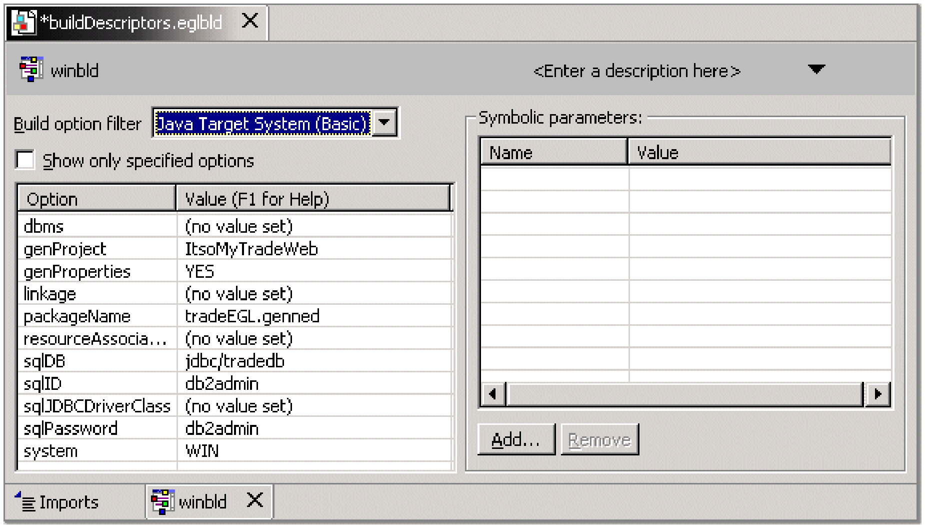
sqlPassword `db2admin`—The password used when installing the trade sample application.

system `WIN`—Specifies the target system for the generated source code. Java code for a J2EE environment on Windows is generated.

If you select *Java Target System (All)*, then you can set more options, for example:

destXxxx You would specify `destHost` and `destPort` when building on another machine using a build server. For building on the local machine, this is not required and no build server must be started.

Figure 8-23 show the build descriptor options.



*buildDescriptors.eglbld

winbld <Enter a description here>

Build option filter: Java Target System (Basic)

☐ Show only specified options

Option	Value (F1 for Help)
dbms	(no value set)
genProject	ItsoMyTradeWeb
genProperties	YES
linkage	(no value set)
packageName	tradeEGL.genned
resourceAssocia...	(no value set)
sqlDB	jdbc/tradedb
sqlID	db2admin
sqlJDBCDriverClass	(no value set)
sqlPassword	db2admin
system	WIN

Symbolic parameters:

Name	Value

Figure 8-23 Build descriptor options for Java

Save the changes and close the `winbld` part by clicking the X on its tab. Refer to the Enterprise Developer Help for details on the build description options not presented here.

Creating linkage options

To access the generated Java code from Struts action classes, we also have to generate Java wrapper classes for the programs. A Java wrapper is a generated Java program that calls another generated program; therefore the build process requires linkage options to specify how the call will be made. The linkage options part is then used as input to a build descriptor part that generates the wrapper code.

We first create linkage options for a simple Java-to-Java call.

- Make sure the `buildDescriptors.eglbld` file is visible in the EGL Outline view.
- Select the *buildDescriptors* node at the top of the hierarchy and select *Add Part* from the context menu.
- Select the *Linkage Options* radio button and click *Next*.
- Enter `wrapperopts` as the name of the part and click *Finish*.

The linkage options are visible in the parts editor; however, the CallLink elements list is empty.

- Click *Add* at the bottom of the CallLink elements table. This adds a line for a program.
- Replace *NewName* with an asterisk character *. This column identifies the program to be invoked by name. By entering an asterisk, we specify that this entry is used for all programs.
- In the type column, select `remoteCall` from the drop-down list. A remote call uses EGL middleware and allows for a return value; a local call would not use EGL middleware.

We must now adjust the properties for the call. With the CallLink entry of * selected, make the following adjustments:

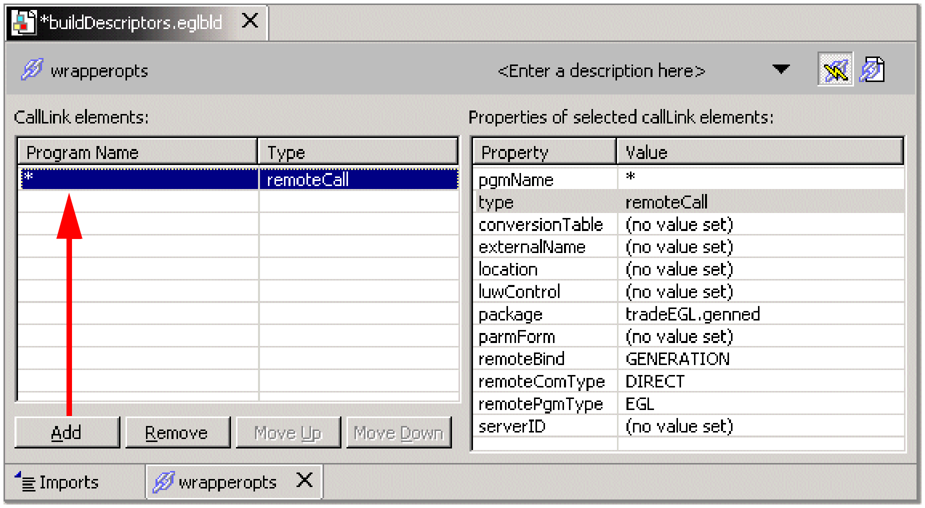
package `tradeEGL.genned`—The package into which the EGL program is generated.

remotePgmType `EGL`—The called program is an EGL-generated Java program.

remoteBind `GENERATION`—The properties used for the call are set at generation time. The other option is `runtime`, in which case the calling options can be set at run time.

remoteComType `DIRECT`—The invocation should be a direct Java invocation in the same process.

Figure 8-24 shows the resulting linkage option part. Save the changes.



CallLink elements:

Program Name	Type
*	remoteCall

Properties of selected callLink elements:

Property	Value
pgmName	*
type	remoteCall
conversionTable	(no value set)
externalName	(no value set)
location	(no value set)
luwControl	(no value set)
package	tradeEGL.genned
parmForm	(no value set)
remoteBind	GENERATION
remoteComType	DIRECT
remotePgmType	EGL
serverID	(no value set)

Figure 8-24 Linkage options for Java wrapper generation

Creating a Java wrapper build descriptor

A Java wrapper build descriptor is a normal build descriptor but with some options set to generate Java wrapper code and an associated linkage options part.

- Make sure the `buildDescriptors.eglbld` file is visible in the EGL Outline view.
- Select the *buildDescriptors* node at the top of the hierarchy and select *Add Part* from the context menu.
- Select the *Build Descriptor* radio button and click *Next*.
- Enter `wrapperbld` as the name of the part and click *Finish*.
- In the *Category* drop-down list, select *Java Wrapper (Basic)*. Enter the following values for the options:

  genProject `ItsoMyTradeWeb`—The Java wrapper code is generated into this project.

  genProperties YES—we want environment files generated.

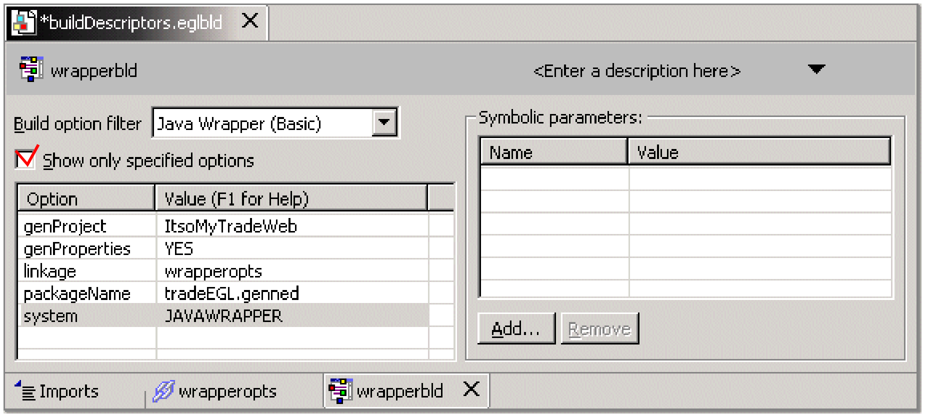
  linkage `wrapperopts`—The `wrapperopts` linkage options part is used to generate the calling properties for the wrapper code.

  packageName `tradeEGL.genned`—Java package for output.

  system `JAVAWRAPPER`—Java wrapper classes are generated.

- Save the changes and close the `buildDescriptors.eglbld` file (Figure 8-25).

To make easier for you to see which options have been specified, check the box *Show only specified options.*



Build option filter: Java Wrapper (Basic)

☑ Show only specified options

Option	Value (F1 for Help)
genProject	ItsoMyTradeWeb
genProperties	YES
linkage	wrapperopts
packageName	tradeEGL.genned
system	JAVAWRAPPER

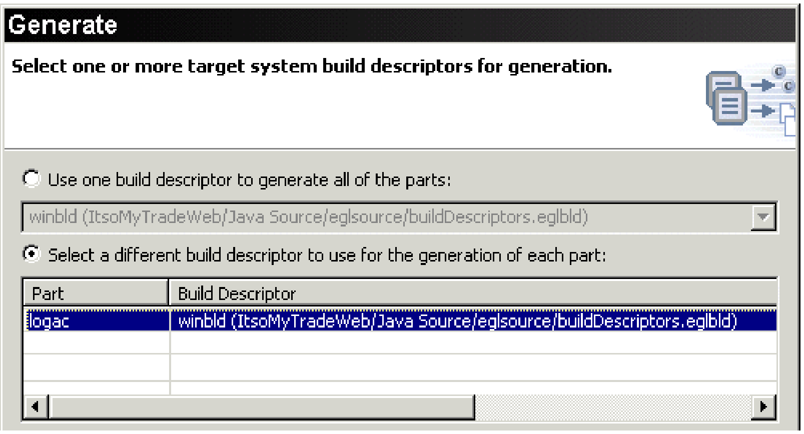
Figure 8-25 Java wrapper build descriptor

Generating Java

Once the control parts have been configured as described, Java code can be generated from the EGL perspective.

- Select the `eglsource` folder and select *Generate EGL With -> Target System Build Descriptor* from the context menu.
- In the dialog box, make sure the check box for `logac` is ticked. Click *Next*.
- For the build descriptor, select the correct `winbld` entry from the drop-down list. You can choose the entry either for all of the parts, or for an individual part. Because we only have one part, it has the same effect (Figure 8-26).

  `winbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)`



Select one or more target system build descriptors for generation.

○ Use one build descriptor to generate all of the parts:

winbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)

● Select a different build descriptor to use for the generation of each part:

Part	Build Descriptor
logac	winbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)

Figure 8-26 Selecting the build descriptor

Be careful to select the correct build descriptor. The `Trade` project from the real sample application shipped with Enterprise Developer also contains a build descriptor named `winbld`.

- Click *Next* (we can skip the SQL user ID panel) and click *Finish*.

The Java source is now generated and placed in the project and package that is specified in the control parts of the build descriptor specified. The Generation Results view opens with messages:

```
IWN.VAL.9994.i 1/1 Program logac generated using build descriptor winbld
from file ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld.
IWN.VAL.9996.i 1/1 Generation completed for program logac with no errors.
```

Generating the Java wrapper

We can now generate the Java wrapper code that is used in the Struts action classes. Perform the same steps as for generating the Java program:

- Select the `eglsource` folder and select *Generate EGL With -> Java Wrapper Build Descriptor* from the context menu.
- In the dialog box, make sure the check box for `logac` is ticked. Click *Next*.
- For the build descriptor, select the correct `wrapperbld` entry from the drop-down list and click *Finish*.

  ```
  wrapperbld (ItsoMyTrade/Java Source/eglsource/buildDescriptors.eglbld)
  ```

> **Tip:** You can select *Generate EGL With -> Target System and Java Wrapper Build Descriptors* to combine the two generation steps. You are then prompted to select the build descriptor for each steps.

Generated Java code

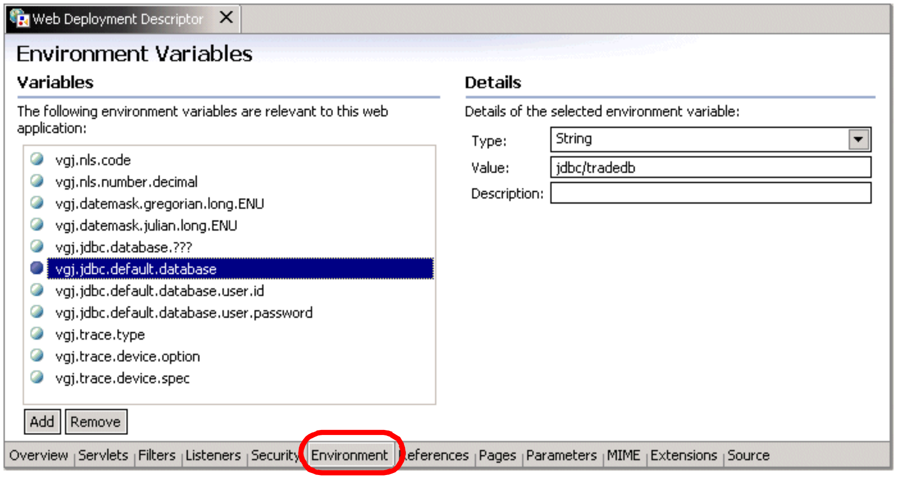
Five Java classes are generated into the `tradeEGL.genned` package—the first three from `winbld` and last two from `wrapperbld`:

- `logac`—The program. The main script is in the `start` method. The two functions are in the `$funcregistry$002dselect` and `$funcregistry$002dadd` methods.
- `Ezelogws`—This class represents the `logws` working storage record.
- `Ezeregistry`—This class represents the SQL record.
- `LogacWrapper`—This is the Java wrapper class.
- `Logws`—This class represents the `logws` working storage record for the Java wrapper.

Generated deployment descriptor

The environment variables, such as database connection information, is stored in the Web application deployment descriptor `web.xml` (in the `WEB-INF` folder).

Open the `web.xml` file and select the *Environment* tab (Figure 8-27).



Environment Variables

Variables

The following environment variables are relevant to this web application:

- vgj.nls.code
- vgj.nls.number.decimal
- vgj.datemask.gregorian.long.ENU
- vgj.datemask.julian.long.ENU
- vgj.jdbc.database.???
- vgj.jdbc.default.database
- vgj.jdbc.default.database.user.id
- vgj.jdbc.default.database.user.password
- vgj.trace.type
- vgj.trace.device.option
- vgj.trace.device.spec

Details

Details of the selected environment variable:

Type: String

Value: jdbc/tradedb

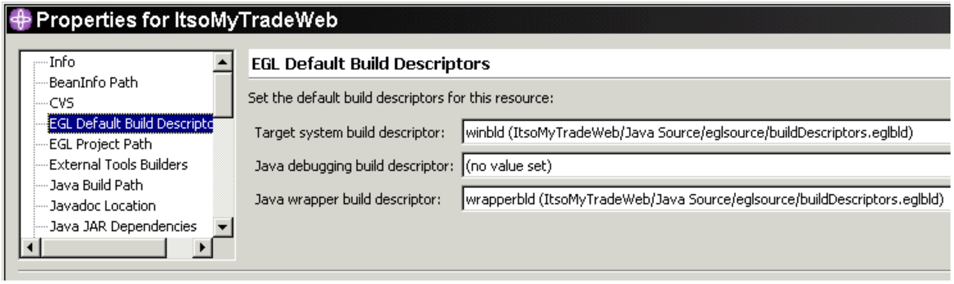
Description:

Figure 8-27 Web deployment descriptor with EGL environment variables

Defining the default build descriptors

Instead of selecting the build descriptor each time you generate code, you can set up the default build descriptors for a project or for an individual file. Select the `ItsoMyTradeWeb` project and *Properties* from the context menu.

Select *EGL Default Build Descriptors* and set the defaults for the three actions (target system, debug, Java wrapper) as shown in Figure 8-28.



EGL Default Build Descriptors

Set the default build descriptors for this resource:

Target system build descriptor: winbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)

Java debugging build descriptor: (no value set)

Java wrapper build descriptor: wrapperbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)

Figure 8-28 Defining default build descriptors

You can define the default for *debug* later after we have defined a build descriptor for debugging purposes.

Testing EGL programs

EGL programs can be tested through the EGL debugger in Enterprise Developer without having to create a J2EE application.

Preparing the project

Because the EGL program accesses the `TRADEDB` database, we have to add the JDBC driver to the Java build path:

- Open the properties of the `ItsoMyTradeWeb` project (select the project and *Properties* from the context menu).
- Select the *Java Build Path* entry and on the Libraries page, click *Add Variable*.
- In the New Variable Classpath Entry dialog, select the *DB2JAVA* variable and click *OK* to add the `db2java.zip` file to the class path.
- Click *OK* to close the properties of the project.

Create build descriptor for debugging

EGL code can be debugged directly in the Enterprise Developer environment. This is done by creating a mapping file that cross-references generated Java code with the EGL source lines from which it was generated. As the Java class runs, the Enterprise Developer EGL debugger views shows the EGL source. The user can step through the EGL statements, inspect EGL data items, and change data values.

To create the appropriate Java code for debugging EGL, we have to create a special build descriptor part. In the `buildDescriptors.eglbld` file, add a new build descriptor part called `windebugbld`. Set the build options as follows:

> **Tip:** Set the category in the build descriptor part editor to *Java Debugging (Basic)* to narrow the build descriptor options list.

dbms `DB2`—Tells EGL the database type.

debug `YES`—Tells EGL to generate Java code with debug hooks into the EGL script.

genProject `ItsoMyTradeWeb`—The Workbench project where the generated code will be stored.

genProperties `YES`—Causes EGL to generate the environment properties file.

J2EE `NO`—The code will not run in a J2EE environment. EGL will store the run-time parameters in a properties file and create a `main` method for the Java class. This is required for debugging the Java class.

packageName `tradeEGL.debug`—Target Java package.

sqlDB `jdbc:db2:tradedb`—For debugging outside of J2EE. It specifies the JDBC URL of the database.

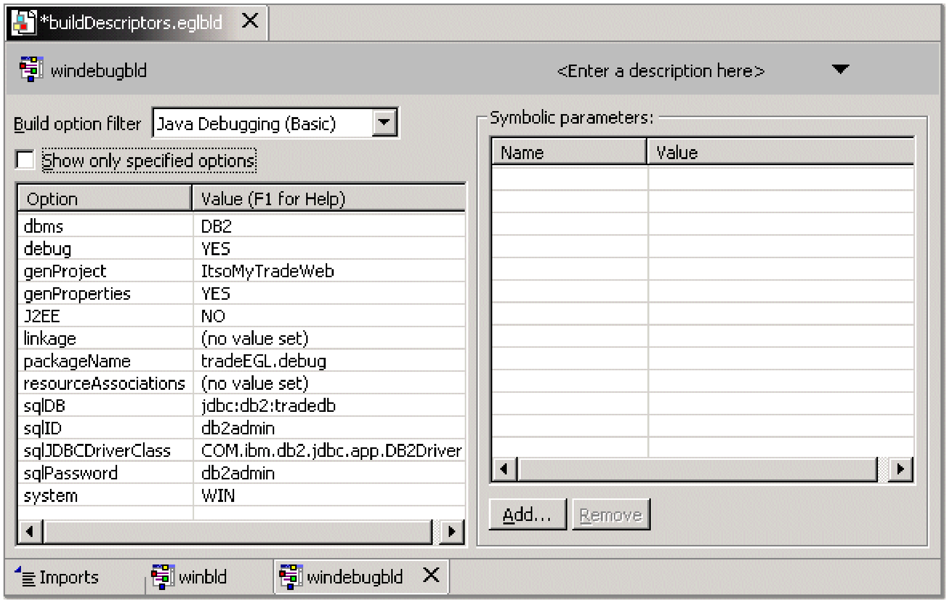
sqlID `db2admin`—The user ID used when installing the trade sample application (see “Loading the trade sample” on page 57).

sqlJDBCDriverClass `COM.ibm.db2.jdbc.app.DB2Driver`— JDBC driver class used to access the database.

sqlPassword `db2admin`—The password used when installing the trade sample application.

system `WIN`—The target system is Windows.

Save the build descriptor and close `buildDescriptors.eglbld` (Figure 8-29).



Build option filter: Java Debugging (Basic)

□ Show only specified options

Option	Value (F1 for Help)
dbms	DB2
debug	YES
genProject	ItsoMyTradeWeb
genProperties	YES
J2EE	NO
linkage	(no value set)
packageName	tradeEGL.debug
resourceAssociations	(no value set)
sqlDB	jdbc:db2:tradedb
sqlID	db2admin
sqlJDBCDriverClass	COM.ibm.db2.jdbc.app.DB2Driver
sqlPassword	db2admin
system	WIN

Name	Value

Figure 8-29 Build descriptor for debugging

Generating code for debugging

We are now ready to generate EGL code that we can debug:

- In the eglsource folder, select the logac.eglpgm file and *Generate EGL With -> Java Debugging Build Descriptor* from the context menu.
- Make sure the check box for the logac program is ticked and click *Next.*
- Select the windebugbld build descriptor and click *Finish*.

  ```
  windebugbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)
  ```

> **Note:** You can add the windebugbld build descriptor as the default to the project properties (Figure 8-28 on page 213).

Generated files

Three Java classes (logac, Ezelogws, Ezeregistry) are generated into the tradeEGL.debug package.

Two files are generated into the Java Source folder:

- The logac_debug.xml file is the mapping file between EGL source code and generated Java code.
- The logac.properties file contains the database connection information.

Debugging EGL code

An EGL-generated Java program can be launched with the EGL debugger by setting up a launch configuration when starting the Java class.

Setting breakpoints

You can set breakpoints in an EGL program in the same way as in a Java program. Open the logac.eglpgm program, place the cursor into the left-hand border of the code, and select *Add Breakpoint* from the context menu (or double-click in the border):

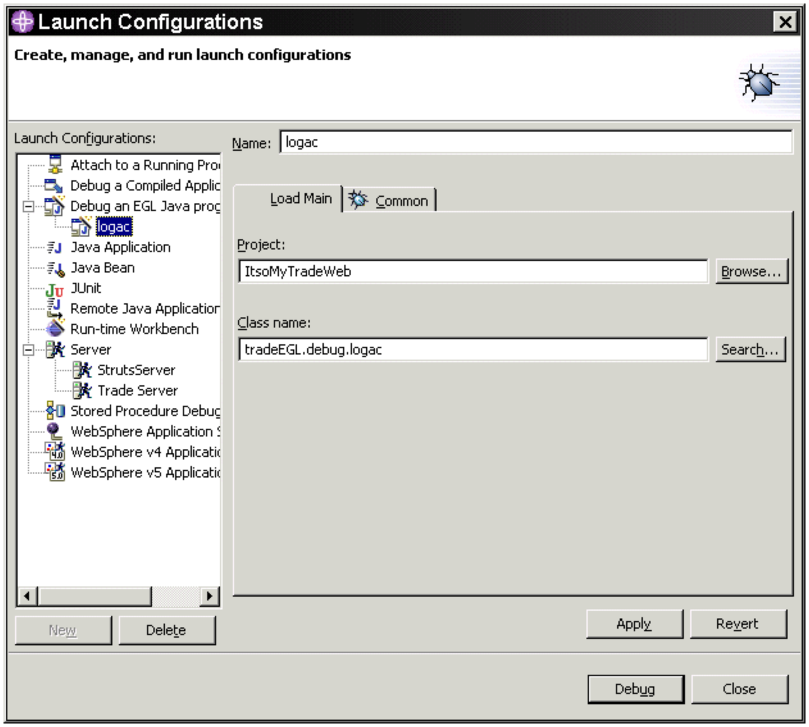
- Set a breakpoint at the start of the program (at the line ezefec = 1).
- Set breakpoints in the two functions that issue SQL calls.
- Set a breakpoint at the last line of the main program. This makes it easy to check the variables after the SQL calls have been processed.
- Close the editor (save is not necessary).

Note that you lose breakpoints when you update the source using the Part Editor.

Configuring the logac program for debugging

The EGL program is run from the Java code that has been generated.

- In a Java perspective, navigate to the `logac` Java program in the `tradeEGL.debug` package.
- Select the `logac` class and click the *Debug* icon or select *Run -> Debug...*
- The Launch Configurations dialog opens (Figure 8-30).
  - Select *Debug an EGL Java program* and click *New* (at the bottom).
  - `logac` is added under *Debug an EGL Java program*.
  - In the right-hand pane, `logac` (as name) and `ItsoMyTradeWeb` (as project) are prefilled. Enter `tradeEGL.debug.logac` as the class name.
  - Click *Apply* and then *Debug*.



Name: logac

Project: ItsoMyTradeWeb

Class name: tradeEGL.debug.logac

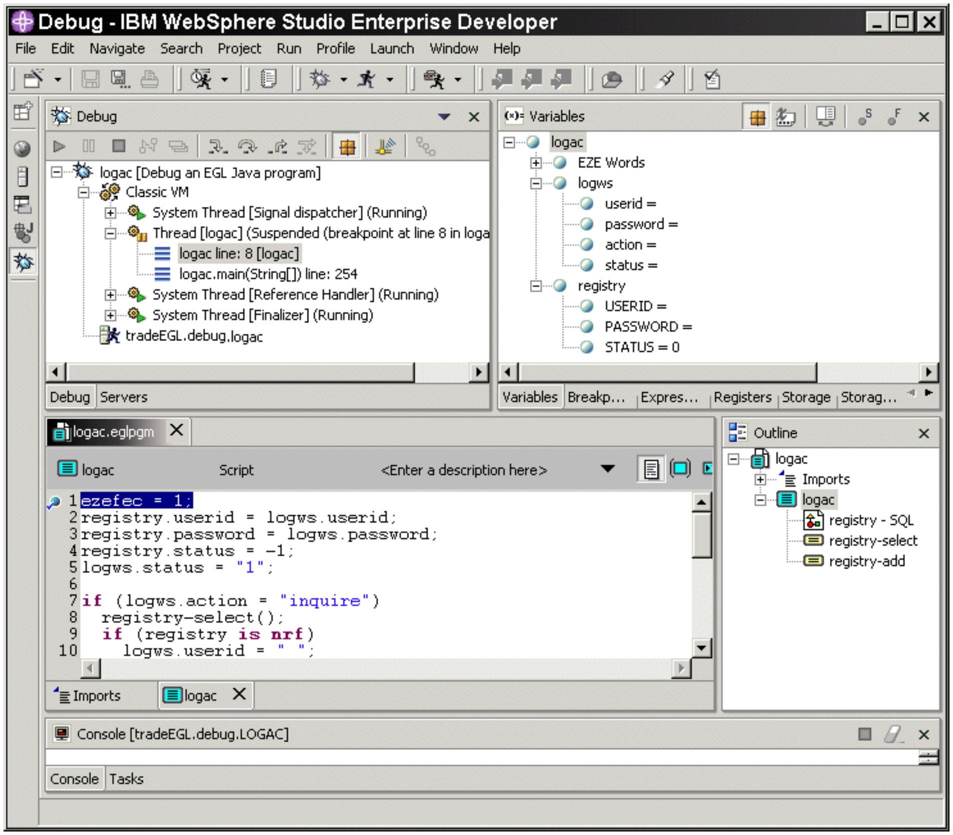
Figure 8-30 Launch configurations

Tip: After setting up the launch configuration, the next time you want to debug the `logac` class, click the arrow pull-down of the *Debug* icon and select the *logac* entry.

Debug perspective

The Debug perspective opens and the program stops at the breakpoint (Figure 8-31):

- Expand the program in the Variables view to see the data items of the `logws` working storage record.
- You can also expand the parameters, variables, and parts in the Outline view.



Debug - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Run Profile Launch Window Help
Debug
Servers
Variables
Breakp...
Expres...
Registers
Storage
Storag...
logac.eglpgm
Script
<Enter a description here>
Outline
Imports
Console [tradeEGL.debug.LOGAC]
Console
Tasks

```
logac [Debug an EGL Java program]
  Classic VM
    System Thread [Signal dispatcher] (Running)
    Thread [logac] (Suspended (breakpoint at line 8 in loga
      logac line: 8 [logac]
      logac.main(String[]) line: 254
    System Thread [Reference Handler] (Running)
    System Thread [Finalizer] (Running)
  tradeEGL.debug.logac
```

```
logac
  EZE Words
  logws
    userid =
    password =
    action =
    status =
  registry
    USERID =
    PASSWORD =
    STATUS = 0
```

```
ezefec = 1;
registry.userid = logws.userid;
registry.password = logws.password;
registry.status = -1;
logws.status = "1";

if (logws.action = "inquire")
  registry-select();
  if (registry is nrf)
    logws.userid = " ";
```

```
logac
  Imports
  logac
    registry - SQL
    registry-select
    registry-add
```

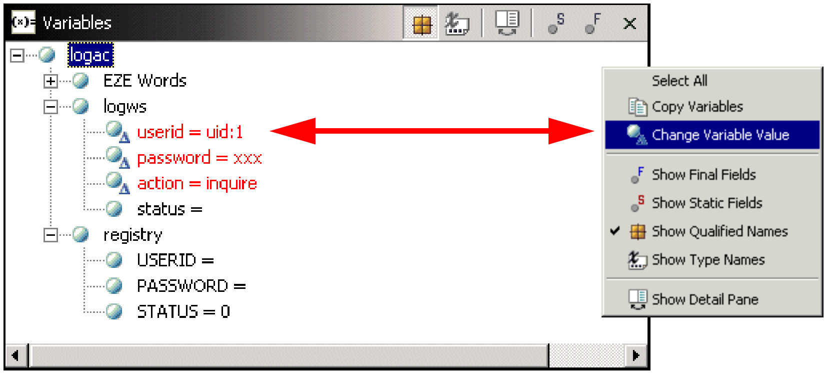
Figure 8-31 Debugging an EGL program

Important: The source pane looks different depending on what EGL editor was used last on the `logac.eglpgm` file. Figure 8-31 shows the format used when the EGL Part Editor was used last.

Debugging the logac program

To run through the retrieve of an existing user, we have to set up the `userid` variable in the `logws` working storage record.

- In the Variables view, select the *userid* item and *Change Variable Value* from the context menu (or double-click the variable name).
- The space after the item name opens for editing. Enter `uid:1` as the new value and press *Enter*. (`uid:1` is a valid user ID in the `traderegistrybean` table.)
- Change the value of the *password* item to `xxx` in the same way.
- Change the value of the *action* item to `inquire` in the same way (Figure 8-32).



```
logac
  EZE Words
  logws
    userid = uid:1
    password = xxx
    action = inquire
    status =
  registry
    USERID =
    PASSWORD =
    STATUS = 0
```

Figure 8-32 Change variable values

Step through the program:

- Use the *Step over* icon in the Debug view. Note how values of changed variables appear in red in the Variables view.
- Use the *Step into* icon when you come to the `registry-select()` line. This brings you into the function instead of stepping over it.
- Continue stepping through the code.

To test the `registry-add` function:

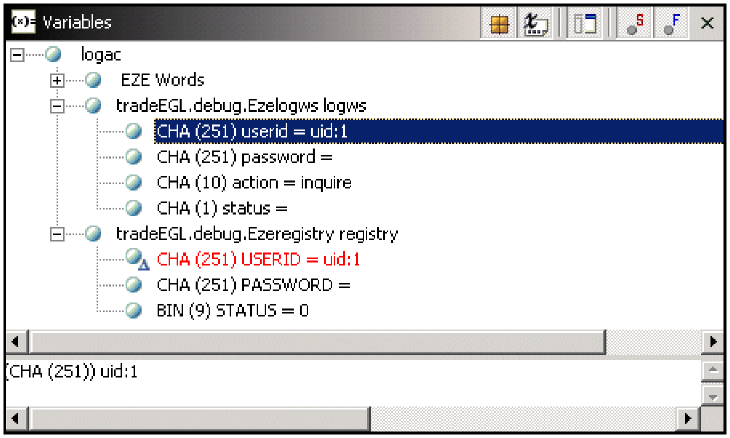
- Restart the debugger by selecting the *logac* entry in the *Debug* icon pull-down.
- At the first breakpoint, change the `logws` variables to:
  - userid: `anything` (for example, your name)
  - password: `anything`
  - action: `add`
- Step through the code or click the *Run* icon .

- When finished, you can check that a row was inserted into the registry table by executing these SQL statements in a DB2 command window:

```
D:\SQLLIB\BIN>db2 connect to tradedb
D:\SQLLIB\BIN>db2 select substr(userid,1,8), substr(password,1,8), status
              from db2admin.traderegistrybean where userid = 'anything'
```

Experiment with the debugger:

- Look at the Breakpoint view to see all the breakpoints. You can remove them easily from this view.
- In the Variables view, select *Show Type Names* and *Show Detail Pane* from the context menu. This displays the types and definitions of selected variables (Figure 8-33).



```
logac
   EZE Words
   tradeEGL.debug.Ezelogws logws
      CHA (251) userid = uid:1
      CHA (251) password =
      CHA (10) action = inquire
      CHA (1) status =
   tradeEGL.debug.Ezeregistry registry
      CHA (251) USERID = uid:1
      CHA (251) PASSWORD =
      BIN (9) STATUS = 0
(CHA (251)) uid:1
```

Figure 8-33 Variable details

Close the Debug perspective when done.

Incorporating EGL code into a Struts application

An EGL-generated Java wrapper for an EGL-generated program is a JavaBean class. This makes it very easy to incorporate the EGL programs into other Java applications as well as Struts applications.

The recommended design practice is to create a model object that accesses the EGL wrapper classes and performs other business logic. The Struts action class calls the model object passing the parameters from the input form. The results from the model object are passed to the output form, which renders the results on the client workstation.

This approach may seem like overkill in our sample application, because we are only accessing one EGL program and not performing any further processing on the result of the EGL program call. However, in a more complex environment, we may be accessing many EGL programs and combining the results using some business logic from another EGL program before returning to the calling object. This processing is too complex for a Struts action class because it will embed too much business logic in the controller components of the MVC-2 pattern.

The Enterprise Developer can create an action class that calls an EGL program wrapper to execute an EGL program. Using this approach, we can see what coding is required in the model class that we want to create.

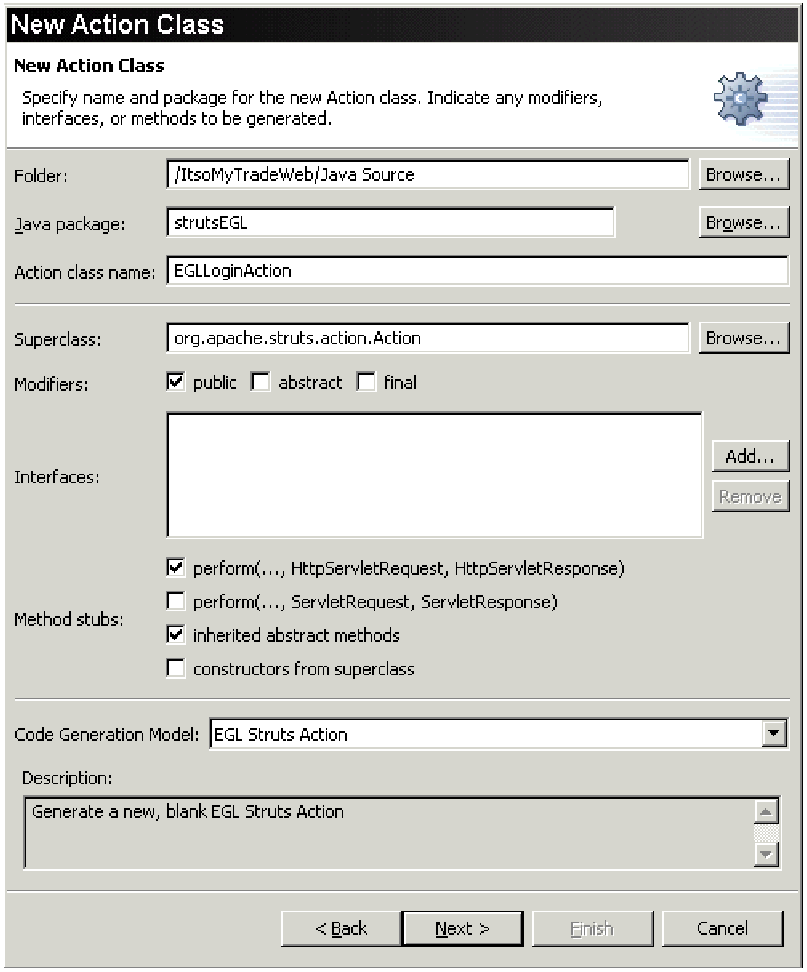
Our approach is:

- Create an action class for an EGL program.
- Break the code into a model class with the processing and a simple action class that calls the model class.

Creating an action class for an EGL program

We create a new action class in the `ItsoMyTradeWeb` project in a new Java package called `strutsEGL`. This package will contain action classes and model objects that access EGL Java programs.

- In the Web perspective create a Java package named `strutsEGL` in the Java Source folder.
- Create a `EGLLoginAction` class (*New* -> *Other* -> *Web* -> *Struts* -> *Action Class*), subclass of `org.apache.struts.action.Action` (Figure 8-34).
  - Make sure the package is set to `strutsEGL`.
  - Select *perform* and *inherited abstract methods*.
  - Select *EGL Struts Action* for the Code Generation Model.



New Action Class

Specify name and package for the new Action class. Indicate any modifiers, interfaces, or methods to be generated.

Folder: /ItsoMyTradeWeb/Java Source

Java package: strutsEGL

Action class name: EGLLoginAction

Superclass: org.apache.struts.action.Action

Modifiers: ☑ public □ abstract □ final

Interfaces:

Method stubs:
☑ perform(..., HttpServletRequest, HttpServletResponse)
□ perform(..., ServletRequest, ServletResponse)
☑ inherited abstract methods
□ constructors from superclass

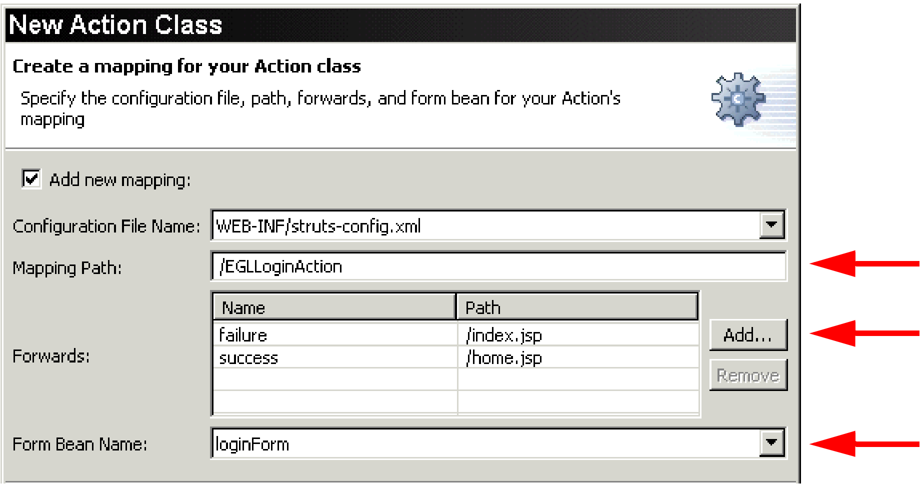
Code Generation Model: EGL Struts Action

Description: Generate a new, blank EGL Struts Action

Figure 8-34 EGLLoginAction class

- – Click *Next.*

- ► For the mapping that is added to the Struts configuration file (Figure 8-35):
  - – Change the mapping path to `/EGLLoginAction`.
  - – Add two forwards, *failure* and *success*, to point to `/index.jsp` and `/home.jsp` (click *Add* twice, then change the two entries).
  - – Select *loginForm* in the pull-down for the form bean name.
  - – Click *Next.*



New Action Class

Create a mapping for your Action class

Specify the configuration file, path, forwards, and form bean for your Action's mapping

☑ Add new mapping:

Configuration File Name: WEB-INF/struts-config.xml

Mapping Path: /EGLLoginAction

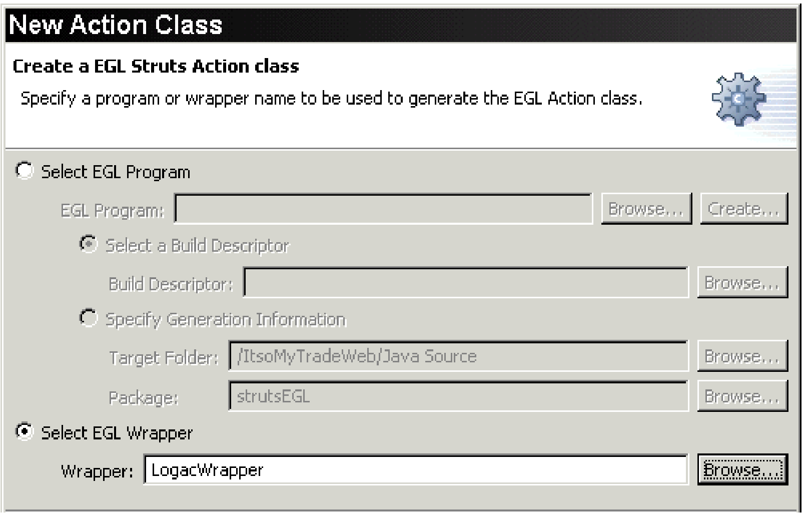
Forwards:

Name	Path
failure	/index.jsp
success	/home.jsp

Form Bean Name: loginForm

Figure 8-35 EGLLoginAction class mapping

- For the program or wrapper name (Figure 8-36):
  - Select *Select EGL Wrapper* and click *Browse* to locate the `LogacWrapper`.
  - We could also select the EGL program itself and a wrapper would be generated for us. Because we already have the wrapper, we just use it.
  - Click *Next*.



New Action Class

Create a EGL Struts Action class

Specify a program or wrapper name to be used to generate the EGL Action class.

○ Select EGL Program

EGL Program:

● Select a Build Descriptor

Build Descriptor:

○ Specify Generation Information

Target Folder: /ItsoMyTradeWeb/Java Source

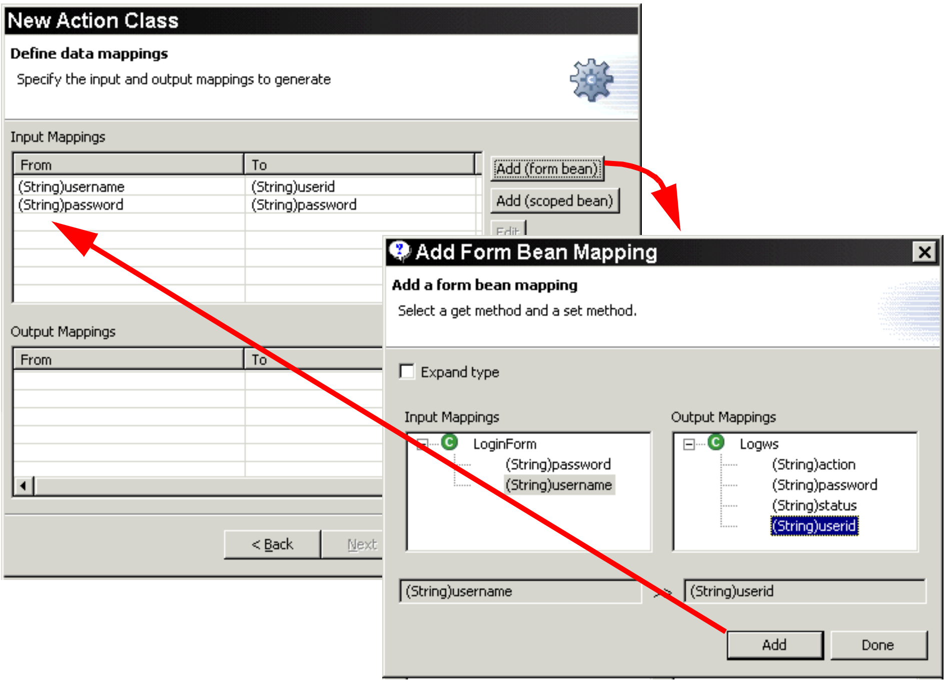
Package: strutsEGL

● Select EGL Wrapper

Wrapper: LogacWrapper

Figure 8-36 EGLLoginAction class program wrapper

- For the data mappings (Figure 8-37):
  - Click *Add (form bean)*. In the dialog, expand the `Loginform` (input, left side) and the `Logws` record (output, right side). Select matching pairs (username—userid, password—password) and click *Add* for each pair.



New Action Class

Define data mappings

Specify the input and output mappings to generate

Input Mappings

From	To
(String)username	(String)userid
(String)password	(String)password

Output Mappings

From	To

Add Form Bean Mapping

Add a form bean mapping

Select a get method and a set method.

☐ Expand type

(String)username >> (String)userid

Figure 8-37 EGLLoginAction data mappings

- Click *Finish* and the EGL`LoginAction` class is opened in the Java editor.

Tailor the perform method

Change the `perform` method try/catch block as shown in Figure 8-38.

The `perform` method creates an instance of the wrapper class (`LogacWrapper`) and the input record (`Logws`), and fills the record with the form data. We set the action and execute the wrapper. The status variable of the `Logws` record signifies if the credentials were successfully authenticated (value 1).

If the authentication was successful, the request object `uidbean` attribute is updated with the user ID information so that further processes know that the user has been authenticated. If the authentication was unsuccessful, then the `uidbean` attribute in the request object is reset and an error object is created and added to the error list.

```
......
import tradeEGL.genned.LogacWrapper;                               <=== generated
import tradeEGL.genned.Logws;                                      <=== not generated

public class EGLLoginAction extends Action {
    private CSOPowerServer powerServer;
    public EGLLoginAction() {
        try {
            powerServer = new com.ibm.vgj.cso.CSOLocalPowerServerProxy();
        } catch (com.ibm.vgj.cso.CSOException e) {
            e.printStackTrace();
        }
    }
    public ActionForward perform(
        ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
                    throws IOException, ServletException {
        // Get the form bean.
        strutscommon.LoginForm actionForm = (strutscommon.LoginForm) form;
        // Create ActionErrors.
        org.apache.struts.action.ActionErrors errors = new ActionErrors();
        // Declare the return value.
        org.apache.struts.action.ActionForward forward = null;
        // Create an instance of the wrapper class
        LogacWrapper wrapper = new LogacWrapper(powerServer);
        // Create instances of each of the record parms
        // ......
        Logws newLogws = wrapper.getLogws();
        // Add your code here to set up newLogws
        newLogws.setUserid(actionForm.getUsername());                 <=== generated
        newLogws.setPassword(actionForm.getPassword());               <=== generated
        newLogws.setAction("inquire");                                <=== set action
        try {
            // execute the wrapper
            wrapper.execute();
            if ( Integer.parseInt(newLogws.getStatus()) == 1 ) {      <=== test value
                request.getSession(true).setAttribute("uidBean",
                                                  actionForm.getUsername());
            } else {
                request.getSession(true).setAttribute("uidBean", "");
                errors.add("login", new ActionError("error.login.failed"));
            }
            powerServer.close();
        } catch (Exception e) {
            // Report the error using the appropriate name and ID.
            // errors.add("name", new ActionError("id"));
            errors.add("login", new ActionError("error.login.exception"));
            e.printStackTrace();
        }
        ........
```

Figure 8-38 EGLLoginAction class extract (changes are in bold)

The power server is closed so that any connections to remote systems are closed.

If an exception is thrown, we add an error to the errors list.

The rest of the generated `perform` method code (not shown in Figure 8-38) either forwards a *success* or *failure* mapping, depending on wheter errors are present in the error list.

Creating the Login model object

To enable our action class to work with different implementations, we extract the main logic from the perform method into its own model class called `Login`:

- Create a `Login` class in the `strutsEGL` package.
- The `Login` class is opened in the Java editor. Add the import statements and a `perform` method to the class, with code as shown in Figure 8-39.
- Save the new class.

Remember that the `logac` program can perform an inquiry of user credentials as well as add a new user to the user registry. In this case, we want to perform an inquiry. Therefore, the `perform` method sets the `action` item for the record instance to `inquire` and also sets the `userid` and `password` that is passed as a parameter to the method.

The `execute` method on the `LogacWrapper` performs the action using the `Logws` instance that was set in the wrapper class. The `Logws` data record also contains an item for the status, which is passed back to the calling program.

Basically, the `perform` method contains the logic from the `EGLLoginAction` class:

- Create a power server, a `LogacWrapper`, and a `Logws`.
- Set the user ID, password, and action into the `Logws` record and execute the wrapper.
- The `perform` method returns the status from the `Logws` record. If exceptions are thrown, a status value of 0 is returned.

```
package strutsEGL;

import com.ibm.vgj.cso.CSOException;
import com.ibm.vgj.cso.CSOLocalPowerServerProxy;
import com.ibm.vgj.cso.CSOPowerServer;
import tradeEGL.genned.LogacWrapper;
import tradeEGL.genned.Logws;

public class Login {

   private CSOPowerServer powerServer;

   public int perform(String userid, String password) {
      try {
         powerServer = new CSOLocalPowerServerProxy();
      } catch (CSOException e) {
         e.printStackTrace();
         return 0;
      }

      // Create an instance of the wrapper class
      LogacWrapper wrapper = new LogacWrapper(powerServer);

      // Set up parameter Logws
      Logws newLogws = wrapper.getLogws();

      try {
         newLogws.setUserid  (userid);
         newLogws.setPassword(password);
         newLogws.setAction  ("inquire");

         // execute the wrapper
         wrapper.execute();
         powerServer.close();

         try { return Integer.parseInt(newLogws.getStatus()); }
         catch (NumberFormatException e) { return 0; }
      } catch (Exception e) {
         System.out.println("Login Exception: " + e.getMessage());
         e.printStackTrace();
         return 0;
      }
   }
}
```

Figure 8-39   Login class with perform method

Creating an action class for the EGL model

We could now remove the logic from the EGLLoginAction class and use the Login model object for the processing. However, it is almost easier to create a new simple action class that uses the Login model. This also enables us to keep the EGLLoginAction untouched:

- Create a EGLLoginAction2 class (*New* -> *Other* -> *Web* -> *Struts* -> *Action Class*), subclass of org.apache.struts.action.Action:
  - Make sure the package is set to strutsEGL.
  - Select *perform* and *inherited abstract methods*.
  - Select *Generic Action Class* for the Code Generation Model.
  - Click *Next*.
- For the mapping that is added to the Struts configuration file:
  - Change the mapping path to /EGLLoginAction2.
  - Add two forwards, *failure* and *success*, to point to /index.jsp and /home.jsp.
  - Select *loginForm* in the pull-down for the form bean name.
- Click *Finish* and the EGLLoginAction2 class is opened in the Java editor.
- Change the perform method try/catch block as shown in Figure 8-40.
- Save the class.

```
try {
   // do something here
   int status = 0;
   String userid   = loginForm.getUsername();
   String password = loginForm.getPassword();
   // make EGL call
   Login login = new Login();
   status = login.perform(userid, password);
   if (status == 1) {
      request.getSession(true).setAttribute("uidBean", userid);
   } else {
      request.getSession(true).setAttribute("uidBean", "");
      errors.add("login", new ActionError("error.login.failed"));
   }
} catch (Exception e) {
   // Report the error using the appropriate name and ID.
   errors.add("login", new ActionError("error.login.exception"));
   e.printStackTrace();
}
```

Figure 8-40 EGLLoginAction2 perform method extract

The perform method creates an instance of the Login model class and calls its perform method passing in the username and password from the input form. The Login.perform returns an integer signifying if the credentials were successfully authenticated.

If the authentication was successful, the request object uidbean attribute is updated with the user ID information so that further processes know that the user has been authenticated.

If the authentication was unsuccessful, then the uidbean attribute in the request object is reset and an error object is created and added to the error list.

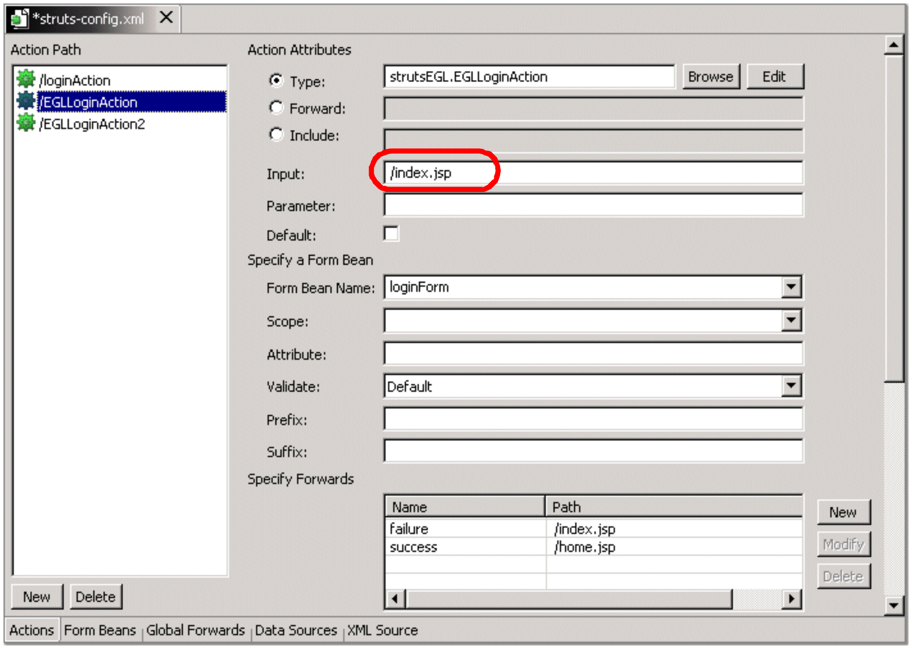
The generated perform method code then either forwards a *success* or *failure* mapping depending on errors being present in the error list.

This completes the changes required to have the Struts application access the EGL-generated code.

Modifying the Struts configuration file

The Struts configuration file has to be modified so that the action class implementation is mapped correctly for the application.

- Open the Struts configuration file (struts-config.xml) located in the Web Content/WEB-INF folder.
- There are now three actions—loginAction, EGLLoginAction, and EGLLoginAction2.
- Select the EGLLoginAction action path:
  - Add /index.jsp in the Input field. We use the same welcome page for the EGL action.
  - Make sure that the Form Bean Name is set to loginForm.
  - Make sure that *success* and *failure* map to home.jsp and index.jsp.
- Make the same change for the EGLLoginAction2:
  - Add /index.jsp as the Input field.
  - Check the Form Bean Name and the success/failure actions.
- Save the changes.
- The Actions page of the configuration file should appear as in Figure 8-41.



Action Path:
- /loginAction
- /EGLLoginAction
- /EGLLoginAction2

● Type: strutsEGL.EGLLoginAction
○ Forward:
○ Include:
Input: /index.jsp
Parameter:
Default: □

Specify a Form Bean
Form Bean Name: loginForm
Scope:
Attribute:
Validate: Default
Prefix:
Suffix:

Specify Forwards

Name	Path
failure	/index.jsp
success	/home.jsp

Figure 8-41 Struts configuration for the EGLLoginAction

Modifying the welcome page

The welcome page (`index.jsp`) invokes the `loginAction`. To use the EGL program we can use either the `EGLLoginAction` or the `EGLLoginAction2`.

We will use the `EGLLoginAction2` and invoke the EGL program through the `Login` model object.

We can change the action in the form:

```
from: <html:form action="/loginAction">
to:   <html:form action="/EGLLoginAction2">
```

Alternatively, we create a duplicate form inside the `index.jsp` so that we can invoke either the Struts action or the EGL action (Figure 8-42).

```
<body>
<h1 align="center"><bean:message key="index.title"/></h1>
<h2>Struts Normal</h2>
<html:form action="/loginAction">                      <==== ORIGINAL FORM
<html:errors/>
<p>
<table>
  <tr>
   <td><bean:message key="global.field.username"/></td>
   <td><html:text property="username" size="20" maxlength="30"/></td>
  </tr>
  <tr>
   <td><bean:message key="global.field.password"/></td>
   <td><html:password property="password" size="20" maxlength="30"/></td>
  </tr>
</table>
<p>
<html:submit><bean:message key="welcome.button.login"/>
</html:submit>
<input type="reset"><br>
</html:form>
<hr>
<h2>Struts EGL</h2>
<html:form action="/EGLLoginAction2">                   <==== DUPLICATE FORM
...
... same as above
...
</html:form>
</body>
```

Figure 8-42 Welcome page with two actions

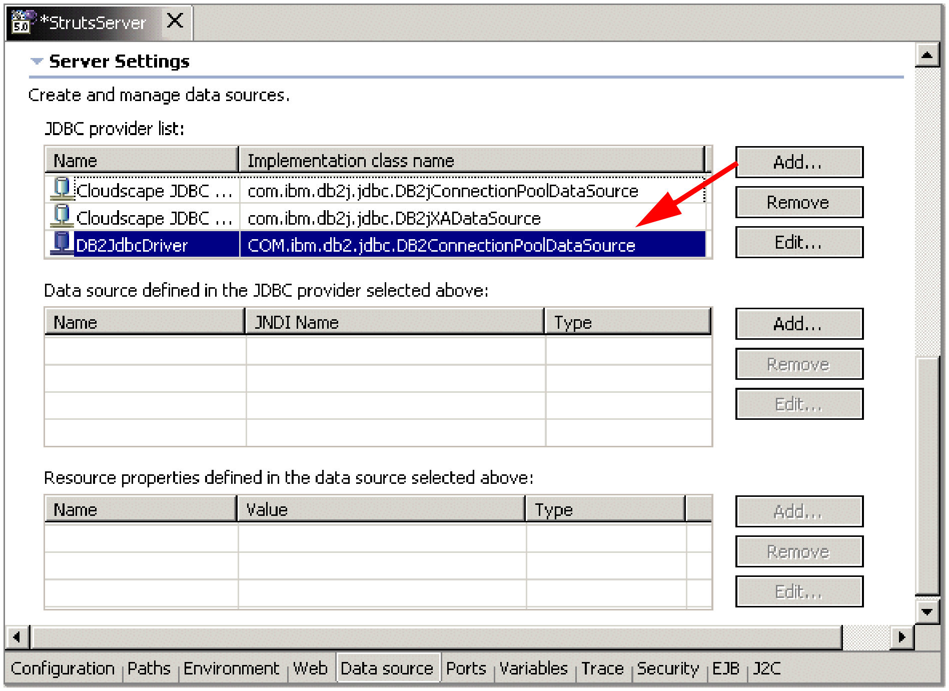
Preparing the Struts server

The Struts application accesses the `TRADEDB` database through the EGL program. We defined the data source as `jdbc/tradedb` (see “Configuring control parts for Java generation” on page 207), and therefore have to define this data source to the test server.

In the Server perspective, open the `StrutsServer` configuration from the Server Configuration view. The only change we have to perform is in the Data source page. Note there are *Node Settings* and *Server Setting*s. Scroll down to Server Settings.

- For JDBC provider list, click *Add*.

- In the dialog, select *IBM DB2* for the database type, *DB2 JDBC Provider* for provider type, and click *Next*.
- Enter `DB2JdbcDriver` as the name. The implementation class name is prefilled with `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`, and the class path points to the `db2java.zip` file. Click *Finish*.
- The `DB2JdbcDriver` is added to the JDBC provider list (Figure 8-43).



*StrutsServer

Server Settings

Create and manage data sources.

JDBC provider list:

Name	Implementation class name
Cloudscape JDBC ...	com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource
Cloudscape JDBC ...	com.ibm.db2j.jdbc.DB2jXADataSource
DB2JdbcDriver	COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

Data source defined in the JDBC provider selected above:

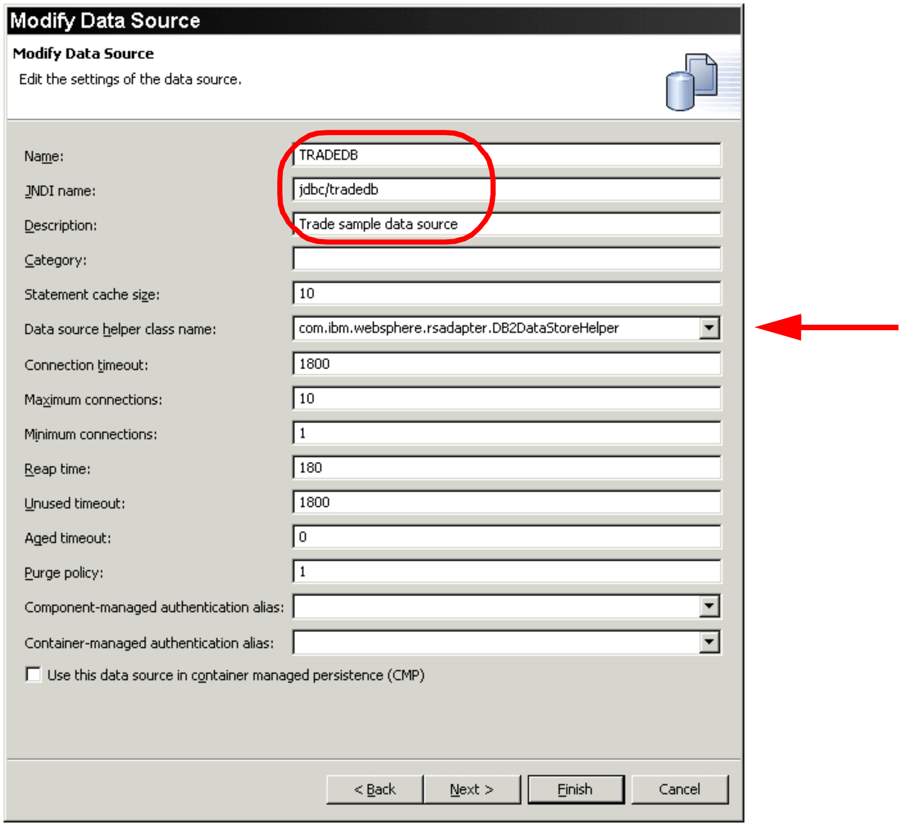
Name	JNDI Name	Type

Resource properties defined in the data source selected above:

Name	Value	Type

Figure 8-43 Adding a JDBC driver to the server

- Select the `DB2JdbcDriver`.
- For data source (below the driver) click *Add*. In the data source dialog:
  - Select *Version 5.0 data source* and click *Next*.
  - Enter `TRADEDB` (as the name), `jdbc/tradedb` (as the JNDI name), and `com.ibm.websphere.rsadapter.DB2DataStoreHelper` for the helper class (Figure 8-44).
  - Click *Next*.



Modify Data Source

Modify Data Source

Edit the settings of the data source.

Name: TRADEDB

JNDI name: jdbc/tradedb

Description: Trade sample data source

Category:

Statement cache size: 10

Data source helper class name: com.ibm.websphere.rsadapter.DB2DataStoreHelper

Connection timeout: 1800

Maximum connections: 10

Minimum connections: 1

Reap time: 180

Unused timeout: 1800

Aged timeout: 0

Purge policy: 1

Component-managed authentication alias:

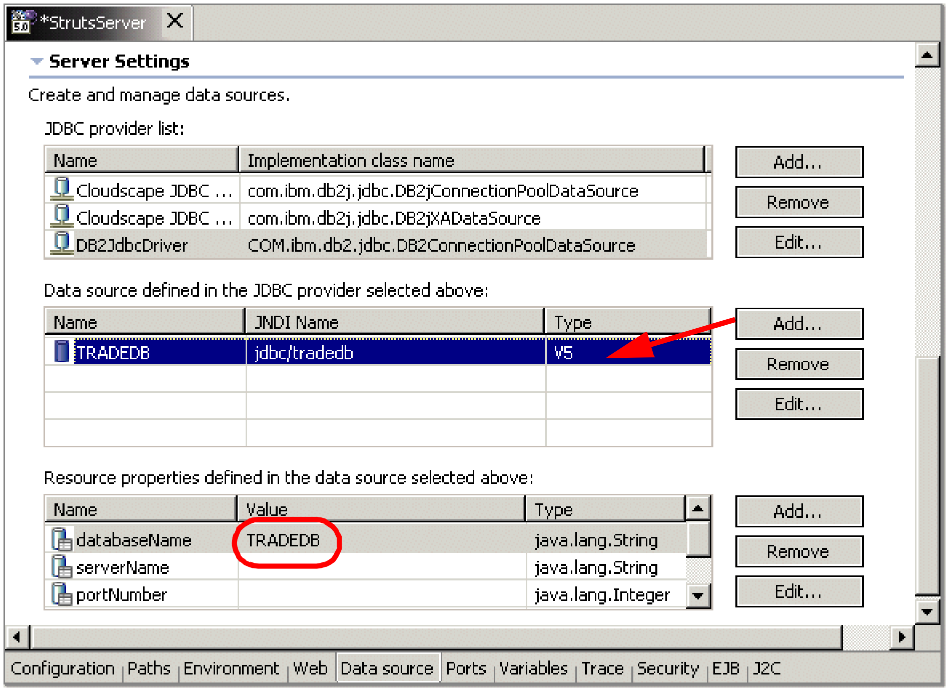
Container-managed authentication alias:

☐ Use this data source in container managed persistence (CMP)

Figure 8-44 Defining the data source for the TRADEDB database

- For the resource properties, select the *databaseName* property and enter `TRADEDB` as value. Leave the other properties as defaults. Click *Finish*.

The Server Settings are shown in Figure 8-45.



Create and manage data sources.

JDBC provider list:

Name	Implementation class name
Cloudscape JDBC ...	com.ibm.db2j.jdbc.DB2jConnectionPoolDataSource
Cloudscape JDBC ...	com.ibm.db2j.jdbc.DB2jXADataSource
DB2JdbcDriver	COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
TRADEDB	jdbc/tradedb	V5

Resource properties defined in the data source selected above:

Name	Value	Type
databaseName	TRADEDB	java.lang.String
serverName		java.lang.String
portNumber		java.lang.Integer

Figure 8-45 Data source for the trade database

- Save and close the server configuration.

Testing the Struts application with the EGL action

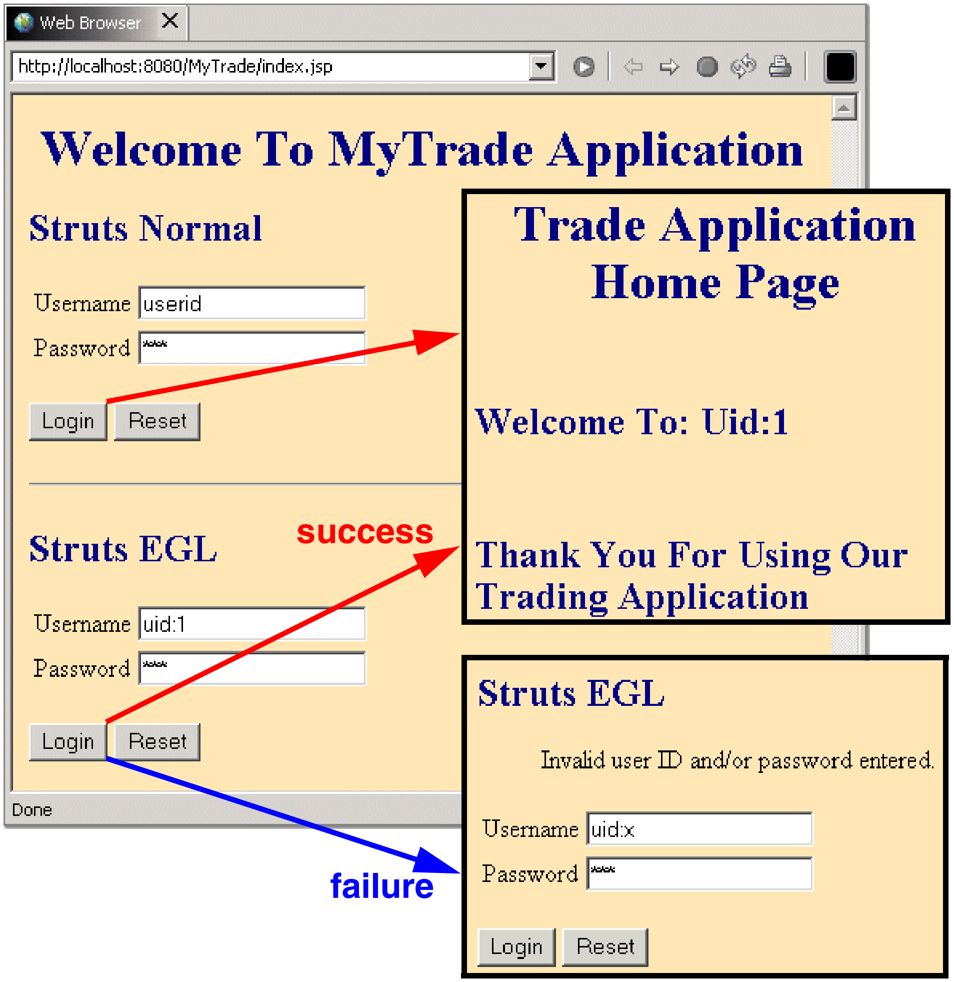
The `ItsoMyTradeEAR` project is attached to the `StrutsServer`. In the Server perspective, start the `StrutsServer`, wait for the console message `Server...open for e-business`, then select the `ItsoMyTradeWeb` project and *Run on Server* from the context menu.

The welcome page (`index.jsp`) appears. In the Struts EGL form, enter a user ID of `uid:1` and a password of `xxx` (`three x's`). If the application is working, you should see the home page in the Struts Web application flow.

The login is authenticated against the registered users in the database. The `traderegistrybean` table has been primed with 500 user IDs starting with `uid:0` up to `uid:499`. The password for each user ID is `xxx`.

Sample run

From the modified welcome page, we can select the normal action or the new EGL action (Figure 8-46).



Welcome To MyTrade Application

Struts Normal

Username userid

Password ***

Struts EGL

Username uid:1

Password ***

Trade Application Home Page

Welcome To: Uid:1

Thank You For Using Our Trading Application

Struts EGL

Invalid user ID and/or password entered.

Username uid:x

Password ***

Figure 8-46 Struts application run with EGL action

Recommendations for EGL files

Here are some recommendations for the number and size of EGL files. These recommendations are based on the facts that:

- EGL files can reference other EGL files
- EGL files can contain multiple parts
- EGL parts can include other parts
- Whenever a file is changed, a rebuild of the file is performed

Rebuilding of a file can be time consuming, depending on how many other files are referenced by the changed file, and by how many other files include the changed file. On the other hand, putting too many parts into a single file has adverse effects on the performance of the editor.

To work efficiently with EGL files and parts, we recommend:

- Put less than 500 parts into a file.
- Keep non-shared parts in the same file as the program that uses them.
- Avoid wildcard includes.
- Group shared parts in egldef files by affinity. Avoid putting lots of unrelated parts in the same shared parts file.
- After following the other rules, minimize the number of shared files you have to include.

9

Implementing EJB actions

This chapter describes creating EJBs from EGL programs and how to access EJBs from a Struts action class.

For a more in-depth discussion on EJBs, please refer to the redbook *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292.

In this chapter, the following topics are discussed:

- How to generate a session EJB as a wrapper to an EGL program
- How to incorporate a session EJB into a Struts action class
- Testing EJB code

Generating EJB session beans from EGL

EGL can generate Java wrapper code as session EJBs. This allows the wrapper code to take advantage of J2EE EJB container functions, such as transaction management, object caching, and object life-cycle management.

Creating an EJB project

Session EJBs must be stored in an EJB project. Create a new project as follows:

- In the New dialog, select *EJB* (left pane) and *EJB Project* (right pane).
- For the EJB Version, select *Create 2.0 EJB Project*.
- Enter `ItsoMyTradeEJB` as the name of the project.
- For the enterprise application project, select *Existing* and *Browse* to the `ItsoMyTradeEAR` project.
- Click *Finish*.

> **Important:** You get a prompt to repair the server configuration and add the EJB project to the server because the owning EAR project is attached to the server. Click *OK*.

The J2EE perspective opens and you can see the new EJB module.

Set the EJB module dependency in the Web project

The `ItsoMyTradeWeb` project will use the session EJB in the EJB project:

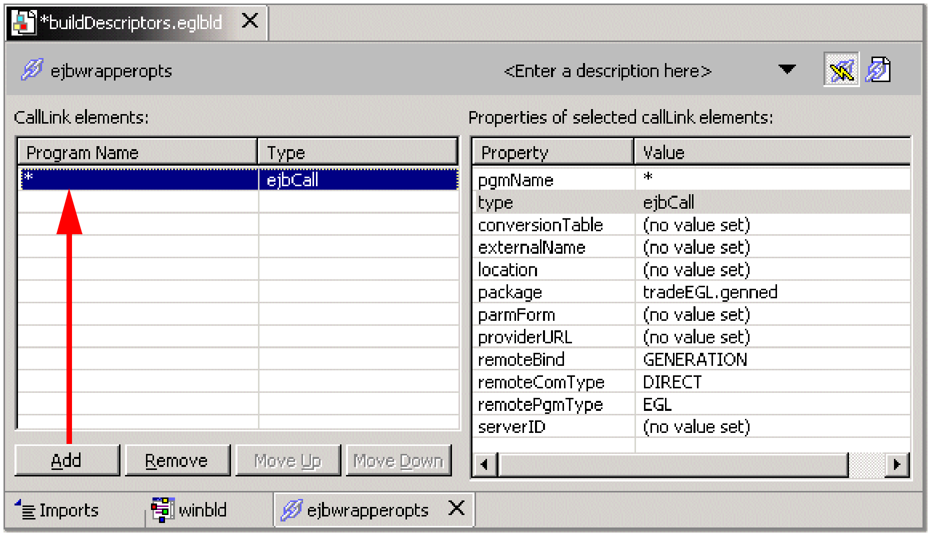
- Select the `ItsoMyTradeWeb` project (expand Web Modules) and *Properties* from the context menu.
- Select *Java JAR Dependencies* and select the `ItsoMyTradeEJB.jar` file.
- Select *Java Build Path* and on the Projects page select the `ItsoMyTradeEJB` project.
- Click *OK*.

Switch back to the EGL perspective.

Creating linkage options for a session EJB wrapper

The linkage options for a Java wrapper control if a session EJB is created. When the call link type for a called program is set to `ejbCall`, EGL generates a wrapper as well as a session EJB. The linkage options part is specified in the build descriptor for a Java wrapper.

- Edit the `buildDescriptors.eglbld` file (in `eglsource` of the `ItsoMyTradeWeb` project) and create a new linkage options part called `ejbwrapperopts` by selecting *Add Part*, select *Linkage Options*, and enter the name `ejbwrapperopts` (Figure 9-1).



Program Name	Type
*	ejbCall

Property	Value
pgmName	*
type	ejbCall
conversionTable	(no value set)
externalName	(no value set)
location	(no value set)
package	tradeEGL.genned
parmForm	(no value set)
providerURL	(no value set)
remoteBind	GENERATION
remoteComType	DIRECT
remotePgmType	EGL
serverID	(no value set)

Figure 9-1 Build descriptor options for EJB generation

- Click *Add* under the CallLink elements table.
- For the new entry set the program name to `*` and the type to `ejbCall`.
- Set these options:

  package `tradeEGL.genned`—The package into which the EGL program was generated. This is not the package where the session EJB will be generated.

  remoteBind `GENERATION`—The properties used for the call will be set at generation time. The other option is `runtime`, in which case the calling options can be set at run time.

  remoteComType `DIRECT`—The invocation should be a direct Java invocation in the same process.

  remotePgmType `EGL`—The called program is an EGL-generated Java program.

- Save the changes.

Essentially, the only difference between the linkage options part of an EJB session bean and a standard JavaBean is the `callLink` type.

Creating a build descriptor for a session EJB wrapper

We can create a new build descriptor to reference this linkage options part:

- Create a new build descriptor for a Java wrapper called ejbwrapperbld.
- In the *Category* drop-down list, select *Java Wrapper (Basic)*. Enter the following values for the options (similar to the options used for wrapperbld (see "Creating a Java wrapper build descriptor" on page 210):

genProject	ItsoMyTradeEJB—The EJB project where the session bean is generated.
genProperties	YES—we want environment files generated.
linkage	ejbwrapperopts—The linkage options part.
packageName	tradeEGL.ejbs—Java package for output session EJB.
system	JAVAWRAPPER—Java wrapper classes are generated.

- Save the changes and close the file.

Generating the session EJB wrapper and the session EJB

We are now ready to generate the EJB session bean wrapper:

- Select the eglsource folder and *Generate EGL With -> Java Wrapper Build Descriptor* from the context menu.
- In the dialog box, make sure the check box of logac is selected. Click *Next*.
- For the build descriptor, select ejbwrapperbld from the drop-down list.
- Click *Finish*.

The EJB session bean source code is generated into the tradeEGL.ejbs package of the ItsoMyTradeEJB project.

Generated session EJB and helper code

Switch to the J2EE perspective and J2EE Hierarchy view. You can see a session EJB named LogacEJBBean (Figure 9-2):

The session EJB consists of three Java classes:

- LogacEJBHome—This is the home interface (used by clients to create an instance). The only method in this class is create.
- LogacEJB—This is the remote interface (used by clients to access the EJB). The only method in this class is call.
- LogacEJBBean—This is the session bean implementation. This class extends CSOSupportSessionBean and implements the call method.

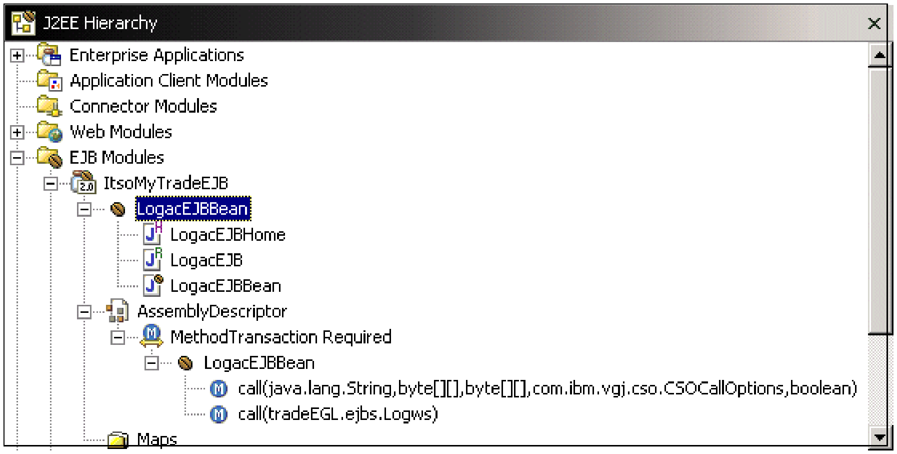


Figure 9-2 Session EJB in J2EE Hierarchy view

The assembly descriptor specifies that transactions are required for the call method on the `LogacEJB` session bean.

Switch to the J2EE Navigator view and you can see that two more classes have been generated into the `tradeEGL.ejbs` package (Figure 9-3):

- `LogacWrapper`—This is the wrapper class that calls the session EJB (in a way, this is the client that uses the EJB).
- `Logws`—This class represents the working storage record that is used as a parameter to the `call` method of the EJB.

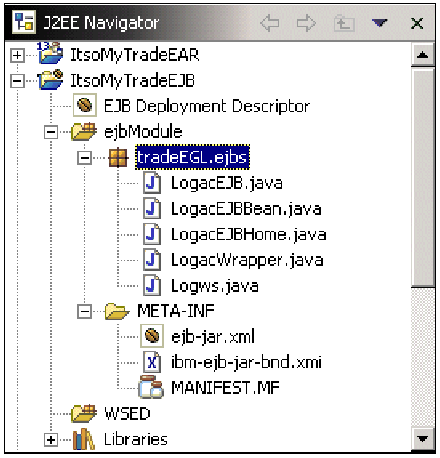


Figure 9-3 Session EJB in J2EE Navigator view

EJB deployment descriptor

In the J2EE view, double-click the ItsoMyTradeEJB module. This opens the EJB deployment descriptor editor, where you can see the definition of the LogacEJBBean by going through the tabs.

Generate the deployed code

Before an EJB can be used, the deployed code and the RMI-IIOP code must be generated and built from the generated EJB session bean source code.

- Select the ItsoMyTradeEJB project and *Generate -> Deploy and RMIC Code* from the context menu.
- Select the check box for the LogacEJBBean and click *Finish*.

The deploy and RMI code is generated. You can see a number of additional classes in the tradeEGL.ejbs package (in the Navigator view).

Regenerating the EGL program and wrapper

The session EJB is in the EJB project and will run in the EJB container of the application server. The generated logac program is in the Web project, but will be accessed by the session EJB. This is not possible in an application server.

We have to generate the logac program into the EJB project for this to work. There is no problem for the Struts application to access the logac program in the EJB project; a Web project can refer to an EJB project.

Important: If you leave the EGL program (tradeEGL.genned package) in the Web project, then not even the direct Struts action to Java wrapper to EGL program will work once the session EJB has been generated.

Moving the package from the Web project to the EJB project is not enough either. When the logac program is generated into the EJB project, information is added to the EJB deployment descriptor of the session bean.

Edit the build descriptor

Edit the buildDescriptors.eglbld file:

- Edit the winbld part and change the genProject to ItsoMyTradeEJB.
- Edit the wrapperbld part and change the genProject to ItsoMyTradeEJB.

Regenerate the EGL program and the wrapper

Select the eglsource folder and *Generate EGL With -> Target System and Java Wrapper Build Descriptors* from the context menu:

- In the dialog box, make sure the check box of logac is selected. Click *Next*.
- For the Java wrapper build descriptor, select wrapperbld from the drop-down list.
- For the target system build descriptor, select winbld from the drop-down list.
- Click *Finish*.

The generation process creates the tradeEGL.genned package in the ItsoMyTradeEJB project (under ejbModule).

Important: Check that the EJB wrapper bean (tradeEGL.ejbs.LogacWrapper) uses iiop:/// to access the name server. In early code iiop:// was generated.

This may be corrected in your system, but check the code to make sure that iiop:/// is used (there are four occurrences).

Check the EJB deployment descriptor

Edit the ItsoMyTradeEJB module in the J2EE perspective (double-click the module):

- On the Beans page, select the LogacEJBBean.
- Scroll down to Environment Variables. You should find a number of environment variables, such as vgj.jdbc.database, that were added to the EJB deployment descriptor.

These variables provide the connection information for the logac program.

Delete the generated code from the Web project

In the ItsoMyTradeWeb project delete the tradeEGL.genned package (select the package and *Delete* from the context menu).

Testing the session EJB

You can test your session EJB using the `StrutsServer` that was created in "Define a WebSphere test server" on page 138.

Start the server

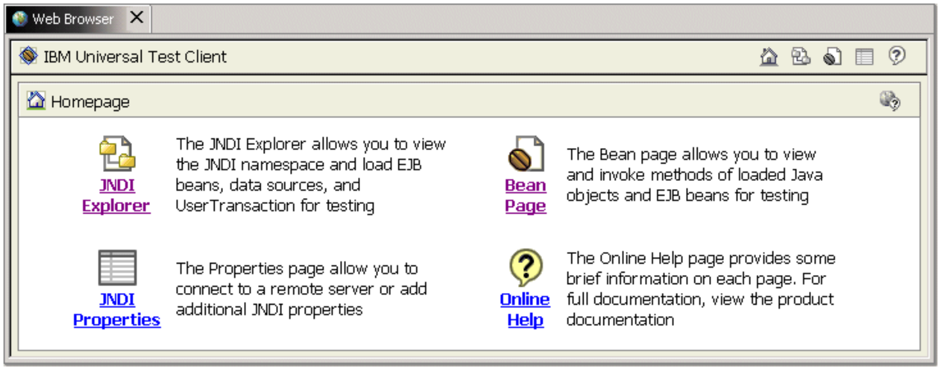
- Open the Server perspective.
- In the Server Configuration view, double-click the `StrutsServer`. Select the Configuration tab and make sure that *Enable universal test client* is selected. Close the editor.
- In the Servers view, select the `StrutsServer` and start it. You know it is ready when the *Server open for e-business* message appears in the Console view.
- Select the `ItsoMyTradeEJB` project and *Run on Server* from the context menu. Select the `StrutsServer`. The universal test client starts in the browser.

Using the universal test client

The universal test client opens on the home page (Figure 9-4).

Tip: You can also start the universal test client in a browser using the URL:

```
http://localhost:9080/UTC/
```



The JNDI Explorer allows you to view the JNDI namespace and load EJB beans, data sources, and UserTransaction for testing

The Bean page allows you to view and invoke methods of loaded Java objects and EJB beans for testing

The Properties page allow you to connect to a remote server or add additional JNDI properties

The Online Help page provides some brief information on each page. For full documentation, view the product documentation

Figure 9-4 Universal test client home page

JNDI Explorer

Select the JNDI Explorer (Figure 9-5).

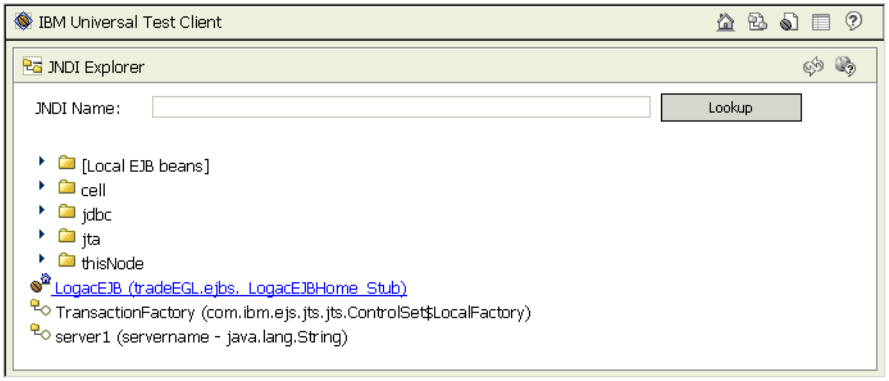
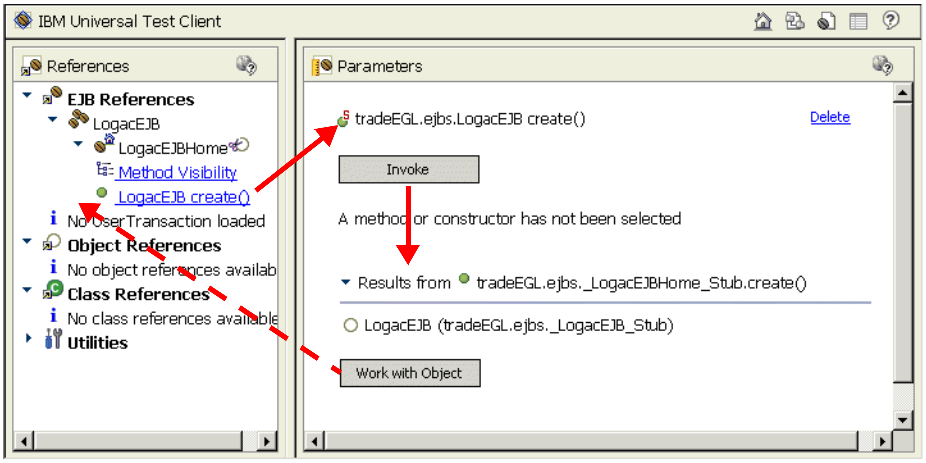

IBM Universal Test Client
JNDI Explorer
JNDI Name:
Lookup
[Local EJB beans]
cell
jdbc
jta
thisNode
LogacEJB (tradeEGL.ejbs._LogacEJBHome_Stub)
TransactionFactory (com.ibm.ejs.jts.jts.ControlSet$LocalFactory)
server1 (servername - java.lang.String)

Figure 9-5 Universal test client JNDI Explorer

EJB page with references

Select the `LogacEJB` to open the EJB page where the home interface is visible under EJB References. Let us create an instance (Figure 9-6).



No UserTransaction loaded

No object references availab

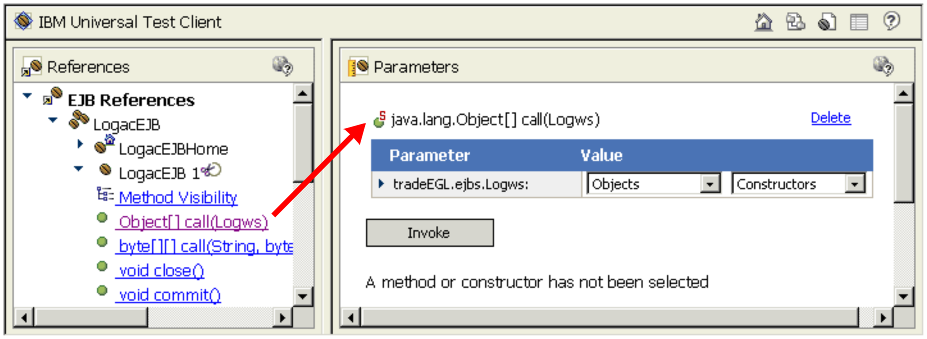
No class references available

A method or constructor has not been selected

Figure 9-6 Universal test client EJB page with home interface

- Expand the `LogacEJB` and `LogacEJBHome` by clicking the arrows. Select the *LogacEJB create* method and it appears in the Parameters pane. Click *Invoke*.
- An instance of the session EJB is created and appears under Results. Click *Work with Object* and the session EJB instance is added to the EJB References.

We want to invoke the call method of the session EJB (Figure 9-7).



java.lang.Object[] call(Logws)

Parameter	Value	
tradeEGL.ejbs.Logws:	Objects	Constructors

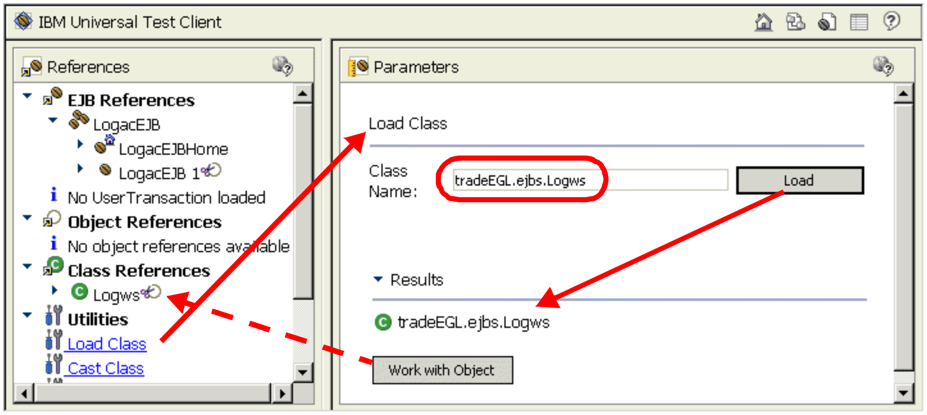
A method or constructor has not been selected

Figure 9-7 Universal test client EJB page with remote interface

- Expand the `LogacEJB` instance to see its methods. Select the *Object[] call* method; this is the method to invoke the EGL program.
- The `call` method appears in the Parameters pane. Looking at the signature, we see that a `tradeEGL.ejbs.Logws` bean is required as the parameter.
- The tool allows us to use a constructor to create a `Logws` bean by selecting the *Constructors* pull-down and click the *Logws()* constructor. However, this constructor creates an empty bean with no values for user ID and password.
- Alternatively, we can use a saved object from the Objects pull-down.

Creating objects for reuse

To create and populate a `Logws` object, we use the Utilities section (Figure 9-8).



No UserTransaction loaded

No object references available

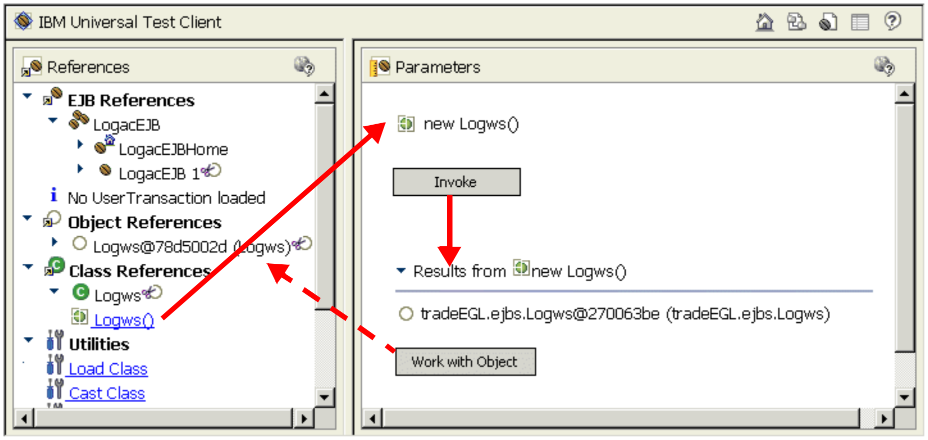
Class Name: tradeEGL.ejbs.Logws

tradeEGL.ejbs.Logws

Figure 9-8 Universal test client loading a class

- Expand Utilities and click *Load Class*.
- Enter `tradeEGL.ejbs.Logws` as class name and click *Load*.
- Click *Work with Object* and the class appears under Class References.

Next we invoke the constructor of the class to create a `Logws` object (Figure 9-9).



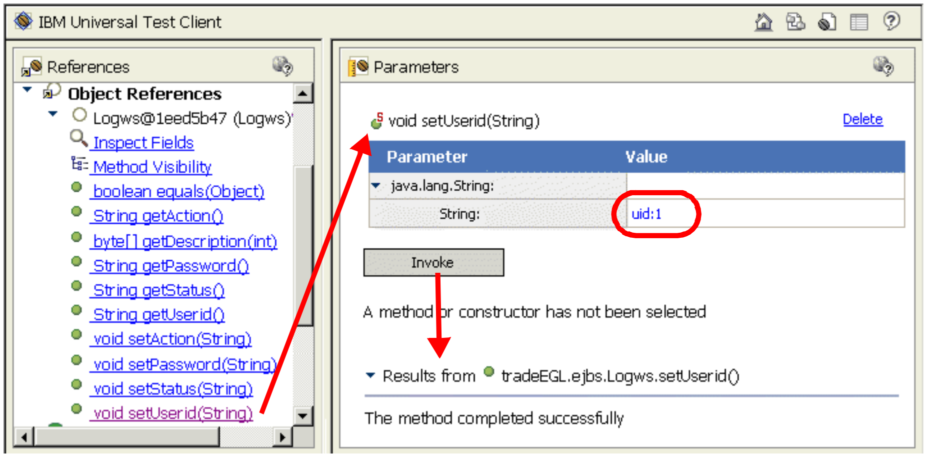
Results from new Logws()

tradeEGL.ejbs.Logws@270063be (tradeEGL.ejbs.Logws)

Figure 9-9 Universal test client creating a JavaBean

- Expand the `Logws` class and click the *`Logws()`* constructor.
- Click *Invoke* and then *Work with Object* and the `Logws` object appears under Object References.

Next, we have to fill the `Logws` record with values (Figure 9-10).



Parameter	Value
java.lang.String:	
String:	uid:1

A method or constructor has not been selected

Results from tradeEGL.ejbs.Logws.setUserid()

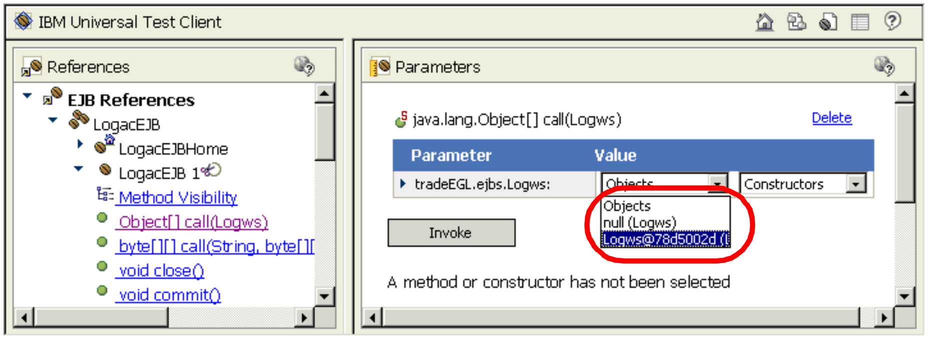
The method completed successfully

Figure 9-10 EJB test client setting values

- Expand the Logws JavaBean under Object References.
- Select the *setUserid* method. Enter uid:1 as String value and click *Invoke*.
- Repeat this for the *setPassword* method with a value of xxx.
- Repeat this for the *setAction* method with a value of inquire.
- You can check the contents by invoking the *getXxxx* methods.
- The Logws working storage object is now ready to be used as a parameter in the call method.

Using an object in a method

Select the call method of the LogacEJB bean again. In the Parameters, select the *Objects* pull-down and select the *Logws* object that was created (Figure 9-11).



A method or constructor has not been selected

Figure 9-11 Universal test client using a JavaBean

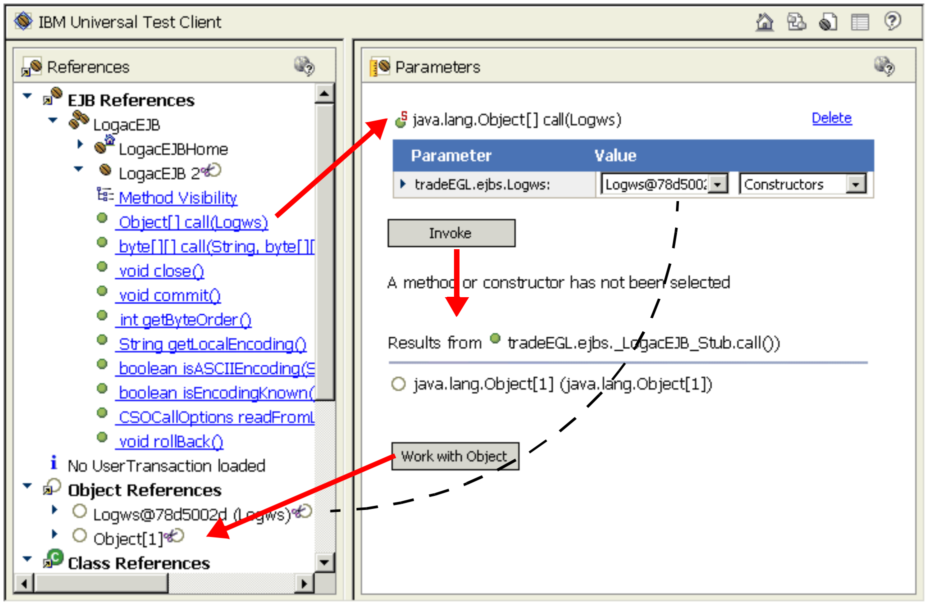
We now have a valid Logws JavaBean that can be used as a parameter for the call method.

Click *Invoke* to run the call method of the session EJB.

Tip: If you get an error message that the CORBA object does not exist, then the session bean instance has expired. Remove the session bean from the EJB References (click the *scissor* icon), then create a new session bean using the create method of the home (Figure 9-6 on page 245).

The call method of the session bean invokes the logac EGL program. The method returns an array of objects with one instance.

Click *Work with Object* to have the result object added under Object References (Figure 9-12).



IBM Universal Test Client

References

EJB References
LogacEJB
LogacEJBHome
LogacEJB 2
Method Visibility
Object[] call(Logws)
byte[][] call(String, byte[][
void close()
void commit()
int getByteOrder()
String getLocalEncoding()
boolean isASCIIEncoding(S
boolean isEncodingKnown(
CSOCallOptions readFromI
void rollBack()
No UserTransaction loaded
Object References
Logws@78d5002d (Logws)
Object[1]
Class References

Parameters

java.lang.Object[] call(Logws) Delete

Parameter	Value	
tradeEGL.ejbs.Logws:	Logws@78d5002	Constructors

Invoke

A method or constructor has not been selected

Results from tradeEGL.ejbs._LogacEJB_Stub.call()

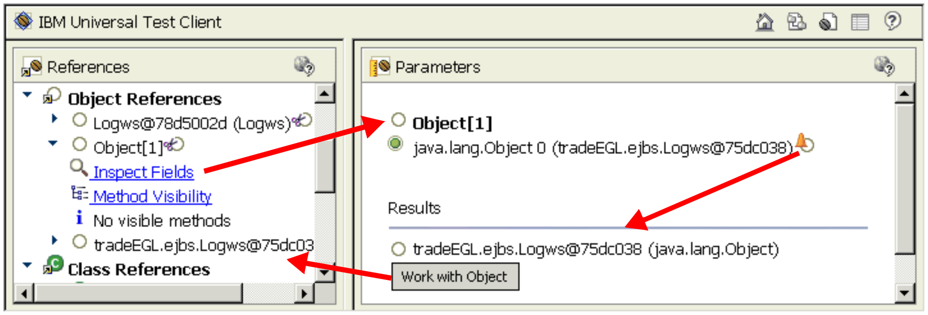
java.lang.Object[1] (java.lang.Object[1])

Work with Object

Figure 9-12 Universal test client invoking the call method

Analyze the result

We have to extract the object from the result array (Figure 9-13).



IBM Universal Test Client

References

Object References
Logws@78d5002d (Logws)
Object[1]
Inspect Fields
Method Visibility
No visible methods
tradeEGL.ejbs.Logws@75dc03
Class References

Parameters

Object[1]
java.lang.Object 0 (tradeEGL.ejbs.Logws@75dc038)

Results

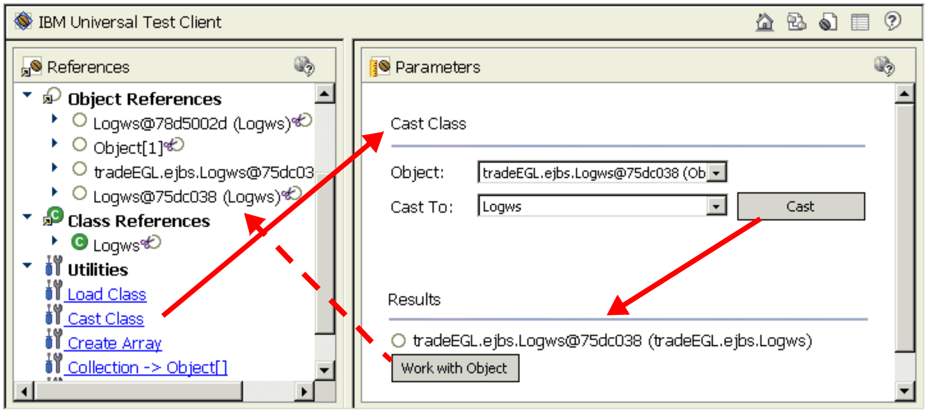
tradeEGL.ejbs.Logws@75dc038 (java.lang.Object)

Work with Object

Figure 9-13 EJB test client result object

- Expand the result *Object[1]* and click *Inspect Fields*.
- Select the icon at the end of the object 🔔 (*Work with Object*) and it appears under Results.
- Click *Work with Object* and a `Logws` object appears under Object References.

To examine the result object, we have to cast it to a real Logws (Figure 9-14).


IBM Universal Test Client

References

- Object References
  - Logws@78d5002d (Logws)
  - Object[1]
  - tradeEGL.ejbs.Logws@75dc03
  - Logws@75dc038 (Logws)
- Class References
  - Logws
- Utilities
  - Load Class
  - Cast Class
  - Create Array
  - Collection -> Object[]

Parameters

Cast Class

Object: tradeEGL.ejbs.Logws@75dc038 (Ob

Cast To: Logws

Cast

Results

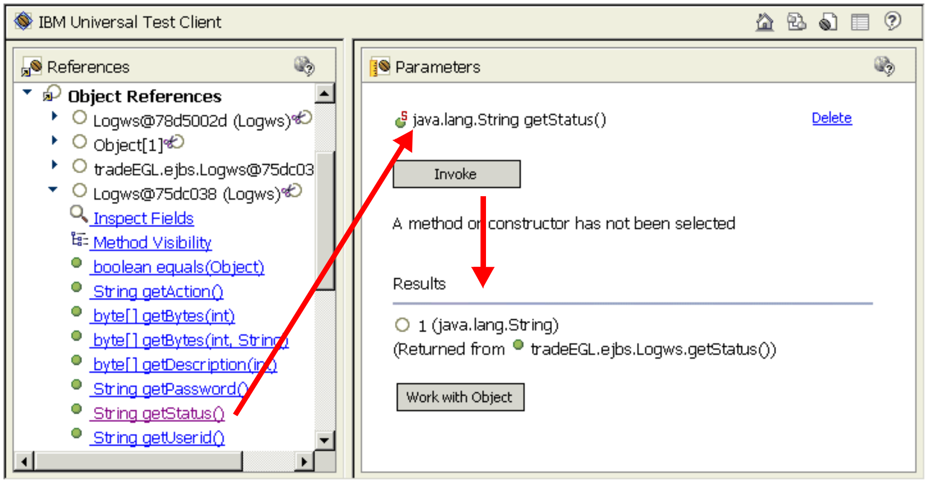
○ tradeEGL.ejbs.Logws@75dc038 (tradeEGL.ejbs.Logws)

Work with Object

Figure 9-14 EJB test client casting a result object

- ► Select *Cast Class* under Utilities.
- ► Select `tradeEGL.ejbs.Logws` in the *Object* pull-down.
- ► Select `Logws` in the *Cast to* pull-down (it is the only one) and click *Cast*.
- ► Click *Work with Object* under Results and a `Logws` object appears under Object References.

Expand the `Logws` object and select the *getStatus* method and click *Invoke*. The result value 1 indicates success (Figure 9-15).


IBM Universal Test Client

References

- Object References
  - Logws@78d5002d (Logws)
  - Object[1]
  - tradeEGL.ejbs.Logws@75dc03
  - Logws@75dc038 (Logws)
    - Inspect Fields
    - Method Visibility
    - boolean equals(Object)
    - String getAction()
    - byte[] getBytes(int)
    - byte[] getBytes(int, String)
    - byte[] getDescription(int)
    - String getPassword()
    - String getStatus()
    - String getUserid()

Parameters

java.lang.String getStatus() Delete

Invoke

A method or constructor has not been selected

Results

○ 1 (java.lang.String)
(Returned from ● tradeEGL.ejbs.Logws.getStatus())

Work with Object

Figure 9-15 EJB test client result status indicates success

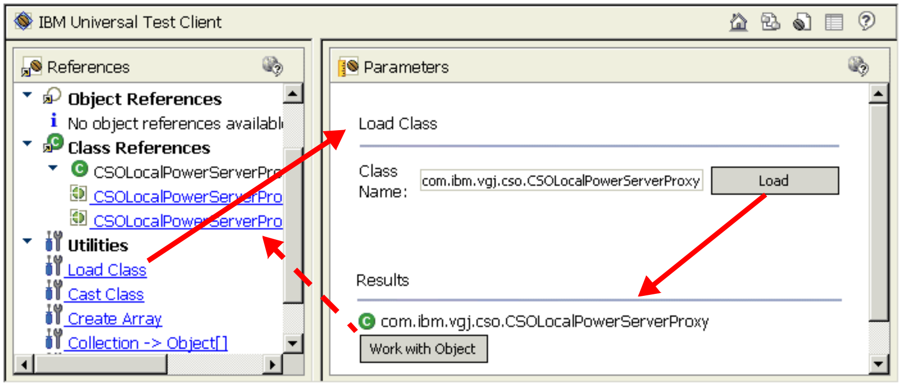
Testing the session EJB with the wrapper class

The test with the session EJB is very cumbersome because the result is an array of objects. This becomes easier with the wrapper class.

Setting up the wrapper class is somewhat complex because we require a unit of work object, `CSOLocalPowerServerProxy`.

Create power server and wrapper objects

In the EJB test client under Utilities, select *Load Class*. Enter a name of `com.ibm.vgj.cso.CSOLocalPowerServerProxy`, click *Load*, and then *Work with Object* under Results (Figure 9-16).



Class Name: com.ibm.vgj.cso.CSOLocalPowerServerProxy

com.ibm.vgj.cso.CSOLocalPowerServerProxy

Figure 9-16 Load the power server unit of work class

Use the constructor to create a `CSOLocalPowerServerProxy` object. Select the `CSOLocalPowerServerProxy()` constructor, click *Invoke*, and click *Work with Object*.

Repeat this sequence for the wrapper class:

- Select *Load Class* under Utilities.
- Enter a name of `tradeEGL.ejbs.LogacWrapper`, click *Load*, and then *Work with Object* under Results.
- Select the constructor `LogacWrapper(CSOPowerServer)` to create a `LogacWrapper` object (Figure 9-17).
- Select the `CSOLocalPowerServerProxy` created earlier in the *Objects* pull-down as parameter.
- Click *Invoke* and *Work with Object*.

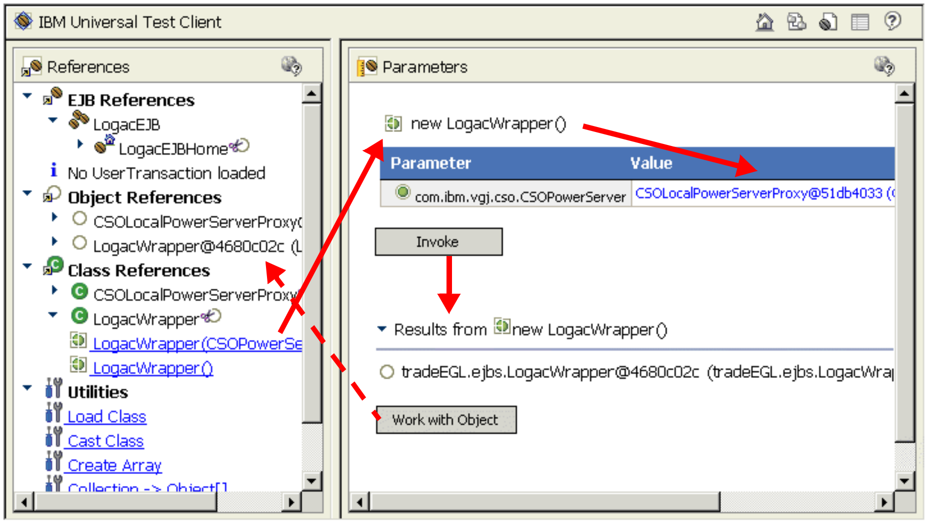


Figure 9-17 Create the wrapper object

You have now two objects under Object References—`LogacWrapper` and `CSOLocalPowerServerProxy`.

Create the working storage record

From the wrapper object, use the *getLogws* method to get a `Logws` working storage record object (Figure 9-18).

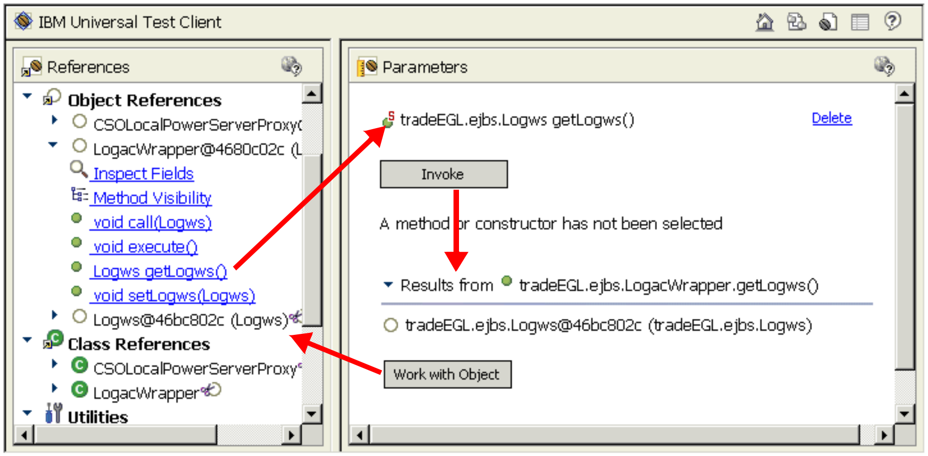
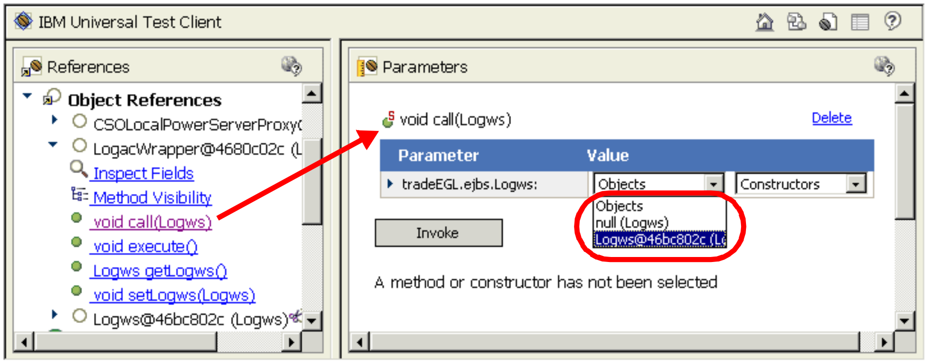


Figure 9-18 Get a Logws record

Use the `setUserid`, `setPassword`, and `setAction` methods of the `Logws` object to set the values to `uid:1`, `xxx`, and `inquire`.

Issuing the call to the session EJB

Select the `call` method of the `LogacWrapper` object to run the EGL program through the session EJB. Use the *Objects* pull-down to select the `Logws` instance (Figure 9-19). Alternatively, use the `execute` method.

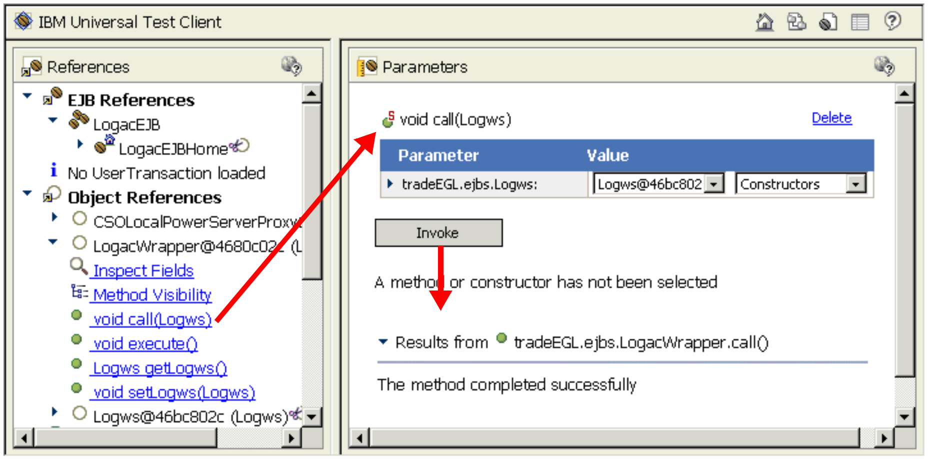


tradeEGL.ejbs.Logws: Objects

A method or constructor has not been selected

Figure 9-19 Preparing the call from the wrapper class

Now click *Invoke* to issue the call (Figure 9-20).



No UserTransaction loaded

tradeEGL.ejbs.Logws: Logws@46bc802

A method or constructor has not been selected

Results from tradeEGL.ejbs.LogacWrapper.call()

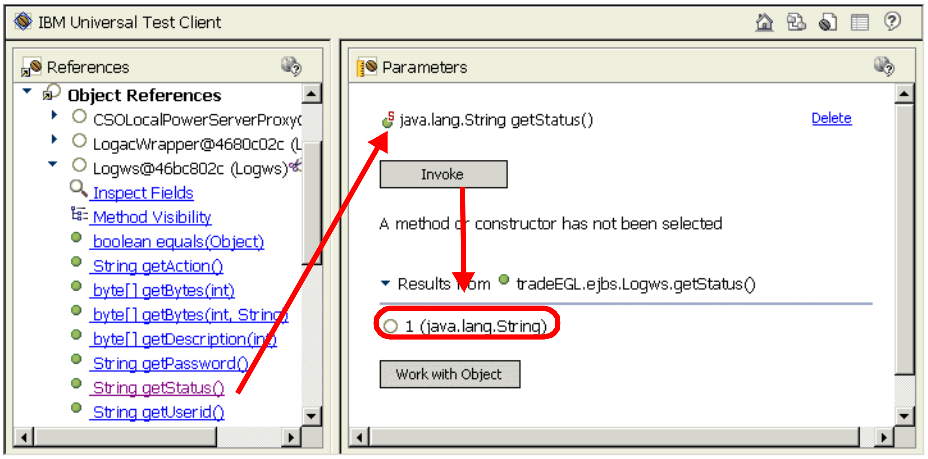
The method completed successfully

Figure 9-20 Issuing the call from the wrapper class

The call method of the wrapper has no result; we have to retrieve the result from the `Logws` object.

Retrieving the result

To retrieve the result, invoke the `getStatus` method in the returned `Logws` object (Figure 9-21).



A method or constructor has not been selected

Figure 9-21 Retrieving the result of the wrapper call

Close the universal test client and stop the server.

Accessing an EJB from a Struts action class

As previously mentioned, a Struts action class represents a part of the controller in the MVC-2 pattern. As such, the action class should interpret messages from the view pages and call the appropriate model objects. Model objects can interact with the EJB through the program wrapper or directly.

Using the program wrapper

The business logic is implemented as EJBs wrapped with a program wrapper, and the action class interacts with the program wrapper. Figure 9-22 shows this interaction in a simplified diagram.

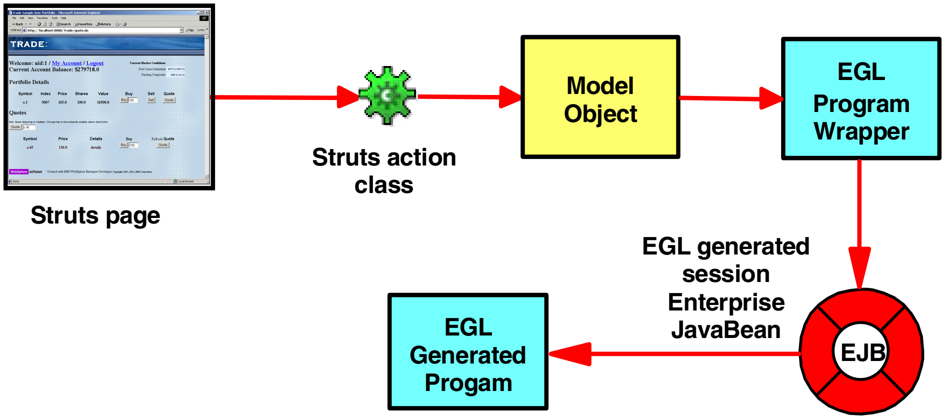


Figure 9-22 Struts to EJB interaction using the program wrapper

We create a model object that makes use of the program wrapper to complete the business logic.

In the `ItsoMyTradeWeb` project under the Java Source folder, there should already be a Java package called `strutsEGL`. This package contains model objects that access EGL Java programs.

Create a `LoginEJB` class in the `strutsEGL` package.

The `LoginEJB` class opens in the Java editor. Make the changes in the `LoginEJB` class as shown in Figure 9-23. Save the `LoginEJB` class.

This is basically the same code as the `Login` model class; however, it uses the `LogacWrapper` and `Logws` classes from the `tradeEGL.ejbs` package of the `ItsoMyTradeEJB` project.

```
package strutsEGL;

import com.ibm.vgj.cso.CSOException;
import com.ibm.vgj.cso.CSOLocalPowerServerProxy;
import com.ibm.vgj.cso.CSOPowerServer;
import tradeEGL.ejbs.LogacWrapper;
import tradeEGL.ejbs.Logws;

public class LoginEJB {
   private CSOPowerServer powerServer;

   public int perform(String userid, String password) {

      try {
         powerServer = new CSOLocalPowerServerProxy();
      } catch (CSOException e) {
         e.printStackTrace();
         return 0;
      }

      // Create an instance of the wrapper class
      LogacWrapper wrapper = new LogacWrapper(powerServer);
      // Set up parameter Logws
      Logws newLogws = wrapper.getLogws();

      try {
         newLogws.setUserid  (userid);
         newLogws.setPassword(password);
         newLogws.setAction  ("inquire");

         // execute the wrapper
         wrapper.execute();
         powerServer.close();

         try { return Integer.parseInt(newLogws.getStatus()); }
         catch (NumberFormatException e) { return 0; }

      } catch (Exception e) {
         System.out.println("LoginEJB Exception: " + e.getMessage());
         e.printStackTrace();
         return 0;
      }
   }
}
```

Figure 9-23   LoginEJB model class for EGL EJB access through a wrapper

Using the session EJB directly

The business logic is implemented as EJBs and the action class can call the session EJB directly. Figure 9-24 shows this interaction in a simplified diagram.

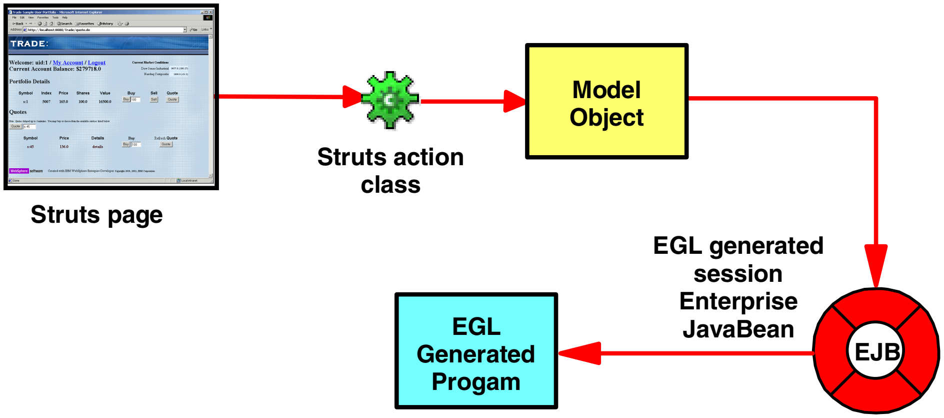


Figure 9-24 Struts to EJB interaction without using the program wrapper

We create a model object that looks up and uses the session EJB to complete the business logic.

Create a `LoginClient` class in the `strutsEGL` package.

The `LoginClient` class opens in the Java editor. Make the changes in the Login class as shown in Figure 9-25.

The logic of this implementation follows this path:

- First the home of the session EJB is acquired.
- A working storage record is setup with the user ID and password parameters.
- An instance of the session bean is created and the call method is invoked.
- The status is extracted from the result working storage record.

Save the `LoginClient` class.

```
import javax.ejb.EJBException;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import tradeEGL.ejbs.*;

public class LoginClient {

   public int perform(String userid, String password){
      LogacEJBHome logacHome;
      LogacEJB     logacEJB;
      Logws        logws;
      Logws        result = null;

      try {
         InitialContext initCtx = new InitialContext();
         Object objref = initCtx.lookup("LogacEJB");
         logacHome  = (LogacEJBHome)PortableRemoteObject.narrow
                         (objref,LogacEJBHome.class);
      } catch (NamingException ex) {
         System.out.println("Logac EJB home failed: " + ex.getMessage());
         ex.printStackTrace();
         return 0;
      }
      logws = new Logws();
      logws.setUserid(userid);
      logws.setPassword(password);
      logws.setAction("inquire");

      try {
         logacEJB = logacHome.create();
         Object resultobj[] = logacEJB.call(logws);
         result = (Logws)( resultobj[0] );
         try { return Integer.parseInt(result.getStatus()); }
         catch (NumberFormatException e) { return 0; }
      } catch (Exception e) {
         System.out.println("Logac EJB call failed: " + e.getMessage());
         e.printStackTrace();
         return 0;
      }
   }
}
```

Figure 9-25 LoginClient model class for EGL EJB direct call

Testing the Struts application with the EJB

To run the trade application with the EJB all we have to do is change the `EGLLoginAction2` class to use the `LoginEJB` or `LoginClient` class instead of the `Login` class.

In the `perform` method, change the code so that the `Login`, `LoginEJB`, or `LoginClient` class is selected, depending on the user ID entered (Figure 9-26).

```
public class EGLLoginAction2 extends Action {

   public ActionForward perform(
      ......
      try {
         // do something here
         int status = 0;
         String userid   = loginForm.getUsername();
         String password = loginForm.getPassword();

         // make EGL call
         // - regular or EJB call depending on userid
         // uid:x   -> normal
         // uid:xx  -> EJB through wrapper
         // uid:xxx -> EJB directly
         if (userid.length() < 6) {
            Login login1 = new Login();
            status = login1.perform(userid, password);
         }
         else if (userid.length() < 7) {
            LoginEJB login2 = new LoginEJB();
            status = login2.perform(userid, password);
         }
         else {
            LoginClient login3 = new LoginClient();
            status = login3.perform(userid, password);
         }

         if (status == 1) {
            request.getSession(true).setAttribute("uidBean", userid);
         } else {
            request.getSession(true).setAttribute("uidBean", "");
            errors.add("login", new ActionError("error.login.failed"));
         }
      } catch ......
```

Figure 9-26 EGLLoginAction2 class calling different actions

Using the welcome page

Redo the test as described in “Testing the Struts application with the EGL action” on page 234:

- Start the server.
- Run the Web application (`index.jsp`) with different user IDs to test the access to the session bean through the wrapper or directly:

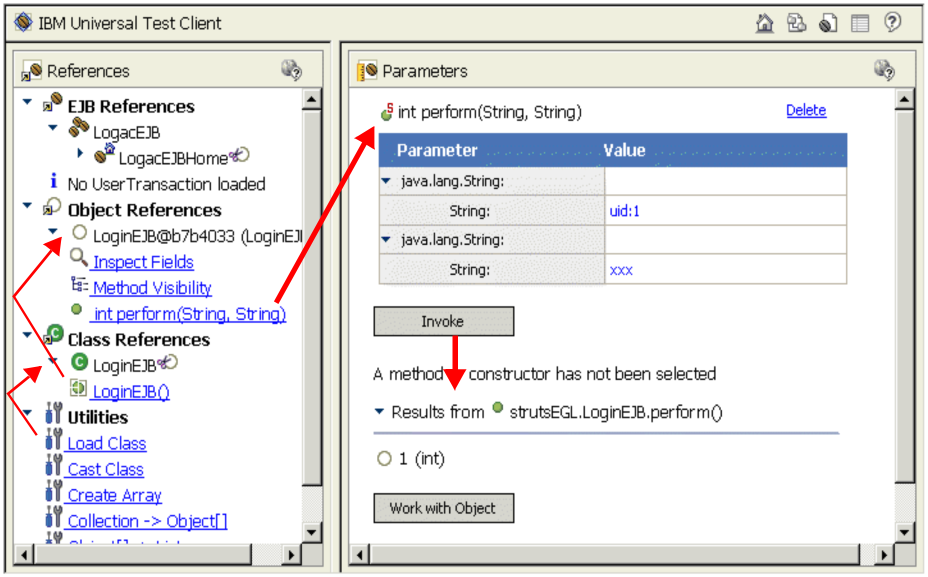
```
uid:1   ==> non-EJB access through wrapper
uid:11  ==> session EJB access through wrapper
uid:111 ==> direct access to session EJB
```

- The output should be identical.

Using the universal test client

You can also use the universal test client to run the EGL part of the Struts application that uses the session bean:

- Use the *Load Class* function to load the `strutsEGL.LoginEJB` class.
- Use the constructor to create an instance of `LoginEJB`.
- Use the `perform` method with parameters `uid:1` and `xxx` to issue the call to the session bean.
- The result is shown in Figure 9-27.



int perform(String, String)

Parameter	Value
java.lang.String:	
String:	uid:1
java.lang.String:	
String:	xxx

A method constructor has not been selected

Results from strutsEGL.LoginEJB.perform()

○ 1 (int)

Figure 9-27 Universal test client using the LoginEJB class

Alternatively use the `LoginClient` class:

- Use the *Load Class* function to load the `strutsEGL.LoginClient` class.
- Use the constructor to create an instance of `LoginClient`.
- Use the `perform` method with parameters `uid:1` and `xxx` to issue the call to the session bean.
- The result is identical to Figure 9-27.

Debugging the Java code

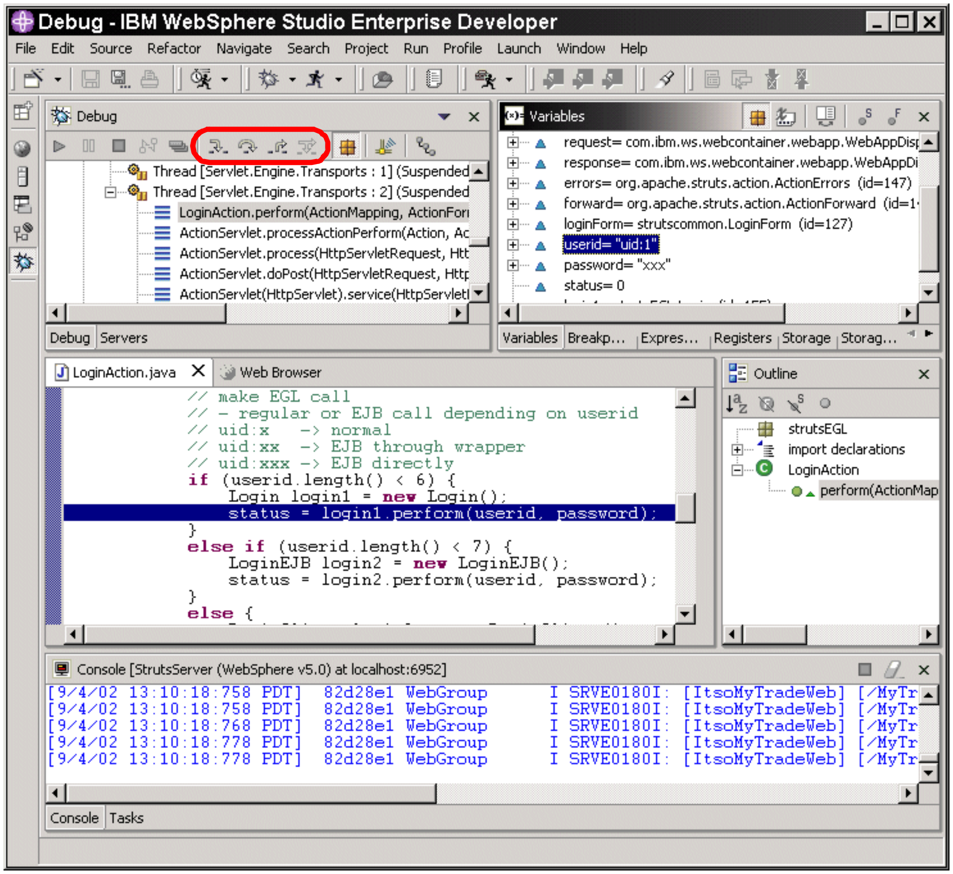
You can set a breakpoint anywhere in your Java source code, including EJB and Java code generated from EGL, to stop execution at that point in the code. To set breakpoints:

- Open the `strutsEGL.LoginAction` class and set a breakpoint in the `perform` method.
- Open the `tradeEGL.ebjs.LogacEJBBean` class and set a breakpoint in the `call` method.
- In the Server perspective, start the server in debug mode.
- The Debug perspective opens. It provides better facilities than debugging in the Server perspective.
- Run the application from the Server or Web perspective by selecting the `index.jsp` and *Run on Server*.
- If you are prompted to step into the `index.jsp`, select *Skip* and *Disable step by step mode*. Click *OK*. Step by step mode can be enabled and disabled in the Debug view using the icon.
- The `index.jsp` displays in the Web browser. Enter **`uid:1`** and **`xxx`** in the Struts EGL form and click *Login*.
- The application stops at the first breakpoint encountered, and the Debug perspective is displayed.

Debug perspective

In the Debug perspective you can see the source code of the program where the breakpoint was encountered (Figure 9-28).

- Use the icons in the Debug view to step through the code.


Debug - IBM WebSphere Studio Enterprise Developer
File Edit Source Refactor Navigate Search Project Run Profile Launch Window Help
Debug
Variables
Debug Servers
Variables Breakp... Expres... Registers Storage Storag...
LoginAction.java
Web Browser
Outline
Console Tasks

```
Thread [Servlet.Engine.Transports : 1] (Suspended
Thread [Servlet.Engine.Transports : 2] (Suspended
    LoginAction.perform(ActionMapping, ActionFor
    ActionServlet.processActionPerform(Action, Ac
    ActionServlet.process(HttpServletRequest, Htt
    ActionServlet.doPost(HttpServletRequest, Http
    ActionServlet(HttpServlet).service(HttpServlet
```

```
request= com.ibm.ws.webcontainer.webapp.WebAppDisp
response= com.ibm.ws.webcontainer.webapp.WebAppDi
errors= org.apache.struts.action.ActionErrors  (id=147)
forward= org.apache.struts.action.ActionForward  (id=1
loginForm= strutscommon.LoginForm  (id=127)
userid= "uid:1"
password= "xxx"
status= 0
```

```
// make EGL call
// - regular or EJB call depending on userid
// uid:x   -> normal
// uid:xx  -> EJB through wrapper
// uid:xxx -> EJB directly
if (userid.length() < 6) {
    Login login1 = new Login();
    status = login1.perform(userid, password);
}
else if (userid.length() < 7) {
    LoginEJB login2 = new LoginEJB();
    status = login2.perform(userid, password);
}
else {
```

```
strutsEGL
import declarations
LoginAction
    perform(ActionMap
```

Console [StrutsServer (WebSphere v5.0) at localhost:6952]

```
[9/4/02 13:10:18:758 PDT]  82d28e1 WebGroup     I SRVE0180I: [ItsoMyTradeWeb] [/MyTr
[9/4/02 13:10:18:758 PDT]  82d28e1 WebGroup     I SRVE0180I: [ItsoMyTradeWeb] [/MyTr
[9/4/02 13:10:18:768 PDT]  82d28e1 WebGroup     I SRVE0180I: [ItsoMyTradeWeb] [/MyTr
[9/4/02 13:10:18:778 PDT]  82d28e1 WebGroup     I SRVE0180I: [ItsoMyTradeWeb] [/MyTr
[9/4/02 13:10:18:778 PDT]  82d28e1 WebGroup     I SRVE0180I: [ItsoMyTradeWeb] [/MyTr
```

Figure 9-28 Debug perspective for debugging Java code

- Use the Variables view to see the values of the currently accessible variables. You can also change the values by selecting *Change Variable Value* from the context menu.
  - Select *Show Type Names* from the context menu and the variable names are displayed with their full name (with package).
  - Select *Show Detail Pane* from the context menu and a subpane opens that displays the value.
  - Select variables or expressions in the source and select *Inspect* or *Display* from the context menu. Inspect opens the Expressions view, and Display opens the Display view, where the result values are shown.
- You can place the cursor over a variable in the source and wait; the variable value or its definition is displayed as a hover pop-up.

Preparation for deployment

Each EJB has a JNDI name that is registered with a name server. For testing, we did not set a JNDI name and the default name `LogacEJB` was used (Figure 9-5 on page 245).

In a real application server environment, JNDI names must be unique. A more appropriate name may be `trade/Logac`. To change the JNDI name for deployment:

- In the J2EE view of the J2EE perspective, open the `ItsoMyTradeEJB` module in the EJB editor.
- On the Beans tab, select the `LogacEJBBean` and on the right side under WebSphere Bindings change the JNDI name from `LogacEJB` to `trade/Logac`.
- Save the changes.

Important: Changing the JNDI name is currently not supported. The CSO middleware is assuming a certain EJB naming convention based on the name of the program being called. This may be allowed in the future.

10

Generating COBOL for z/OS from EGL

In this chapter we look at the process of generating COBOL programs from EGL to run under CICS on z/OS.

We describe the deployment architecture, the prerequisites on z/OS, and the process of generating the COBOL code using a build processor on z/OS.

COBOL generation and deployment architecture

Figure 10-1 shows how the Enterprise Developer generates and deploys the code to z/OS.

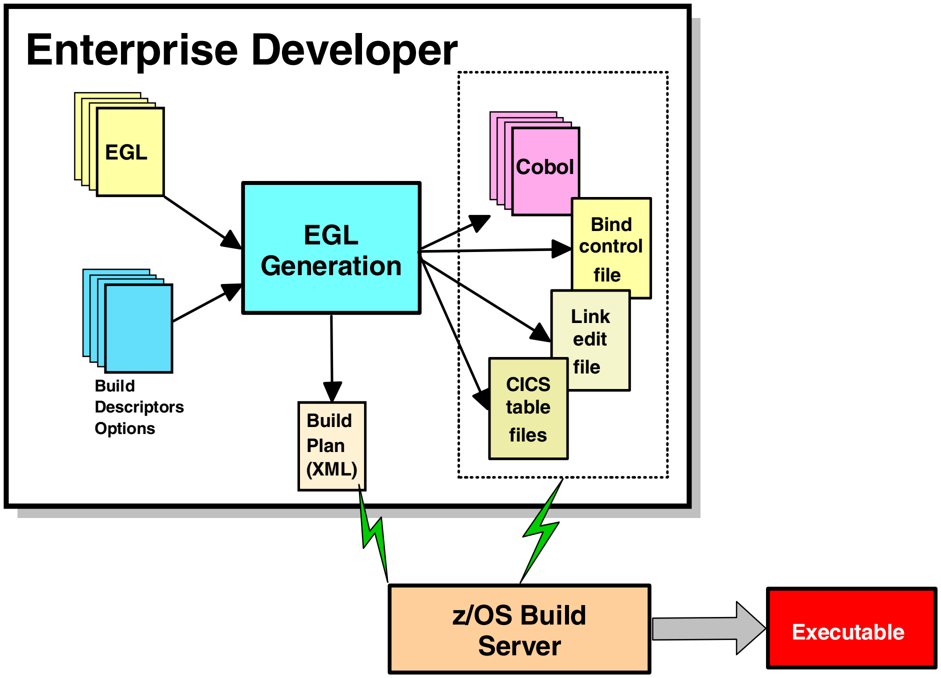


Figure 10-1   EGL generation for z/OS

The sequence of operations is as follows:

- The Enterprise Developer generates the COBOL code as well as the control files required for link edit, bind to relational database, and CICS tables into a directory at the workstation.
- A build plan, which is an XML file, is generated. This build plan provides the commands and instructions to control such operations as DB2 precompile, CICS translation, compile, and link edit.
- Once the build plan is complete, the Enterprise Developer transfers the generated files to the target platform using standard TCP/IP protocols.
- Once the transfer is complete, the z/OS build server is triggered to process the build commands created.

- Results of the build process on z/OS are transmitted back to the originating machine and the Enterprise Developer is notified of the success or failure. Files, such as compile listings, are created at the workstation.

After Enterprise Developer applications are generated, prepared, and stored on the z/OS system, the applications can be run with the support of *Enterprise Developer Server for z/OS* (5655-I57). This product can also be used to run z/OS programs developed with VisualAge Generator Developer.

The run-time library implements data conversion, file and database services, CICS services, error handling, transaction control, and other functions shared among generated applications.

Prerequisites for COBOL generation

Before you start generating COBOL, make sure that the following tasks are completed on the workstation and on z/OS.

Workstation configuration

Make sure that you followed all the installation instructions described in the *Installation Guide*, which is on the first installation CD-ROM (`install.html` or `install.pdf`), particularly:

- The Microsoft Loopback Adapter is installed and configured.
- If your workstation is running Windows 2000, the file `etc\hosts` is modified.

z/OS configuration

The z/OS build server must be configured and started. Details on this task are in *Program Directory for WebSphere Studio Enterprise Developer Options for z/OS*, document number GI10-3242, which is on the installation CD-ROM that is labeled *WebSphere Studio Enterprise Developer for z/OS & FFS (*FMID: HEDS500).

The foreign file system (FFS) server must be configured and started if you use the Enterprise Developer z/OS IDE to prepare COBOL code.

The Enterprise Developer Server (5655-I57) is installed and configured. Note that this program is not shipped with Enterprise Edition CDs and must be ordered for z/OS.

Enterprise Developer Server for z/OS

IBM Enterprise Developer Server for z/OS provides multiple components to support the development and execution of programs when using Enterprise Developer and generating code for z/OS:

- The run-time libraries required by COBOL and Java programs generated using enterprise generation language (EGL).
- A z/OS build server used for building native COBOL programs generated using EGL.
- The sample JCL build scripts used by the build server to build COBOL programs generated using EGL programs.
- The modules necessary for Enterprise Developer to interface with Software Configuration Library Manager (SCLM).
- The modules necessary for COBOL to receive error feedback when doing remote project builds.

IBM Enterprise Developer Server for z/OS (Program number 5655-I57, FMID H284500) provides the modules and materials for EGL and IBM Enterprise Developer Options for z/OS (Program number 5724-B67, FMID HEDS500) provides the remaining components.

For Enterprise Developer Options for z/OS installation and details, see the Program Directory (GI10-3242), which is part of the delivered CDs, and for Enterprise Developer Server for z/OS installation and details, see the Program Directory (GI10-3241).

Allocating z/OS data sets required for EGL COBOL generation

After you have installed and customized the Enterprise Developer Server, each user performing COBOL generations must have some data sets allocated to hold the generated code and the build output.

The CLIST `ELACUSER` in `ELA.V5R0M0.ELACLST` can be used to allocate the required user data sets for COBOL generation. This CLIST can allocate data sets for multiple target environments (MVSCICS, IMSVS, IMSBMP, MVSBATCH, TSO, and OS2CICS).

For EGL generation, we only require the MVSCICS environment and a sample invocation can be issued as:

```
ex 'ELA.V5R0M0.ELACLST(elacuser)' 'MVSCICS(Y)'
```

Table 10-1 shows the data sets that are allocated for EGL generation. Note that some data sets are not used by EGL generation, but could be used by VisualAge Generator, which is also supported by the Enterprise Developer Server.

Table 10-1 Example of required data sets for MVSCICS

z/OS data set name	Purpose
userid.MVSCICS.DBRMLIB	DB2 DBRM
userid.MVSCICS.EZEBIND	DB2 bind
userid.MVSCICS.EZEFOBJ	COBOL generated object code
userid.MVSCICS.EZEJCLP	Used only by VisualAge Generator to hold JCL when generating for batch execution
userid.MVSCICS.EZELINK	COBOL linkage editor
userid.MVSCICS.EZEPCTL	Used only by VisualAge Generator to contain REXX preparation scripts
userid.MVSCICS.EZEPPT	CICS generated entries when specifying `cicsEntries` in the build descriptor
userid.MVSCICS.EZESRC	COBOL source (see "Tailoring the build script to keep the COBOL source" on page 283)
userid.MVSCICS.LOAD	COBOL generated load modules
userid.MVSCICS.OBJECT	COBOL modules for link edit

Configuring control parts for EGL COBOL generation

Now let's configure our Web application for generation of CICS COBOL code for the EGL `logac` program.

Build descriptor for COBOL generation

In the `ItsoMyTradeWeb` project, create a new build descriptor part called `cobolbld` in the `buildDescriptors.eglbld` file and set up these options (select *CICS Target System (All)* as filter):

bind `<bindTemplate>`—Required for DB2. The bind template identifies the bind control part for DB2 access in z/OS. We have to create this part afterwards (see "DB2 bind control" on page 272). For our sample we use `coboldb2`.

cicsEntries `RDO`—Optional for `MVSCICS`. Specifies whether to produce CICS definitions when you generate a COBOL program. It will help the CICS expert to find which entries must be added to the PPT. Options can be `RDO` or `MACRO`.

commentLevel `1`–Specifies the level of EGL comments that are included in the generated COBOL source code. This is useful for catching errors when you have to relate the COBOL code to the EGL source code. The possible values are 0 or 1.

destHost `<z/OS machine name or IP address>`—This is the name of the machine where the target build server is running, for example `carmvs1.raleigh.ibm.com`.

destPassword `<z/OS password>`—Password used for the user ID for processing on z/OS.

destPort `<z/OS build server port number>`—This is the port number of the z/OS build server. This number must match the number specified in the job used to start the z/OS build server (see “Starting the z/OS build server” on page 275), for example `9112`.

destUserID `<z/OS user ID>`—User ID used for processing on z/OS.

genDirectory `<windows directory>`—Specifies the location on the workstation’s file system where Enterprise Edition places generated output and preparation status files, for example `d:\WSEDworkspace\EGLgenout`.

projectID `<z/OS PDS prefix>`—High-level qualifier for z/OS PDS for generated code, usually your user ID. Based on that qualifier, Enterprise Edition will look for the z/OS data sets, such as `userid.MVSCICS.xxxx`. See “Allocating z/OS data sets required for EGL COBOL generation” on page 268 for the required data sets.

system `MVSCICS`—Specifies the target system for the generated source code. Currently the only possible option for COBOL generation is z/OS (MVSCICS). In the future other operating systems will be available.

Additional options if the generated application accesses a DB2 database:

sqlID <user ID>—Specifies a user ID that is used to connect to a DB2 system during generation-time validation of SQL statements.

sqlPassword `<password>`—Specifies a password that is used to connect to a DB2 system during generation-time validation of SQL statements.

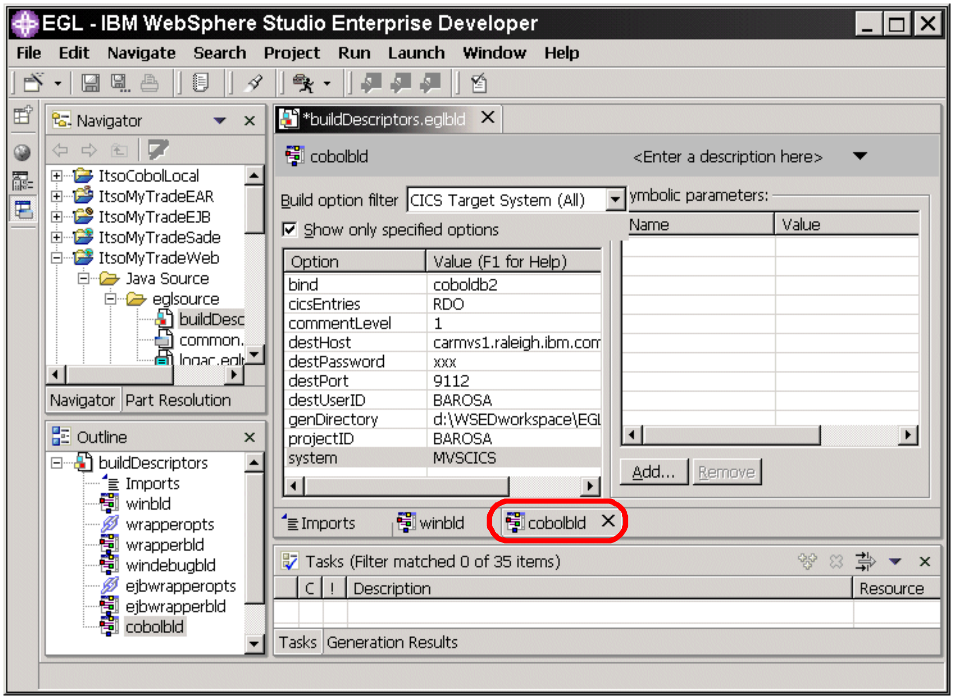
Additional options where the default is usually sufficient:

buildPlan `<YES or NO>`— If set to `YES` (default), specifies that a build plan is generated.

prep `<YES or NO>`—If set to `YES` (default), specifies that, upon successful completion of generation (return code <= 4), preparation of the generated objects is automatically initiated. This will send the source code to the build server for building the run-time objects.

targetNLS `<ENU>`—Specifies the target national language code used for run-time output. Note that the Enterprise Developer Server for the specified language must be installed. This is a very important option in countries where English is not the spoken language, since the messages generated to the end user will be based on this specification. For instance, for Brazil this parameter should be `PTB`. If set to `ENU` (default), the messages will be in US English.

Figure 10-2 shows the `cobolbld` build descriptor part shown in the EGL Part Editor.



Build option filter: CICS Target System (All)

☑ Show only specified options

Option	Value (F1 for Help)
bind	coboldb2
cicsEntries	RDO
commentLevel	1
destHost	carmvs1.raleigh.ibm.com
destPassword	xxx
destPort	9112
destUserID	BAROSA
genDirectory	d:\WSEDworkspace\EGL
projectID	BAROSA
system	MVSCICS

Figure 10-2 Build descriptor for COBOL/CICS generation (cobolbld)

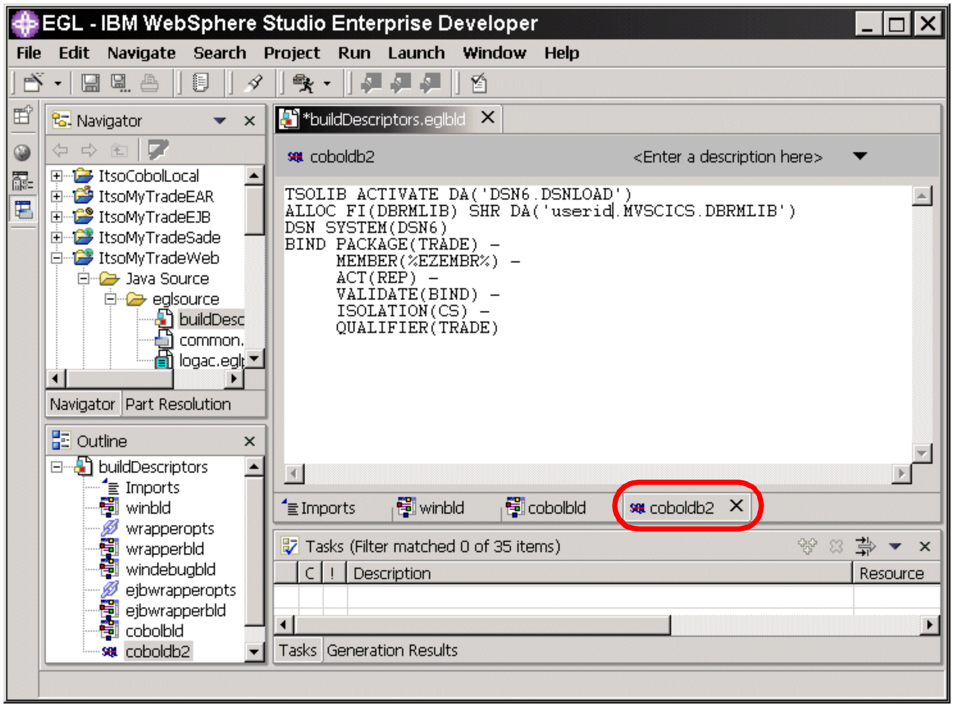
DB2 bind control

In the ItsoMyTradeWeb project, create a new part of type *Bind Control* called coboldb2 in the buildDescriptors.eglbld file and set up the bind options that are required for the DB2 bind operation.

The part is created but is empty. You have to add the statements manually:

```
TSOLIB ACTIVATE DA('DSN6.DSNLOAD')
ALLOC FI(DBRMLIB) SHR DA('userid.MVSCICS.DBRMLIB')
DSN SYSTEM(DSN6)
BIND PACKAGE(TRADE) -
     MEMBER(%EZEMBR%) -
     ACT(REP) -
     VALIDATE(BIND) -
     ISOLATION(CS) -
     QUALIFIER(TRADE)
```

Figure 10-3 shows the DB2 bind control part.



```
TSOLIB ACTIVATE DA('DSN6.DSNLOAD')
ALLOC FI(DBRMLIB) SHR DA('userid.MVSCICS.DBRMLIB')
DSN SYSTEM(DSN6)
BIND PACKAGE(TRADE) -
     MEMBER(%EZEMBR%) -
     ACT(REP) -
     VALIDATE(BIND) -
     ISOLATION(CS) -
     QUALIFIER(TRADE)
```

Figure 10-3 DB2 bind control part

You can use symbolic parameters in the bind control part. For many projects, this may allow you to develop one bind control part that can be used for generation of all SQL programs used in the project. A build descriptor would be used to set any symbolic parameters, such as DSN and user ID.

Creating a Java wrapper build descriptor for COBOL

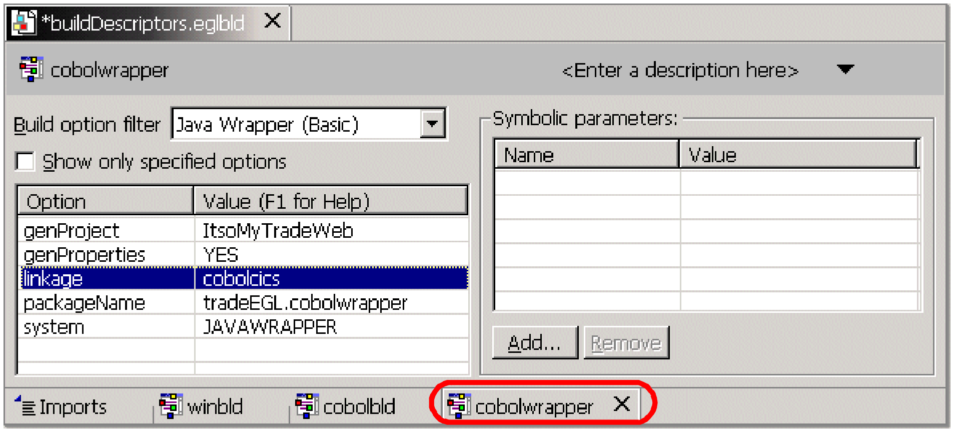
In “Creating a Java wrapper build descriptor” on page 210 you have already created a Java wrapper build descriptor for that will invoke the EGL-generated Java code. Now we will call the generated COBOL program instead of the Java code, so the build descriptor has to use another linkage options part.

We could just simply change the `wrapperbld` that we defined before and change the `linkage` value to point to new linkage options part, but it is better to create a new build descriptor.

In the `ItsoMyTradeWeb` project, create a new part of type *Build Descriptor* called `cobolwrapper` in the `buildDescriptors.eglbld` file and set up the options as you did before. Be sure that you change the `linkage` options pointing to `cobolcics`. We will create the linkage option part right afterwards.

The Java generated code will be created under the project `ItsoMyTradeWeb` in the `tradeEGL.cobolwrapper` package.

Figure 10-4 shows the `cobolwrapper` build descriptor to be used when generating COBOL.



Build option filter: Java Wrapper (Basic)

□ Show only specified options

Option	Value (F1 for Help)
genProject	ItsoMyTradeWeb
genProperties	YES
linkage	cobolcics
packageName	tradeEGL.cobolwrapper
system	JAVAWRAPPER

Name	Value

Figure 10-4 Java wrapper build descriptor used with COBOL generation

Note that you get error messages when you save this part. This is because the `cobolcics` part referenced is not defined yet.

Linkage options for COBOL/MVSCICS

In the ItsoMyTradeWeb project, create a new part of type *Linkage Options* called cobolcics in the buildDescriptors.eglbld file and set up the linkage options that are required for the z/OS COBOL generation with MVSCICS.

Click *Add* to insert a CallLinks element. Overtype the program name with logac, and set the call type to remoteCall.

Select the line entered and set the properties:

pgmName logac—Specifies the name of the program part to which the CallLink element refers.

type remoteCall—Specifies that the call uses EGL middleware, which adds 12 bytes to the end of the data passed. Those bytes allow the caller to receive a return value from the called program.

conversionTable CSOE037—Specifies the name of the conversion table that is used to convert data on a call. CSOE037 is the table for English, another table would be used for other languages.

location eis/ITSOResourceAdapter—Specifies how the location of a called program is determined at run time. Because we are using CICSJ2C as remoteComType, this refers to the JNDI name of the ConnectionFactory object that you establish for the CICS transaction invoked by the call when setting up the J2EE server. By convention, the name of the ConnectionFactory object begins with eis/.

luwControl SERVER—Specifies whether the caller (CLIENT) or called program (SERVER) controls the unit of work. SERVER means that a unit of work started by the called program is independent of any unit of work controlled by the calling program.

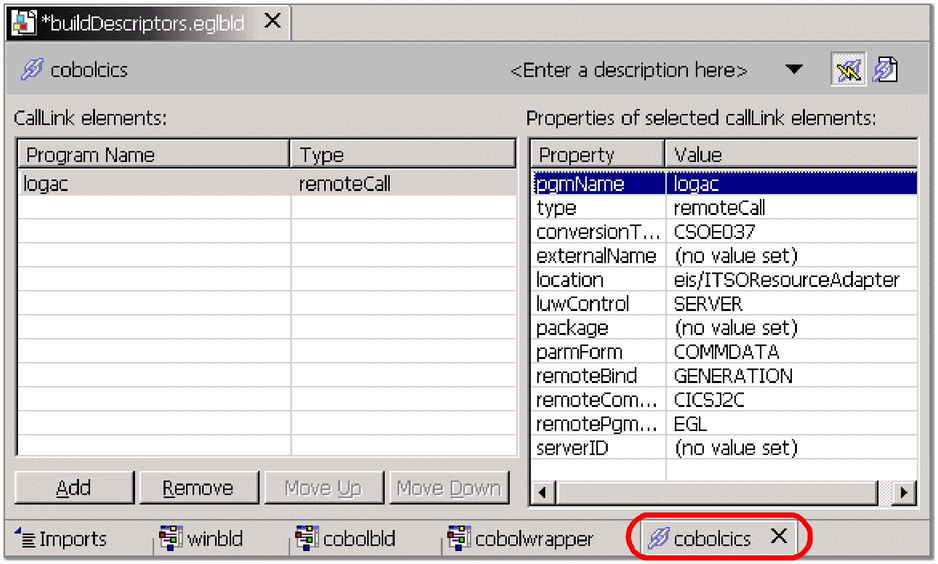
parmForm COMMDATA—Specifies that the caller places business data (rather than pointers to data) in the COMMAREA. Each argument value is moved to the buffer adjoining the previous value without regard for boundary alignment.

remoteBind GENERATION—The properties used for the call are set at generation time.

remoteComType CICSJ2C—Specifies the communication protocol used. WebSphere uses a J2C connector.

remotePgmType EGL—Specifies that the called program is a COBOL program that was generated by Enterprise Developer EGL language.

Figure 10-5 shows the resulting linkage options part.



CallLink elements:

Program Name	Type
logac	remoteCall

Properties of selected callLink elements:

Property	Value
pgmName	logac
type	remoteCall
conversionT...	CSOE037
externalName	(no value set)
location	eis/ITSOResourceAdapter
luwControl	SERVER
package	(no value set)
parmForm	COMMDATA
remoteBind	GENERATION
remoteCom...	CICSJ2C
remotePgm...	EGL
serverID	(no value set)

Figure 10-5 Linkage options for COBOL/MVSCICS generation

Starting the z/OS build server

Before submitting the generation to the z/OS system, be sure that the build server is running.

The instructions to start the z/OS build server are in the document *Program Directory for WebSphere Studio Enterprise Developer Options for z/OS,* Program Number 5724-B67. This document can be found on the CD labeled *WebSphere Studio Enterprise Developer Options for z/OS & FFS*.

The sample z/OS build server startup job looks like this:

```
//JOBNAME   JOB (ACCT#),'TSO ID',CLASS=A
//RUNPGM   EXEC PGM=CCUMAIN,REGION=7400K,
// PARM='-p 9112 -V -a 2 -n 2 -q 10'
//STEPLIB  DD DSN=hlq.CCU.V5R0M0.SCCULOAD,DISP=SHR
//CCUWJCL  DD DISP=SHR,DSN=hlq.CCU.V5R0M0.SCCUSAMP(CCUMVS)
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//CCUBLOG  DD SYSOUT=*
//
```

Note that the parameter *-p* must match the port number specified in the parameter `destPort` of the `cobolbld` build descriptor.

Generating COBOL and Java wrapper from EGL

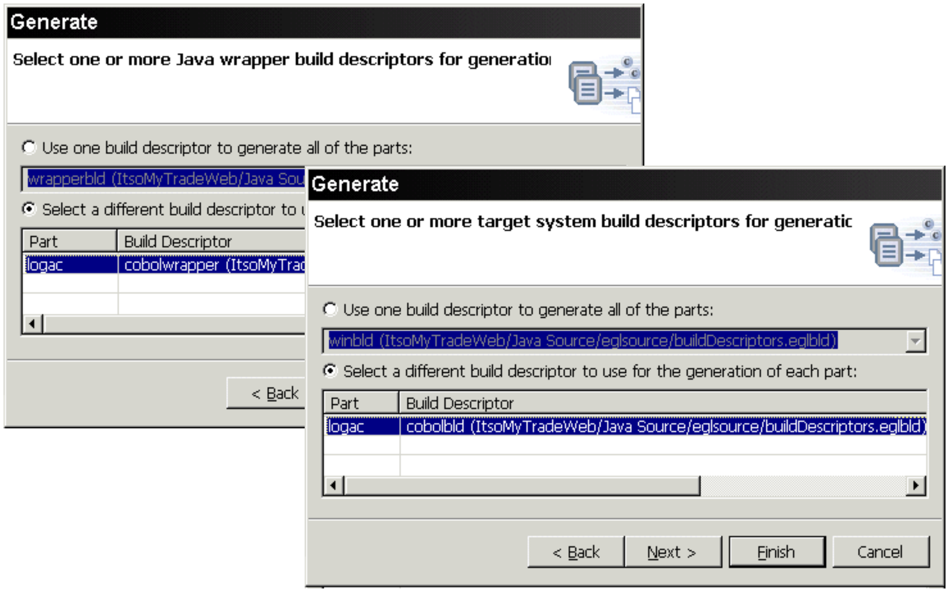
Once the control parts have been configured as described, COBOL code can be generated from the EGL perspective and if the z/OS build server is up the code can be sent to create the executable.

Because the build descriptor is not changing at each generation, its a good idea to have it assigned to a specific default value. That will prevent errors and will keep the necessary descriptor for each component.

> **Tip:** See “Defining the default build descriptors” on page 213 on how to set up the `cobolbld` descriptor as the default build descriptor.

Once all the options are already specified, you can generate the components:

- Select the `eglsource` folder and select *Generate EGL With -> Target System and Java Wrapper Build Descriptor* from the context menu.
- In the dialog box, make sure the check box for `logac` is ticked. Click *Next*.
- For the Java wrapper select the `cobolwrapper` build descriptor and click *Next*. For the target system select the `cobolbld` build descriptor (Figure 10-6).



Generate

Select one or more Java wrapper build descriptors for generatio

○ Use one build descriptor to generate all of the parts:

wrapperbld (ItsoMyTradeWeb/Java Sou

● Select a different build descriptor to u

Part	Build Descriptor
logac	cobolwrapper (ItsoMyTrac

Generate

Select one or more target system build descriptors for generatic

○ Use one build descriptor to generate all of the parts:

winbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)

● Select a different build descriptor to use for the generation of each part:

Part	Build Descriptor
logac	cobolbld (ItsoMyTradeWeb/Java Source/eglsource/buildDescriptors.eglbld)

Figure 10-6 Selecting the build descriptors for COBOL generation

- Click *Next* (you can skip the SQL user ID panel) and click *Finish*.

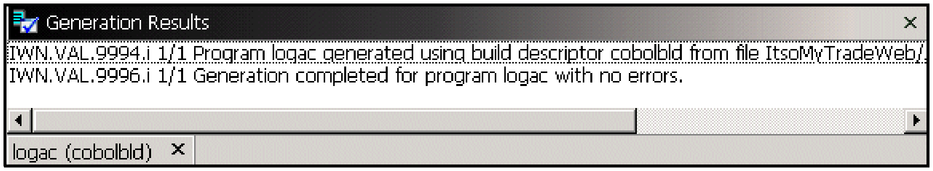
Generation process

Generation is a three-step process:

- The Java wrapper class is generated into the `tradeEGL.cobolwrapper` package.
- The COBOL code and its necessary components are generated into the specified directory:

  `d:\WSEDworkspace\EGLgenout`
- Because we specified `prep=yes` in the `cobolbld`, the code is sent to z/OS and prepared according to a build plan that is generated as well.

The Generation Results view opens with messages as shown in Figure 10-7.



```
IWN.VAL.9994.i 1/1 Program logac generated using build descriptor cobolbld from file ItsoMyTradeWeb/.
IWN.VAL.9996.i 1/1 Generation completed for program logac with no errors.
```

Figure 10-7 COBOL generation results

Generated Java code

In the `ItsoMyTradeWeb` project, the `tradeEGL.cobolwrapper` package has been created with two classes:

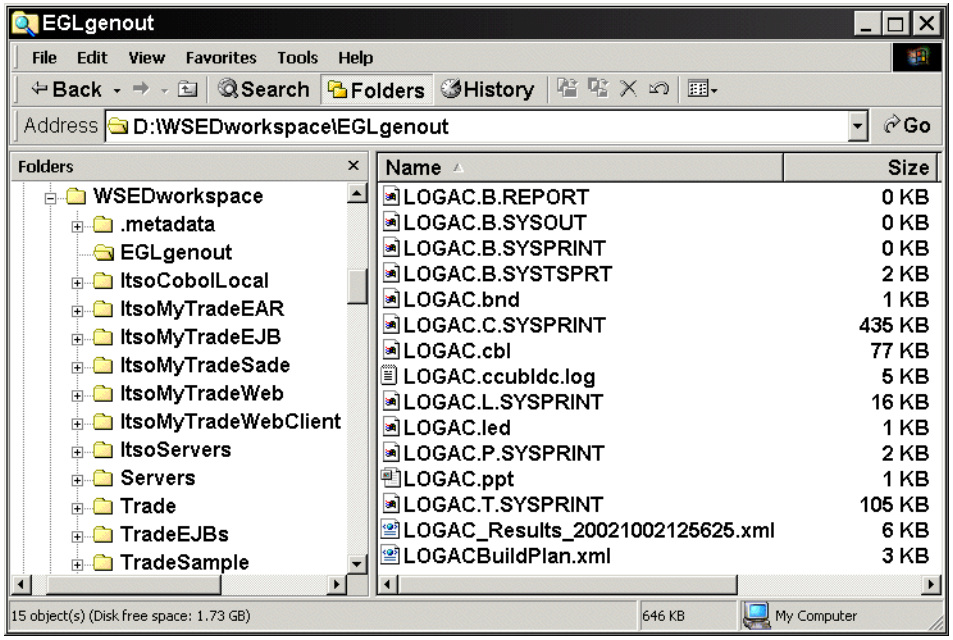
- `LogacWrapper.java`—The Java wrapper class that will invoke the generated COBOL program.
- `Logws.java`—The working storage record used as a parameter when the wrapper is invoked.

We will enhance the Struts application later (“Accessing the EGL-generated COBOL from Struts” on page 300) to invoke the new wrapper class.

Generated local files

A number of files are generated into the local directory specified as the `genDirectory` option in the build descriptor (Figure 10-8):

```
d:\WSEDworkspace\EGLgenout
```



Name	Size
LOGAC.B.REPORT	0 KB
LOGAC.B.SYSOUT	0 KB
LOGAC.B.SYSPRINT	0 KB
LOGAC.B.SYSTSPRT	2 KB
LOGAC.bnd	1 KB
LOGAC.C.SYSPRINT	435 KB
LOGAC.cbl	77 KB
LOGAC.ccubldc.log	5 KB
LOGAC.L.SYSPRINT	16 KB
LOGAC.led	1 KB
LOGAC.P.SYSPRINT	2 KB
LOGAC.ppt	1 KB
LOGAC.T.SYSPRINT	105 KB
LOGAC_Results_20021002125625.xml	6 KB
LOGACBuildPlan.xml	3 KB

15 object(s) (Disk free space: 1.73 GB) 646 KB

Figure 10-8 Files created during COBOL generation

The most important files are:

- `LOGAC_Results_timestamp.xml`—The detailed results of each step with messages about the input and output files.

  We examine the results file in “Build results” on page 282 after we understand all the generated code.

- `LOGACBuildPlan.xml`—The build plan that drives the preparation of the COBOL code on z/OS.

Build plan

The build plan (`LOGACBuildPlan.xml`) has two steps:

1. The `fdaptcl` command invokes the DB2 precompiler, CICS translator, COBOL compiler, and linkage editor.
2. The `fdabind` command invokes the DB2 bind.

Figure 10-9 shows an extract of the build plan.

```
<?xml version="1.0"?>
<buildplan name="Build_LOGAC_20021002125625"
    results="d:\WSEDworkspace\EGLgenout\LOGAC_Results_20021002125625.xml">
   <commands force="ALL">
      <command name="fdaptcl" id="Build1"  buildCondition="LT"
           buildReturnCode="5" prefix="d:\WSEDworkspace\EGLgenout\LOGAC."  >
         <host name="carmvs1.raleigh.ibm.com" port="9112"
               platform="MVSCICS"
            loginID="BAROSA" >
            <codepage client="IBM-850" server="IBM-037" />
         </host>
         <input_files>
            <dir name="d:\WSEDworkspace\EGLgenout">
               <file name="LOGAC.cbl" type="TEXT"/>
            </dir>
         </input_files>
         <dependencies>
            <dir name="d:\WSEDworkspace\EGLgenout">
               <file name="LOGAC.led" type="TEXT"/>
               <file name="LOGAC.ppt" type="TEXT"/>
            </dir>
         </dependencies>
         <env name="CGHLQ" value="BAROSA"/>
          ......
      </command>
      <command name="fdabind" id="Build2"  depends="Build1"
               buildCondition="LT" buildReturnCode="5"
               prefix="d:\WSEDworkspace\EGLgenout\LOGAC."  >
         <host name="carmvs1.raleigh.ibm.com" port="9112"
               platform="MVSCICS"
            loginID="BAROSA" >
            <codepage client="IBM-850" server="IBM-037" />
         </host>
         <input_files>
            <dir name="d:\WSEDworkspace\EGLgenout">
               <file name="LOGAC.bnd" type="TEXT"/>
            </dir>
         </input_files>
         <env ..../>
      </command>
   </commands>
</buildplan>
```

Figure 10-9 Build plan for EGL COBOL generation (extract)

Notice the references to input files and dependencies to other files.

COBOL program and control files

The `LOGAC.cbl` file is the generated COBOL program (Figure 10-10). Notice the comments in **bold**, based on the build descriptor `commentLevel=1 parameter`, which can help locate EGL source in case of execution errors.

```
IDENTIFICATION DIVISION.                                         00001
 PROGRAM-ID.    LOGAC.                                           00002
 ENVIRONMENT DIVISION.                                           00003

 DATA DIVISION.

 WORKING-STORAGE SECTION.

 01  EZEAPP-PROFILE SYNCHRONIZED.
     05 FILLER                   PIC X(8) VALUE "ELARHAPP".
     05 EZEAPP-APPL-NAME         PIC X(8) VALUE "LOGAC".
     05 EZEAPP-PGM-VERSION.
       10 EZEAPP-GEN-DATE        PIC X(8) VALUE "20021001".
       10 EZEAPP-GEN-TIME        PIC X(8) VALUE "10242461".
     05 EZEAPP-RTS-PTR           USAGE IS POINTER VALUE NULL.
     05 EZEAPP-GEN-VERSION       PIC X(16) VALUE "040405".
     05 EZEAPP-COB-SYS           PIC X(8) VALUE "MVSCICS".
 .....
 .....
* 12 *> if (logws.action = "inquire")                            00613
     IF ACTION IN LOGWS = "inquire"
       GO TO EZECONDLBL-1
     END-IF
     GO TO EZECONDLBL-2
     CONTINUE.
 EZECONDLBL-1.
* 13 *> registry-select();
     PERFORM REGISTRY-SELECT
     MOVE "LOGAC" TO EZERTS-PRC-NAME
* 14 *> if (registry is nrf)
     IF EZESTA-REGISTRY-NRF
* 15 *> logws.userid = " ";
       MOVE " " TO EZE6-USERID IN LOGWS
* 16 *> logws.status = "0";
       MOVE "0" TO EZE9-STATUS IN LOGWS
* end;
     END-IF
* 17 *> if (logws.password != registry.password)
     IF EZE7-PASSWORD IN LOGWS NOT = EZE3-PASSWORD IN REGISTRY
* 18 *> logws.status = "0";
       MOVE "0" TO EZE9-STATUS IN LOGWS
* end;
     END-IF
.....
```

Figure 10-10 Generated COBOL source program

The LOGAC.bnd file contains the generated DB2 bind statements from the coboldb2 bind control part (Figure 10-3 on page 272).

The LOGAC.led file contains the linkage editor control statements:

```
INCLUDE OBJLIB(LOGAC)
INCLUDE SELALMD(ELARSINC)
INCLUDE SYSLIB(DFHEAI,DSNCLI)
NAME LOGAC(R)
```

The LOGAC.ppt file contains the CICS program properties table command that can be used to update the CICS resource definitions:

```
* *****************************************************************
* *****************  CICS RDO PROGRAM COMMAND  ******************
* *****************************************************************
*
* A CICS Resource Definition Online (RDO) command  is required
* for each application, map group and table generated by
* VisualAge Generator.  This model entry contains the recommended
* values for this generated part.
*
* PART:               LOGAC
* GENERATION DATE:    20021002
* GENERATION TIME:    125625439

DEF PROG(LOGAC) GROUP(XXXX) L(COBOL) REL(NO) RES(NO) S(ENABLED)

* RES(YES) might provide better performance for frequently used members.
* L(COBOL) parameter should be changed to L(LE370) if you compile
*        the program using COBOL/370.
```

The LOGAC.x.SYSPRINT files contain print output from DB2 precompiler (x=P), CICS translator (x=T), COBOL compiler (x=C), and linkage editor (x=L). The LOGAC.B.SYSTSPRT file contains the print output from DB2 bind and the other LOGAC.B.xxx files are empty files from DB2 bind operation. See "Output of the z/OS build scripts" on page 284 for more details.

Generated z/OS files

The output files of the EGL COBOL generation in z/OS data sets based on the specified build descriptor are:

- userid.MVSCICS.EZEBIND(LOGAC)—The contents of the DB2 bind file LOGAC.bnd.
- userid.MVSCICS.EZEPPT(LOGAC)—The contents of the CICS PPT file LOGAC.ppt. This file is created because cicsEntries=RDO is specified in the build descriptor.

- `userid.MVSCICS.DBRMLIB(LOGAC)`—The DBRM created by the DB2 precompiler.
- `userid.MVSCICS.OBJECT(LOGAC)`—The object module created by the COBOL compiler.
- `userid.MVSCICS.LOAD(LOGAC)`—The load module created by the linkage editor.

Build results

The build results file contains status information for the code preparation steps that were done on z/OS and the files that were produced (Figure 10-11).

```
<?xml version="1.0" encoding="UTF-8" ?>
<buildresults name="Build_LOGAC_20021002125625">
  <commandoutput>
    <command id="Build1" name="fdaptcl" buildCondition="LT"
             buildReturnCode="5" prefix="d:\WSEDworkspace\EGLgenout\LOGAC.">
     ...... command data from build plan ......
    </command>
    <generatedcommand><![CDATA[ccubldc  -h carmvs1.raleigh.ibm.com@9112 -b
fdaptcl -au BAROSA -k IBM-850 -r IBM-037 -P
d:\WSEDworkspace\EGLgenout\LOGAC. -it d:\WSEDworkspace\EGLgenout(LOGAC.cbl
-dt d:\WSEDworkspace\EGLgenout(LOGAC.led
d:\WSEDworkspace\EGLgenout(LOGAC.ppt -c LT -n 5 -v SYSTEM=MVSCICS EZENLS=ENU
EZEPID=BAROSA MBR=LOGAC EZEDATA=31 EZEENV=MVSCICS CGHLQ=BAROSA DATA=31
EZEMBR=logac EZEGDATE=10/02/02 EZEGMBR=logac EZEGTIME=12:56:25 EZESQL=N
EZETRAN=logac ]]></generatedcommand>
<stdout><![CDATA[02/10/02 12:56:25 (c) Copyright, IBM Corp. 2001
                  Copyright (c) 2002 Rational Software Corporation
02/10/02 12:56:36 *** Success ***
......return code of each step......
02/10/02 12:56:36 Message files from build:
02/10/02 12:56:36    1:d:\WSEDworkspace\EGLgenout\LOGAC.P.SYSPRINT
02/10/02 12:56:37    2:d:\WSEDworkspace\EGLgenout\LOGAC.T.SYSPRINT
02/10/02 12:56:38    3:d:\WSEDworkspace\EGLgenout\LOGAC.C.SYSPRINT
02/10/02 12:56:41    4:d:\WSEDworkspace\EGLgenout\LOGAC.L.SYSPRINT
02/10/02 12:56:42 *------------------------------------------------------
]]></stdout>
    <stderr><![CDATA[]]></stderr>
    <returncode>0</returncode>
  </commandoutput>
  ......
</buildresults>
```

Figure 10-11 Build results (extract)

Creating the COBOL executable on z/OS

Run-time objects from COBOL source code for MVSCICS cannot be built on the workstation. The build is always performed on a build server running in z/OS using build scripts.

Build scripts

A build script is a z/OS command file used by a build server to transform one set of files into another. For example, the MVS build server uses a build script written in pseudo-JCL to transform a COBOL source file into an object file and (in some cases) to transform one or more object files into a load module.

The build scripts that are provided by the Enterprise Developer server for z/OS and placed in the `PROCLIB` library of the z/OS build server (allocated by ddname `CCUPROC`) when the build server is installed as a feature of Enterprise Developer are:

fdacl Invokes the COBOL compiler and the linkage editor for generated COBOL source code that does not require the CICS translator or the DB2 preprocessor.

fdatcl Invokes the CICS translator, the COBOL compiler, and the linkage editor for source code generated from an EGL program part that runs in a CICS environment and that does not include SQL.

fdaptcl Invokes the DB2 preprocessor, the CICS translator, the COBOL compiler, and the linkage editor for EGL-generated source code that runs in a CICS environment and that includes SQL**.**

fdabind Binds a generated program to DB2. This build script is used in conjunction with `fdaptcl` for EGL-generated source code that runs in a CICS environment and that includes SQL.

Our build plan used the build scripts `fdaptcl` and `fdabind` (see Figure 10-9 on page 279).

Tailoring the build script to keep the COBOL source

By default, the `fdaptcl` build script does not keep the generated COBOL source in the z/OS data set `userid.MVSCICS.EZESRC`. You can tailor the script to keep the source by removing the comments from these lines:

```
//*UPLOAD EXEC PGM=IEFBR14
//*EZESRC   DD  DSN=&CGHLQ..&SYSTEM..EZESRC,DISP=SHR,CCUEXT=CBL
```

For the other build scripts change the `//*EZESRC` line in the same way.

Output of the z/OS build scripts

In this section we describe the tasks that must be performed to run the example application in the z/OS system and that were performed by the z/OS build script.

DB2 precompiler

The results of the DB2 precompiler are sent back to the Enterprise Edition workstation and can be found in the file `LOGAC.P.SYSPRINT`. An extract of the file is shown here.

```
DB2 SQL PRECOMPILER         VERSION 6 REL. 1.0                      PAGE 1

OPTIONS SPECIFIED: HOST(COB2),APOSTSQL,QUOTE
OPTIONS USED - SPECIFIED OR DEFAULTED
     ......
DB2 SQL PRECOMPILER         MESSAGES                                PAGE 2

DSNH050I I     DSNHMAIN  WARNINGS HAVE BEEN SUPPRESSED DUE TO LACK OF TABLE
DECLARATIONS
DB2 SQL PRECOMPILER         STATISTICS                              PAGE 3

SOURCE STATISTICS
  SOURCE LINES READ: 998
  NUMBER OF SYMBOLS: 231
  SYMBOL TABLE BYTES EXCLUDING ATTRIBUTES: 13688

THERE WERE 1 MESSAGES FOR THIS PROGRAM.
THERE WERE 0 MESSAGES SUPPRESSED BY THE FLAG OPTION.
175480 BYTES OF STORAGE WERE USED BY THE PRECOMPILER.
RETURN CODE IS 0
```

CICS translator

The results of the CICS translator are sent back to the Enterprise Edition workstation and can be found in the file `LOGAC.T.SYSPRINT`. An extract of the file is shown here.

```
CICS 5.3.0 COMMAND LANGUAGE TRANSLATOR   TIME 15.57 DATE 2 OCT 02    PAGE 1

OPTIONS SPECIFIED:-COBOL2,QUOTE,NOSEQ,SP,DBCS
*OPTIONS IN EFFECT*
......
LINE           SOURCE LISTING

00001             IDENTIFICATION DIVISION.                           00001
..... source program ......

NO MESSAGES PRODUCED BY TRANSLATOR.
TRANSLATION TIME:-   0.00 MINS.
```

COBOL compiler

The results of the COBOL compiler are sent back to the Enterprise Edition workstation and can be found in the file LOGAC.C.SYSPRINT. An extract of the file is shown here.

```
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0 Date 10/01/2002
                                                       Time 13:28:04 Page 1

Invocation parameters:
NOSEQ,QUOTE,LIB,RENT,NODYNAM,OPT,DBCS,TRUNC(BIN),NUMPROC(NOPFD),OFFSET,NOLI
ST,MAP,DATA(31)

PROCESS(CBL) statements:
       CBL RENT,RES,NODYNAM,LIB

Options in effect: NOADATA ADV QUOTE ARITH(COMPAT) ZWB
....
PP 5655-G53 IBM Enterprise COBOL for z/OS and OS/390 3.1.0    LOGAC
000001             IDENTIFICATION DIVISION.                     00001
000002             PROGRAM-ID.    LOGAC.                        00002
000003             ENVIRONMENT DIVISION.                        00003
....
1666 IGYOP3091-W Code from "procedure name EZEOVER-ROUTINE" to "GO (line
1684.01)" can never be executed and was therefore discarded.

1904 IGYOP3094-W There may be a loop from the "PERFORM" statement at
"PERFORM (line 1904.01)" to itself. "PERFORM" statement optimization was
not attempted.
Messages Total Informational Warning Error Severe Terminating
Printed:    11        1         10

* Statistics for COBOL program LOGAC:
*    Source records = 1943
*    Data Division statements = 1156
*    Procedure Division statements = 348

End of compilation 1,  program LOGAC,  highest severity 4.
Return code 4
```

Linkage editor

The results of the linkage editor are sent back to the Enterprise Edition workstation and can be found in the file LOGAC.L.SYSPRINT. An extract of the file is shown here.

```
z/OS V1 R3 BINDER     13:28:06 TUESDAY OCTOBER  1, 2002
 BATCH EMULATOR  JOB(BAROSAO ) STEP(RUNPGM  ) PGM= IEWL
 IEW2278I B352 INVOCATION PARAMETERS -
RENT,REUS,LIST,XREF,MAP,AMODE(31),RMODE(ANY)
```

```
 IEW2322I 1220  1     INCLUDE OBJLIB(LOGAC)
 IEW2322I 1220  2     INCLUDE SELALMD(ELARSINC)
 IEW2322I 1220  3     INCLUDE SYSLIB(DFHEAI,DSNCLI)
 IEW2322I 1220  4     NAME LOGAC(R)
 IEW2646W 4B07 ESD RMODE(24) CONFLICTS WITH USER-SPECIFIED RMODE(ANY) FOR
                                SECTION ELARSVCS.
 IEW2646W 4B07 ESD RMODE(24) CONFLICTS WITH USER-SPECIFIED RMODE(ANY) FOR
                                SECTION ELAASADR.

1                         *** M O D U L E  M A P ***

CLASS  B_TEXT LENGTH = 42B8 ATTRIBUTES = CAT, LOAD, RMODE=ANY
...
...
z/OS V1 R3 BINDER     13:28:06 TUESDAY OCTOBER  1, 2002
 BATCH EMULATOR  JOB(BAROSA0 ) STEP(RUNPGM  ) PGM= IEWL
 IEW2008I 0F03 PROCESSING COMPLETED.  RETURN CODE =  4.
```

DB2 bind

The results of the DB2 bind are sent back to the Enterprise Edition workstation and can be found in the file LOGAC.B.SYSTSPRT. An extract of the file is shown here.

```
READY
TSOLIB ACTIVATE DA('DSN6.DSNLOAD')
READY
ALLOC FI(DBRMLIB) SHR DA('BAROSA.MVSCICS.DBRMLIB')
READY
DSN SYSTEM(DSN6)
DSN
BIND PACKAGE(TRADE)     MEMBER(logac)     ACT(REP)     VALIDATE(BIND)
                         ISOLATION(CS)      QUALIFIER(TRADE)
DSNT254I -DSN6 DSNTBCM2 BIND OPTIONS FOR
           PACKAGE = NRARDSN6.TRADE.LOGAC.()
           ACTION        REPLACE
           OWNER         BAROSA
           QUALIFIER     TRADE
           VALIDATE      BIND
           EXPLAIN       NO
           ISOLATION     CS
           RELEASE
           COPY
DSNT255I -DSN6 DSNTBCM2 BIND OPTIONS FOR
           ......
DSNT232I -DSN6 SUCCESSFUL BIND FOR
           PACKAGE = NRARDSN6.TRADE.LOGAC.()
DSN
END
```

Creating a DB2 plan

After having compiled a number of CICS COBOL modules and having built individual DB2 packages, a DB2 plan must be created from the packages.

Run this DB2 command to create a DB2 plan:

```
DSN SYSTEM(DSN6)
BIND PLAN(TRADE) -
    PKLIST(TRADE.*) -
    ACT(REP) -
    VALIDATE(BIND) -
    ISOLATION(CS) -
    QUALIFIER(TRADE)
```

Modifying CICS resource definitions

Additionally, for CICS, you must define your program and transactions to the environment.

The CICS environment uses resource definitions to identify startup parameters, transactions, programs, files, databases, transient data destinations, and system locations for proper operation. You must add to or modify these resource definitions to correctly identify all objects to be used in the new or changed program.

When using CICS tables, the tables are compiled as assembler programs and stored in a run-time library. Some tables can also be maintained through an online facility as described in the resource definition online (RDO) manual for your version of CICS. CICS requires that the online facility be used in place of processing program table (PPT) entries.

Program definition

Either the batch program `DFHCSDUP` utility or the resource definition online command (`CEDA DEFINE PROGRAM`) can be used to define the server program to CICS. Refer to the CICS resource definitions guide for additional information on providing definitions.

One PPT entry is required for each Enterprise Developer generated program. The command generated into the `LOGAC.ppt` file and `userid.MVSCICS.EZEPPT` is:

```
DEF PROG(LOGAC) GROUP(XXXX) L(COBOL) REL(NO) RES(NO) S(ENABLED)
```

Transaction definition

The DB2 plan must be attached to a CICS transaction. In our case a transaction entry of TRDE is used. This transaction name must be specified in the J2C connector factory in the WebSphere Application Server, when we configure the J2C connection specified in "Linkage options for COBOL/MVSCICS" on page 274.

The TRDE transaction invokes the CICS mirror program DFHMIRS and is attached to the DB2 plan TRADE that we generated.

Modifying the CICS startup job

You must include the load library where your generated programs reside in the DFHRPL DD concatenation. Your system administrator included the LE run-time libraries and the Enterprise Developer Server load library in the DFHRPL DD concatenation when the Enterprise Developer Server product was installed.

The CICS startup JCL may have to be modified to add or change allocations for files used by Enterprise Developer programs. These include VSAM files and extrapartition transient data destinations. A sample CICS startup job is shown in "Starting CICS and the CTG" on page 297.

At this point, you are ready to configure the J2C connectors on WebSphere to test the code generated.

11

Implementing CICS actions

In this chapter we describe how CICS transactions can be accessed using the J2C connector architecture.

We then connect the Struts application to the COBOL CICS transaction that was generated in Chapter 10, “Generating COBOL for z/OS from EGL” on page 265.

Accessing CICS transactions

For the purposes of this discussion, the CICS transaction processing environment can be considered to be an enterprise information system (EIS). In this discussion, other examples of EISs include:

- Enterprise resource planning applications such as PeopleSoft, SAP
- Customer relationship management systems such as Siebel
- Legacy applications and computing systems outside of the Java environment, such as a custom-built or vendor business system.
- Database systems not accessible through JDBC

In this section, the following topics are discussed:

- Introduction to connecting to enterprise information systems from a J2EE environment
- Accessing the CICS transaction environment through the J2EE Connection Architecture
- Setting up the CICS resource adapter in Enterprise Developer and in WebSphere Application Server
- Accessing the CICS EIS from an EGL-generated wrapper

For more details on the use of the CICS J2C resource adapters, see the redbook *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401.

Introduction to EIS adapters

As more businesses extend the reach of their applications to their employees, Business Partners, and customers through the use of thin-client Web technologies, they are finding that they need to integrate their existing enterprise information systems (EIS) into the new applications in order for their e-business strategies to be successful. In a J2EE programming model, this means finding some way of accessing heterogeneous enterprise information systems from a Java application server.

There were many different ways of integrating EISs into J2EE applications that were proprietary and not easily portable between J2EE implementations and EISs. J2EE architects integrating EISs required an in-depth knowledge of the EIS they were accessing as well as the transport protocols between the Java application server and the target EIS. Since EIS vendors needed to customize their product for each application server, and application server vendors needed to customize their product to enable connectivity to EISs, applications relying on a particular EIS vendor or application server were less portable.

As can be seen in Figure 11-1, the Enterprise Developer requires knowledge of many EIS adapters. In cases where a single J2EE component needed to access two different EIS systems, it would need to have knowledge of two different EIS adapter APIs.

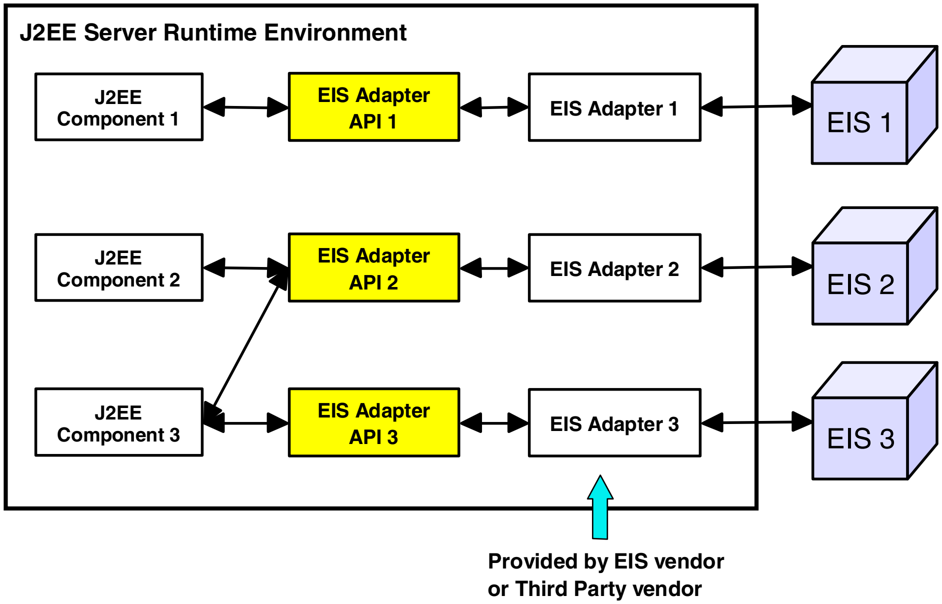


Figure 11-1 J2EE-EIS integration before J2C

IBM provided one architecture for integrating EISs that was called Common Connector Framework (CCF). CCF was delivered with the CICS Transaction Gateway (CTG) and the VisualAge for Java Version 3.02 development environment.

The CCF architecture and programming model provided a level of abstraction between the Java business application and the protocol for accessing the EIS. IBM then provided components for accessing many popular EISs such as CICS, SAP, JD Edwards, PeopleSoft, and Oracle Financials. The CCF architecture allowed developers to concentrate more on the business logic of application and less on the method of access data and transactions in EIS.

J2EE connector architecture (J2C or J2CA)

Recognizing the potential of CCF to help system architects to integrate EISs into business applications, IBM offered the CCF architecture as a Java Specification Request (JSR) for inclusion in the Java platform of open standards. After minor modifications through the Java community process, the CCF architecture was released as J2EE connector architecture (J2C). J2C is now recommended as the strategic method for connecting J2EE applications to EISs instead of CCF. The compete specification for J2C can be found at:

http://java.sun.com/j2ee/download.html#connectorspec

The J2C provides a Java specification that for a standard architecture to integrate heterogeneous EISs into J2EE applications. Because it is an accepted open standard, many vendors are now building adapters that comply with the J2C standard. A list of products and that comply with the standard can be found at the JavaSoft site:

http://java.sun.com/j2ee/connector/products.html

Even though J2C is not part of the J2EE Version 1.2 set of technologies, IBM has provided an implementation of J2C with WebSphere Application Server Version 4.01 as a technology preview. Since Version 4.02 of WebSphere Application Server, the implementation of J2C has been fully supported. J2C has been included in J2EE Version 1.3.

The J2C architecture (Figure 11-2) allows a vendor to provide one EIS-specific interface, or resource adapter, that will plug into any J2EE application server that provides an implementation of the J2C architecture. An application server that provides a J2C implementation can manage several EIS resource adapters to provide a scalable, secure, and transactional environment for application access to services from multiple EISs.

The sections that follow provide an overview of the components of J2C.

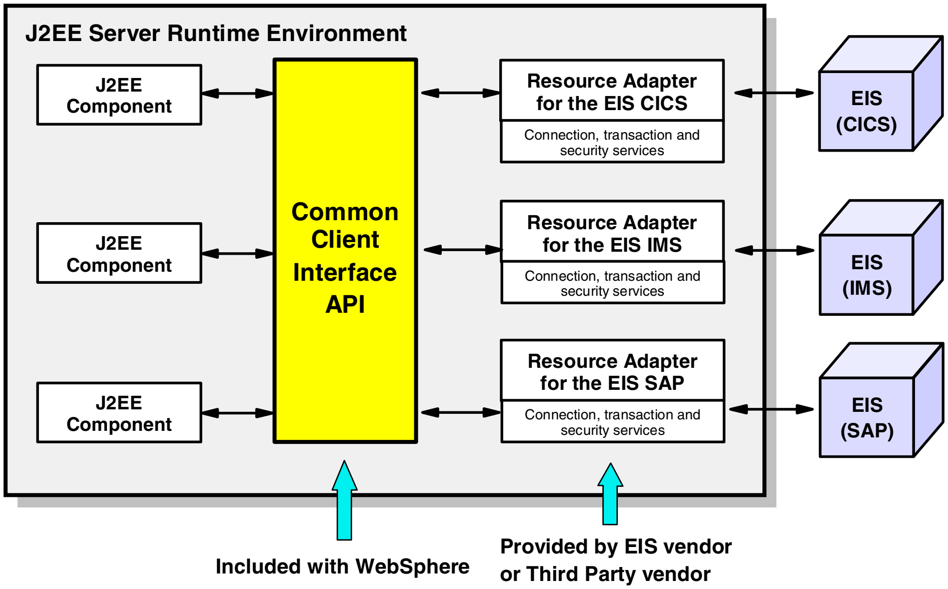

J2EE Server Runtime Environment
J2EE Component
J2EE Component
J2EE Component
Common Client Interface API
Resource Adapter for the EIS CICS
Connection, transaction and security services
Resource Adapter for the EIS IMS
Connection, transaction and security services
Resource Adapter for the EIS SAP
Connection, transaction and security services
EIS (CICS)
EIS (IMS)
EIS (SAP)
Included with WebSphere
Provided by EIS vendor or Third Party vendor

Figure 11-2 J2C architecture

Resource adapter

The J2C architecture provides a set of system-level contracts that define the interface between the EIS and an application server. These contracts define transaction, security and connection API.

A resource adapter provides the EIS-side implementation of these contracts much as a JDBC driver implements the JDBC contract. Because the contract is standard, the resource adapter can plug into any application server that implements J2C. Therefore the resource adapter vendor does not have to customize the adapter for each application server. When the resource adapter is plugged into the application server, it collaborates with the application server to provide the transaction, security and connection management service implementations required by J2C.

Because the transaction, security, and connection management services are provided transparently, the application developer can concentrate on implementing the business logic and functional requirements of the application. This provides faster and easier application development that is scalable, secure, and transactional. Also the portability of the application design between application servers and EIS vendors is assured.

J2C resource adapters are files that have an extension of `.rar`.

System contracts

J2C extends the application server using system contracts. The system contracts provide the services listed below and the resource adapters provide the implementation of these services.

Connection management service

The connection management service provided by the resource adapter allows an application server to provide scalable access to EIS from a large number of clients. The services provides EIS connection pooling capabilities to application servers and allows application components connectivity to the EIS.

Transaction management service

The transaction service allows an application server to provide transactional access to EIS resource managers across multiple resource managers. Transactions that are internal to the EIS resource are also supported without the use of an external transaction manager.

Security management service

The security service allows for a managed and secured access to the EIS resource. The application server can reduce the security threat to an EIS by protecting the EIS from unauthorized access.

Common client interface

The common client interface (CCI) provides a standard Java API for an application component to interact with any EIS. The CCI is intended for enterprise application integration and enterprise tool vendors.

J2C CICS ECI resource adapter

IBM provides two J2C resource adapter for accessing CICS through the CICS Transaction Gateway Version 4.01. The resource adapters are the CICS external call interface (ECI) and the CICS external presentation interface (EPI) J2C resource adapters.

The ECI resource adapter can be used by non-CICS applications to call CICS programs while connected to several CICS servers at the same time. Data is transferred between the two via a COMMAREA as in CICS interprogram communication. The application can call the CICS program either synchronously or asynchronously. ECI calls may also be extended, that is, the application may make several CICS program calls within a single logical unit of work and several logical units of work can be managed if the calls are asynchronous.

The EPI resource adapter allows a non-CICS application to be viewed as a 3270 terminal by a CICS server system to which it is connected. The application can connect to several CICS servers and behave as if it were many 3270 terminals. Data is passed between the application and the CICS program through 3270 data streams and events. The application can then process the data into an appropriate form for its operating environment.

Installation of the CICS ECI resource adapter

The current implementation of EGL for generation of CICS wrapper code is only supported with the ECI type CICS resource adapter. This discussion assumes that you already have access to a CICS Transaction Gateway server (CTG) at Version 4.01 or higher.

CICS Transaction Gateway

The CTG CICS resource adapters, `cicseci.rar` and `cicsepi.rar`, can be found in `<CTG_HOME>\deployable` where CTG_HOME is the directory where the CICS Transaction Gateway is installed. Note that CTG is not delivered as part of Enterprise Developer.

Enterprise Developer

The `cicseci.rar` file is also available in:

```
<WSED-Home>\wstools\eclipse\plugins\com.ibm.etools.ctc.binding.eis_0.5.0\
                                                                   runtime
```

WebSphere Application Server

WebSphere Application Server implements the J2C architecture from Version 4.02 and higher. If the CTG is installed on a different machine, you should get a copy of the resource adapter files on to your WebSphere Application Server machine.

The process of defining installing and configuring the J2C CICS resource adapter in WebSphere Application Server Version 5 is described in “Configuring the J2C connector” on page 341.

Setting up the WebSphere server for CICS J2C calls

You must set up a J2C connection factory in the J2EE server for each CICS transaction accessed through the CICS J2C connector.

If a generated Java wrapper is making the CICS J2C call, you can handle security in any of the following ways (where a wrapper-specified value overrides that of the J2EE server):

- Set the user ID and password in the wrapper's `CSOCallOptions` object
- Set the user ID and password in the `ConnectionFactory` configuration in the J2EE server
- Set up the CICS region so that user authentication is not required

Note: We will set up the J2C connection factory with user ID and password in "Configuring the CICS ECI resource adapter" on page 342.

When calling a program from WebSphere 390, the following restrictions apply:

- If the CallLink element property luwControl is set to `CLIENT`, the call fails. The WebSphere 390 connect implementation does not support an extended unit of work.
- The setting of deployment descriptor property `cso.cicsj2c.timeout` has no effect. By default, timeouts never occur. In the EXCI options table generated by the macro `DFHXCOPT`, however, you can set the parameter `TIMEOUT`, which lets you specify the time that EXCI will wait for an ECI request to complete. A setting of 0 means to wait indefinitely.

Why CICS Transaction Gateway?

CTG is a set of client and server software components that allow a Java application to invoke services in a CICS region. The Java application can be an applet, a servlet, an enterprise bean, or any other Java application.

CTG is required by the CICS ECI resource adapter that acts as a Java client and opens a CTG network connection and sends and receives ECI requests to the CICS region. The classes are supplied in the `ctgclient.jar` file, which is provided by Enterprise Developer.

The latest edition of the CTG is V5.00, and the currently supported platforms are z/OS, OS/390, Linux for S/390, AIX, HP-UX, Sun Solaris, Windows NT, and Windows 2000.

When using z/OS systems, CTG can run on z/OS or a distributed platform. The examples described here and used in our installation is based on the fact that the CTG is installed on the z/OS.

On z/OS, the external CICS interface (EXCI) provides access to COMMAREA-based CICS programs (Figure 11-3). There are few differences between the ECI support on z/OS and the ECI support on distributed platforms.

Note that the user ID and password flowed on ECI requests are verified within the CTG with RACF; afterwards the verified user ID is then flowed to CICS.

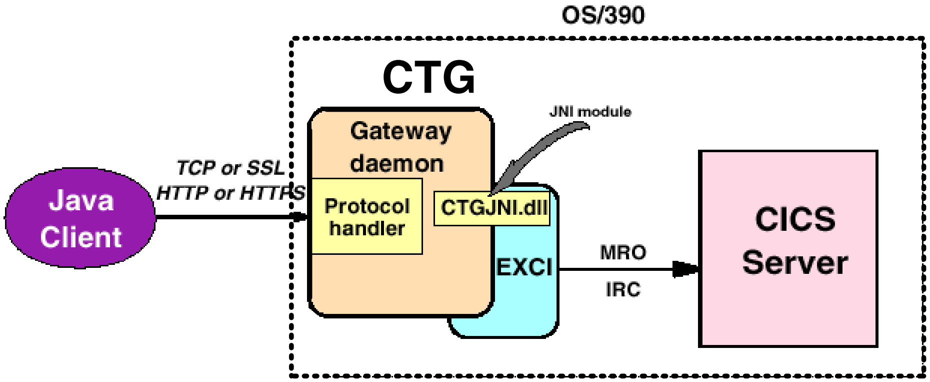


Figure 11-3 CICS Transaction Gateway z/OS

For product information on using CTG, refer to the CICS Transaction Gateway Administration Guides. For information on configuring the CTG, refer to the redbook *CICS Transaction Gateway V5 The WebSphere Connector for CICS,* SG24-6133.

Starting CICS and the CTG

This topic assumes that you have sucessfully installed and configured the CICS Transaction Gateway and CICS.

Before testing the application, be sure that those products are running under z/OS.

Example 11-1 shows a sample CICS startup job. The elements marked in **bold** are used by the J2C definitions. Also note in ***bold italic*** the data sets that are required by Enterprise Developer Server, including the data set of the load module of the EGL generation.

Example 11-1 Sample CICS startup job

```
//NRACSP2  JOB  (3A0195,NONE,,,N),CLASS=O,NOTIFY=&SYSUID,
//              MSGCLASS=T,MSGLEVEL=(1,1),REGION=0M
//CICS     EXEC PGM=DFHSIP,PARM=SYSIN,REGION=0M,PERFORM=12,TIME=1440
//SYSIN    DD   *
```

```
SIT=6$,           SUFFIX OF CICS SYSTEM INIT TABLE
EDSALIM=120M,
APPLID=NRACSP2,   NAME OF THIS CICS REGION
GRPLIST=CSPLIST,  LIST OF GROUPS TO BE USED AT REGION STARTUP
START=INITIAL,    SPECIFICES THE TYPE OF CICS START TO BE PERFORMED
GMTEXT='VGEN 1.2 ON TS 1.3  STARTED BY BAROSA - CSP2',
DCT=NO,           DCT DEFINED USING RDO (NO STATIC TABLES)
FCT=NO,           FCT DEFINED USING RDO (NO STATIC TABLES)
PLTPI=PI,         TELL CICS TO USE DFHPLTPI FOR START UP
PLTSD=SD,         TELL CICS TO USE DFHPLTSD FOR SHUT DOWN
DFLTUSER=TK4CICS, DEFAULT USERID FOR CICS AUTHORITY IF NOT LOGGED ON
STGPROT=YES,      USE STORAGE PROTECTION?
SEC=YES,          REQUIRES LOGON TO CICS SESSION
CSDACC=READWRITE, ALLOW ONLINE UPDATE OF CSD
CSDBKUP=STATIC,   BACKUPTYPE OF CSD (STATIC OR DYNAMIC
CSDBUFND=10,      NUMBER OF DATA BUFFERS FOR THE CSD
CSDBUFNI=11,      NUMBER OF INDEX BUFFERS FOR THE CSD
CSDDISP=SHR,      CSD DISPOSITION FOR DYNAMIC ALLOCATION
CSDDSN=VGEN.NRACSP2.CTS130.DFHCSD, DSN OF CSD BEING USED
CSDFRLOG=NO,      JOURNAL ID. FOR CSD FORWARD RECOVERY
CSDJID=NO,        JOURNAL ID. FOR CSD AUTOMATIC JOURNALING
CSDLSRNO=1,       THE VSAM LSR POOL NUMBER FOR THE CSD
CSDRECOV=NONE,    CSD RECOVERABLE FILE OPTION
CSDSTRNO=200,     CSD NUMBER OF STRINGS
DB2CONN=YES,      AUTO CONNECT TO DB2?
XCMD=NO,          USE DEFAULT NAME FOR RACF CHECK?
XDB2=NO,          USE DEFAULT NAME FOR RACF CHECK?
XDCT=NO,          USE DEFAULT NAME FOR RACF CHECK?
XFCT=NO,          USE DEFAULT NAME FOR RACF CHECK?
XJCT=NO,          USE DEFAULT NAME FOR RACF CHECK?
XPCT=NO,          USE DEFAULT NAME FOR RACF CHECK?
XPPT=NO,          USE DEFAULT NAME FOR RACF CHECK?
XPSB=NO,          USE DEFAULT NAME FOR RACF CHECK?
XRF=NO,           EXTENDED RECOVERY FEATURE (XRF) OPTION
XTRAN=YES,        USE DEFAULT NAME FOR RACF CHECK?
XTST=NO,          USE DEFAULT NAME FOR RACF CHECK?
XUSER=NO,         USE DEFAULT NAME FOR RACF CHECK?
ISC=YES,          INTERSYSTEM COMMUNICATION OPTION
SPOOL=YES,        SYSTEM SPOOLING INTERFACE OPTION
PDIR=NO,          DL/I PSB DIRECTORY OPTION/SUFFIX
IRCSTRT=YES,      INTERREGION COMMUNICATION START
MQCONN=YES,       AUTO CONNECT TO MQSERIES
INITPARM=(DFHD2INI='DSN7',CSQCPARM='SN=SVAG,TN=001,IQ=NRACSP2.INITQ'),
.END
//STEPLIB  DD   DISP=SHR,DSN=CICS.CTS130.PDO0152.SDFHAUTH
.....
//         DD   DISP=SHR,DSN=DSN6.DSNLOAD
//         DD   DISP=SHR,DSN=VGEN.DSN6.RCTLIB
.....
```

```
//         DD   DISP=SHR,DSN=BAROSA.MVSCICS.LOAD
//*
//DFHRPL   DD   DISP=SHR,DSN=VGEN.HS.V1R2M0.NEW.SELALMD
//         DD   DISP=SHR,DSN=VGEN.CTS130.LOADLIB
//         DD   DISP=SHR,DSN=CICS.CTS130.PD00152.SDFHLOAD
//         DD   DISP=SHR,DSN=DSN6.DSNLOAD
//         DD   DISP=SHR,DSN=BAROSA.MVSCICS.LOAD          <=== load module
...
//*
//DFHAUXT  DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHAUXT
//DFHDMPA  DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHDMPA
//DFHDMPB  DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHDMPB
//SYSUDUMP DD   SYSOUT=*
//*
//DFHINTRA DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHINTRA
//DFHTEMP  DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHTEMP
//DFHGCD   DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHGCD
//DFHLCD   DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHLCD
//DFHLRQ   DD   DISP=SHR,DSN=VGEN.NRACSP2.CTS130.DFHLRQ
....
//
```

Example 11-2 shows a sample CTG startup job. Note the CTG port number (22002 in **bold**) that must match the definitions created in Figure 11-11 on page 307.

Example 11-2 Sample CTG startup job

```
//CTGCSP2  JOB (123456,C463,062,,,N),VGUSER,NOTIFY=VGUSER,
//   MSGCLASS=H,CLASS=A,MSGLEVEL=(1,1),REGION=0M
//OEEXCI EXEC PGM=BPXBATCH,
//    PARM='sh /u/ctg402/ctg/bin/ctgstart -port 22002 -noinput'
//STDIN DD PATH='/u/ctg402/ctg/null',
//      PATHOPTS=(ORDONLY)
//STDOUT DD PATH='/u/ctg402/ctg/logs/ctgcsp2o.log',
//       PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
//STDERR DD PATH='/u/ctg402/ctg/logs/ctgcsp2e.log',
//       PATHOPTS=(OWRONLY,OCREAT),PATHMODE=SIRWXU
//STDENV DD *
DFHJVPIPE=JAVACTG
DFHJVSYSTEM_00=NRACSP2-CTG LISTENER FOR NRACSP2 SERVER
/*
//
```

Accessing the EGL-generated COBOL from Struts

The current Struts Web application is not configured to use the Java wrapper created in “Generating COBOL and Java wrapper from EGL” on page 276.

We could write a new Struts action class, but we use the same approach as for the EGL session bean (see “Accessing an EJB from a Struts action class” on page 254) and extend the `EGLLoginAction2` class to call the generated EGL COBOL program.

Using the Java program wrapper to COBOL

The business logic is already implemented in the EGL COBOL program. We have to extend the Struts model object to use the EGL program through the Java wrapper. Figure 11-4 shows the interaction in a simplified diagram.

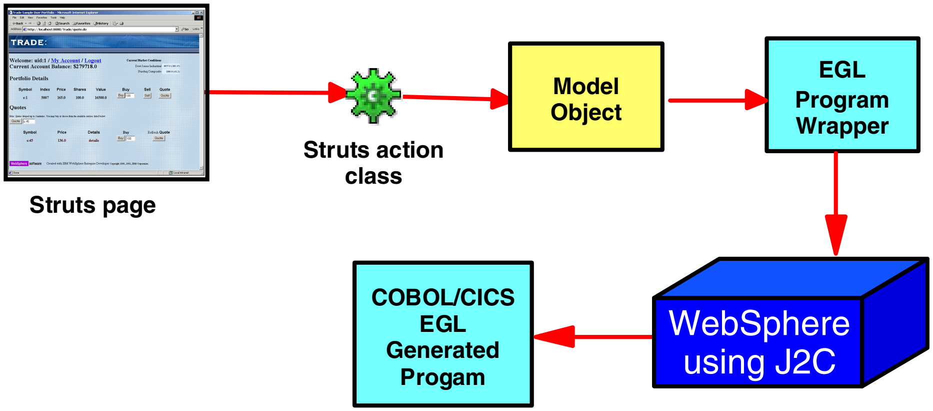


Figure 11-4   Struts to EGL COBOL using a program wrapper

In the `ItsoMyTradeWeb` project under the Java Source folder, there should already be a Java package called `strutsEGL`. This package contains model objects that access EGL Java programs. We create a new model object called `LoginCOBOL`:

- Create a `LoginCOBOL` class in the `strutsEGL` package.
- The `LoginCOBOL` class opens in the Java editor. Make the changes in the `LoginCOBOL` class as shown in Figure 11-2 and save it.

This is basically the same code as the `Login` model class. However, it uses the `LogacWrapper` and `Logws` classes from the `tradeEGL.cobolwrapper` package of the `ItsoMyTradeWeb` project.

```
package strutsEGL;
import com.ibm.vgj.cso.CSOException;
import com.ibm.vgj.cso.CSOLocalPowerServerProxy;
import com.ibm.vgj.cso.CSOPowerServer;

import tradeEGL.cobolwrapper.LogacWrapper;
import tradeEGL.cobolwrapper.Logws;

public class LoginCOBOL {
      private CSOPowerServer powerServer;

   public int perform(String userid, String password) {

      try {
         powerServer = new CSOLocalPowerServerProxy();
      } catch (CSOException e) {
         e.printStackTrace();
         return 0;
      }
         // Create an instance of the wrapper class
      LogacWrapper wrapper = new LogacWrapper(powerServer);

      // Set up parameter Logws
      Logws newLogws = wrapper.getLogws();

      try {

         newLogws.setUserid  (userid);
         newLogws.setPassword(password);
         newLogws.setAction  ("inquire");

         // execute the wrapper
         wrapper.execute();

         try { return Integer.parseInt(newLogws.getStatus()); }
         catch (NumberFormatException e) { return 0; }

      } catch (Exception e) {

         System.out.println("LoginCOBOL Exception: " + e.getMessage());
         e.printStackTrace();
         return 0;
      }
   }
}
```

Figure 11-5 LoginCOBOL model class for COBOL access through a wrapper

Modifying the Struts action to access COBOL

To run the trade application accessing COBOL through the `LoginCOBOL` class, we have to change the `EGLLoginAction2` class. This class already uses the `Login`, `LoginEJB` and `LoginClient` classes. See "Testing the Struts application with the EJB" on page 259 for details of this implementation.

In the `perform` method, change the code so that the `Login`, `LoginEJB`, `LoginClient`, or `LoginCOBOL` class is selected, depending on the user ID entered:

```
uid:1   - uid:9        Java wrapper -> EGL program
uid:10  - uid:99       Java wrapper -> Session EJB -> EGL program
uid:100 - uid:199      Session EJB using JNDI -> EGL program
uid:200 - uid:499      Java wrapper -> J2C connector -> EGL COBOL
```

The change of the `perform` method is shown in Figure 11-6.

```
public class EGLLoginAction2 extends Action {
       ......
   public ActionForward perform(
      try {
         ......
         // make EGL call
         // - regular or EJB call or COBOL call depending on userid
         // uid:x   -> normal
         // uid:xx  -> EJB through wrapper
         // uid:xxx -> EJB directly
         // uid:200+ -> EGL COBOL
         if (userid.length() < 6) {
            Login ...
         }
         else if (userid.length() < 7) {
            LoginEJB ...
         }
         else if ( userid.compareTo("uid:200") < 0 ) {
            LoginClient ...
         }
         else {
            LoginCOBOL login4 = new LoginCOBOL();
            status = login4.perform(userid, password);
         }
         if (status == 1) {
            ......
```

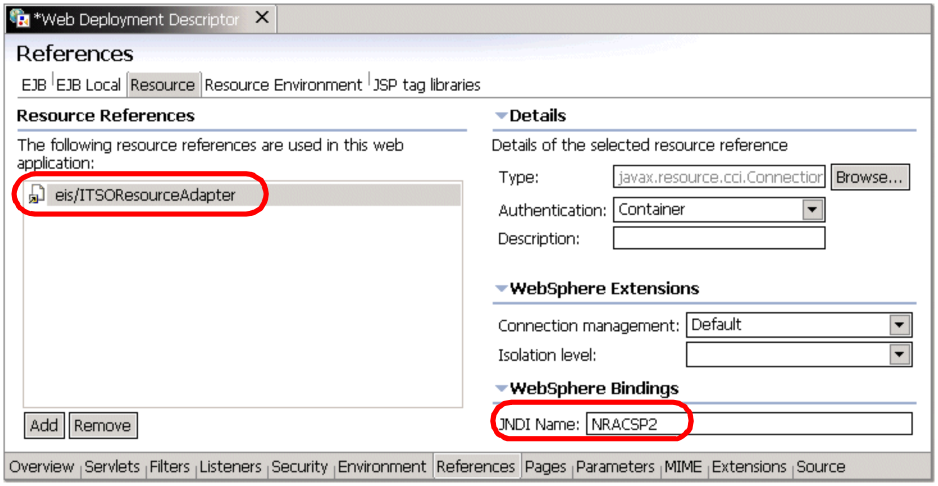
Figure 11-6 Changing the Struts action class to invoke EGL COBOL

Configuring the Web application for J2C

The Java wrapper for the EGL COBOL program uses the J2C connector to access CICS. We defined the location value of `eis/ITSOResourceAdapter` in the linkage options (see “Linkage options for COBOL/MVSCICS” on page 274). This value is a local JNDI name that we have to map to a global JNDI name.

Open the `web.xml` deployment descriptor of the `ItsoMyTradeWeb` project (in `Web Content\WEB-INF`):

- Select the *References* page (at the bottom) and then the *Resources* tab (at the top).
- Click *Add* to define a new resource reference.
- Overtype the generated *(new ResourceRef)* with `eis/ITSOResourceAdapter`.
- For Type, click *Browse*, enter `connectionfactory`, and select the `javax.recource.cci.Connectionfactory`.
- For Authentication, select *Container*, and for Connection management, select *Default*. We will have to set up the user ID and password authentication for the container when we configure the J2C connector in WebSphere.
- For the JNDI name (global name) enter any unique name; we use the *eis* prefix and the name of the CICS server (`eis/NRACSP2`).
- Save the deployment descriptor.
- Figure 11-7 shows the completed dialog.



References

Resource References

The following resource references are used in this web application:

eis/ITSOResourceAdapter

Details

Details of the selected resource reference

Type: javax.resource.cci.Connectior

Authentication: Container

Description:

WebSphere Extensions

Connection management: Default

Isolation level:

WebSphere Bindings

JNDI Name: NRACSP2

Figure 11-7 Defining the resource reference for J2C

Configuring the built-in server for J2C

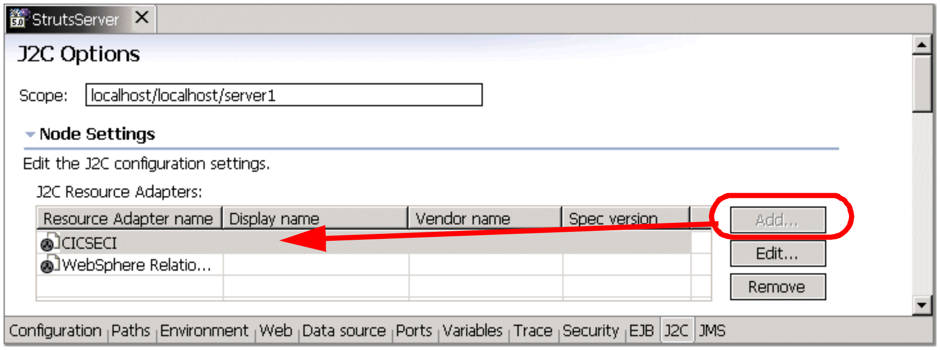
To test the application with the J2C connector to invoke the CICS COBOL program, the server must be configured.

> **Important:** This task is not supported with the early release of the Enterprise Developer. We had to make some manual corrections to get this to work.

Some of the configuration tasks cannot be completed in the server configuration dialog and must be done using the administrative console. To enable the console, edit the `StrutsServer` configuration and on the Configuration page select *Enable administration client*.

Installing the CICS ECI resource adapter

In the final product, a resource adapter can be installed from the J2C page of the configuration editor. Currently the *Add* option is greyed out.



StrutsServer

J2C Options

Scope: localhost/localhost/server1

Node Settings

Edit the J2C configuration settings.

J2C Resource Adapters:

Resource Adapter name	Display name	Vendor name	Spec version
CICSECI			
WebSphere Relatio...			

Figure 11-8 Installing a resource adapter

Instead we can use the administrative console to install a resource adapter:

- Start the `StrutsServer`.
- Start the console by selecting the `StrutsServer` in the Servers view and *Run administrative client* from the context menu.

  The installation process for a resource adapter is described in "Install the resource adapter archive (RAR) file" on page 341, where we install the resource adapter on a real WebSphere Application Server.
- Save the configuration, stop the console (*Logout* action), and stop the server.

Important: The install process did not complete normally and we had to fix the `resources.xml` file manually. This file is in:

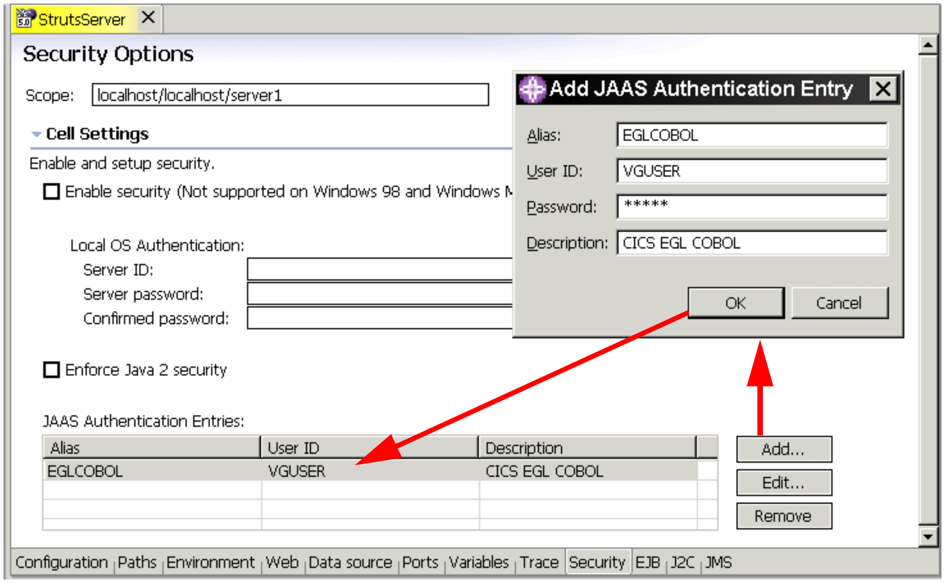
```
WSEDworkspace\ItsoServers\StrutsServer.wsc\cells\localhost\node\localhost
```

Configuring the J2C connector

When the resource adapter is installed correctly, we can progress with the configuration of the J2C connector. Open the `StrutsServer` configuration to complete this task.

J2C security

On the Security page, click *Add* to define a new JAAS authentication entry. You can use any alias name, but the user ID and password must be valid to run the CICS transaction (Figure 11-9).


StrutsServer

Security Options

Scope: localhost/localhost/server1

Cell Settings

Enable and setup security.

□ Enable security (Not supported on Windows 98 and Windows M

Local OS Authentication:

Server ID:

Server password:

Confirmed password:

□ Enforce Java 2 security

JAAS Authentication Entries:

Alias	User ID	Description
EGLCOBOL	VGUSER	CICS EGL COBOL

Add JAAS Authentication Entry

Alias: EGLCOBOL

User ID: VGUSER

Password: *****

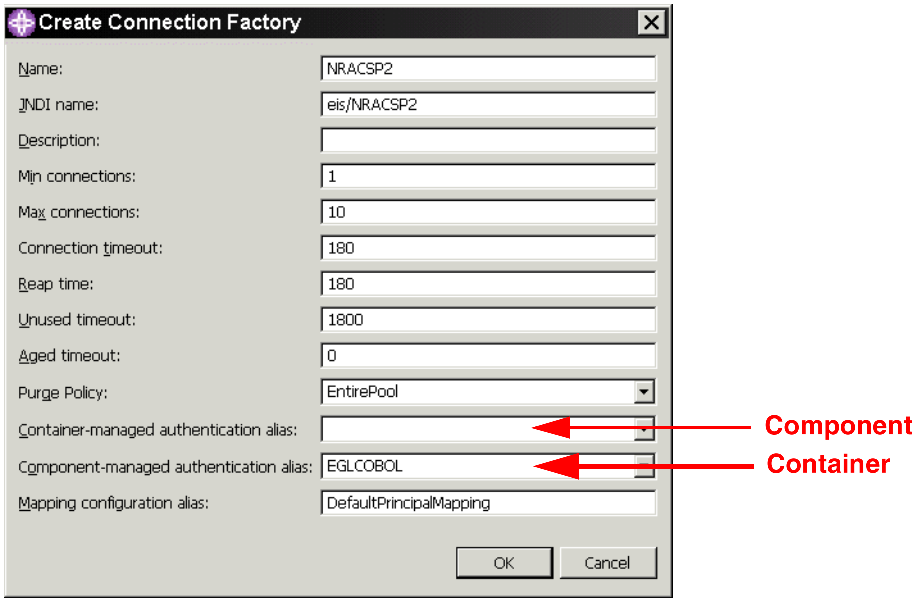
Description: CICS EGL COBOL

Figure 11-9 Defining container authentication for J2C

Connection factory

Switch to the J2C page in the configuration editor. Select the *CICSECI* resource adapter and click *Add* to define a connection factory. In the dialog (Figure 11-10):

- Enter `NRACSP2` (the CICS server name) as name.
- Enter `eis/NRACSP2` as JNDI name (this matches the Web application).
- Select `EGLCOBOL` for container authentication.
- Click *OK*.



Name: NRACSP2

JNDI name: eis/NRACSP2

Description:

Min connections: 1

Max connections: 10

Connection timeout: 180

Reap time: 180

Unused timeout: 1800

Aged timeout: 0

Purge Policy: EntirePool

Container-managed authentication alias:

Component-managed authentication alias: EGLCOBOL

Mapping configuration alias: DefaultPrincipalMapping

Component

Container

Figure 11-10 Defining a connection factory

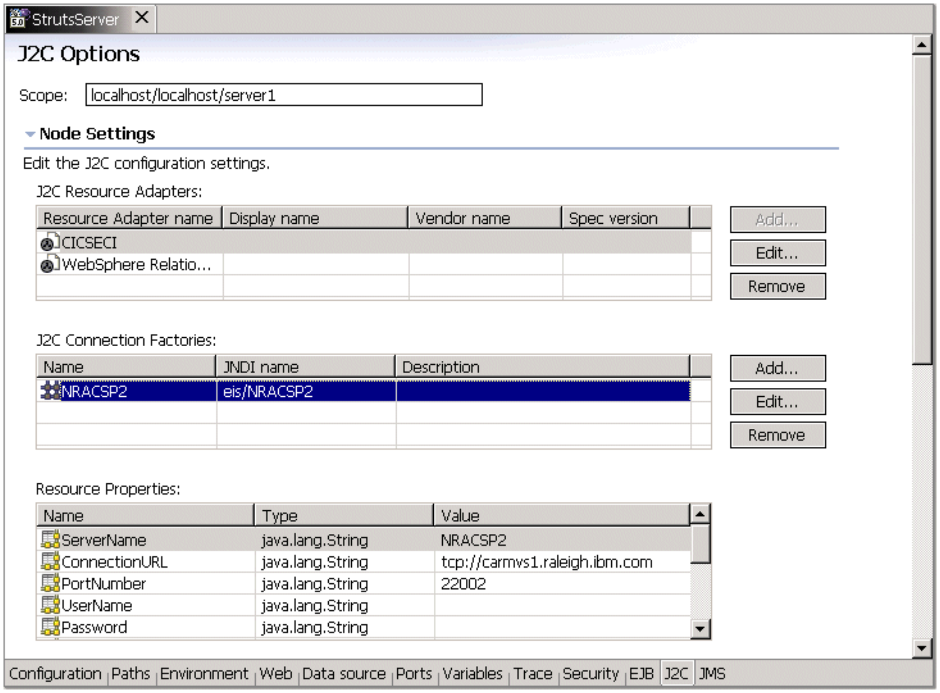
Note: the labels for container and component authentication are wrong in the dialog; they should be reversed.

Properties of the connection factory

Next, we have to define the properties of the connection factory. In the configuration dialog (Figure 11-11) select each property and change the values:

- ServerName—`NRACSP2`, the name of the CICS server
- ConnectionURL—`tcp://carmv1.raleigh.ibm.com`, the z/OS machine
- PortNumber—`22002`, the port of the CICS Transaction Gateway
- TPNName—`TRDE`, the name of the CICS transaction that runs the DB2 plan

The other properties can be left unchanged.



J2C Options

Scope: localhost/localhost/server1

Node Settings

Edit the J2C configuration settings.

J2C Resource Adapters:

Resource Adapter name	Display name	Vendor name	Spec version
CICSECI			
WebSphere Relatio...			

J2C Connection Factories:

Name	JNDI name	Description
NRACSP2	eis/NRACSP2	

Resource Properties:

Name	Type	Value
ServerName	java.lang.String	NRACSP2
ConnectionURL	java.lang.String	tcp://carmvs1.raleigh.ibm.com
PortNumber	java.lang.String	22002
UserName	java.lang.String	
Password	java.lang.String	

Figure 11-11 Connection factory properties

Save and close the configuration.

Configuring the DB2 JDBC connector

After configuring the J2C connector, our basic Struts EGL application did not work any more. We had to configure container authentication also for DB2:

- Open the `StrutsServer` configuration.
- On the Security page, define another entry with the name `DB2user` and user ID and password set to `db2admin` (same process as in Figure 11-9 on page 305).
- On the Data source page, select the *DB2JdbcDriver* and click *Edit* for the `TRADEDB` data source (Figure 8-44 on page 233). For the container-managed authentication alias, select the `DB2user` entry.
- Save the `StrutsServer` configuration.

Testing the COBOL CICS transactions

Start the StrutsServer in the Server perspective, then select the index.jsp and *Run on Server*. Try the login action with different uid:xxx values.

12

Implementing and using Web services

In this chapter we briefly introduce the concept of Web services.

We look at different ways to create Web services from our application parts. In particular we want to access EGL program as Web services. In this way, the EGL program becomes a callable function that can be used from any Web service client.

Note: Development is planning enhancements in the area of Web services for the final Enterprise Developer product. These enhancements will make it easier to create Web services from EGL programs.

Web services concepts

A good introduction to Web services concepts can be found in the redbook *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292.

Possible uses of Web services with Struts and EGL

When analyzing the concept and implementation of Struts-based Web applications and of EGL-generated code, we think that Web services could be used in multiple ways.

Creating Web services

In general, Web services can be created from JavaBeans and from session EJBs. (There are other ways, such as SQL statements and stored procedures, but they do not apply here.)

- Struts model classes that are usually invoked by Struts action classes can be turned into a Web service.
- A Java wrapper class that invokes an EGL program cannot be turned into a Web Service, because the `call` and `execute` methods do not return any data. A JavaBean with similar function can be created and turned into a Web service.
- A session EJB generated from an EGL program cannot be turned into a Web service, because the generated EJB is a stateful session bean; only stateless session beans can be converted into a Web service.

Consuming Web services

Web services created from any source could be consumed by Struts or non-Struts applications.

- Struts actions (action classes or model classes) can use Web services in their processing. The result of the Web services must be analyzed to decide on the proper success or failure actions.
- EGL programs could use Web services, but this may require complex code.

Preparing a client project for Web services

Web services are created in a Web project. We will use our `ItsoMyTradeWeb` project to create and run the Web services. For testing and client access, we use a new Web project called `ItsoMyTradeWebClient`.

Creating the client project

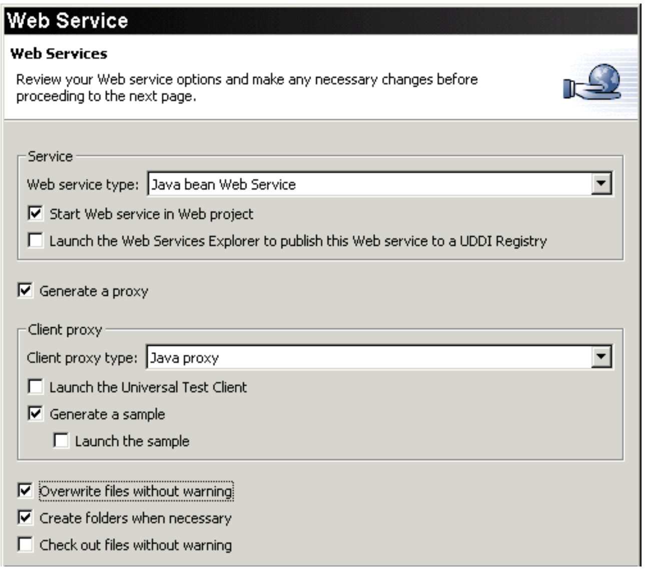
Here are the steps to create the client Web project. The sequence is the same as in “Using the wizard to create a Web project” on page 106:

- Start the wizard using *New -> Web -> Web Project*.
- Enter `ItsoMyTradeWebClient` as the name. Select *J2EE Web Project.* Deselect *Add Struts support*. Select *Create a default CSS file*. Click *Next*.
- Select *Existing* for the enterprise application and *Browse* to `ItsoMyTradeEAR`. Select *1.3* for the J2EE Level. Click *Finish*.
- Click *OK* to repair the server configuration (this adds the Web project to the server configuration under the EAR project).

Creating a Web service from a Struts model class

Let us turn one of the Struts model classes into a Web service. For this exercise, we use the `strutsEGL.Login` class.

- Make sure all servers are stopped.
- In the Web perspective, select the `strutsEGL.Login` class and *New -> Other -> Web Services -> Web Service*. Click *Next*.
- Figure 12-1 shows the starting panel of the Web service wizard.



Web Service

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Service

Web service type: Java bean Web Service

☑ Start Web service in Web project

□ Launch the Web Services Explorer to publish this Web service to a UDDI Registry

☑ Generate a proxy

Client proxy

Client proxy type: Java proxy

□ Launch the Universal Test Client

☑ Generate a sample

□ Launch the sample

☑ Overwrite files without warning

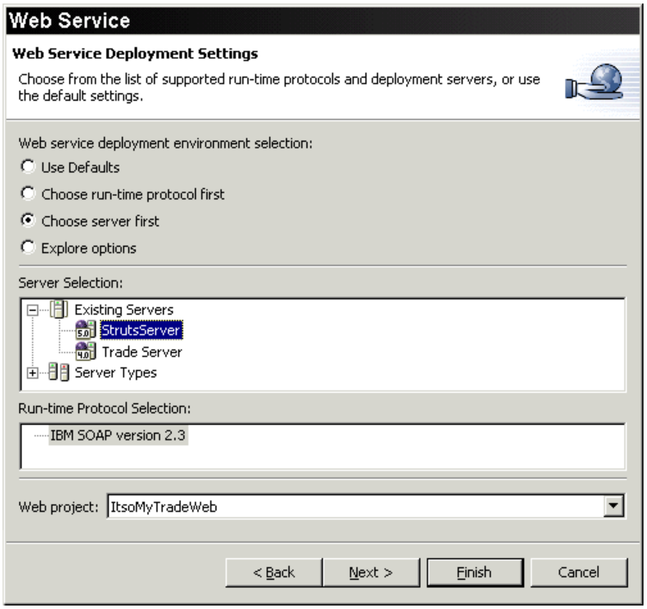
☑ Create folders when necessary

□ Check out files without warning

Figure 12-1 Web service wizard: start

- Select *JavaBean Web service* for the type of Web service.
- Select *Start Web service in Web project*.
- Select *Generate a proxy* and *Generate a sample*.
- Select *Overwrite files without warning* and *Create folders when necessary.*
- Click *Next*.

- For Deployment Settings (Figure 12-2), select *Choose server first* and then select the `StrutsServer`. For the Web project, make sure that `ItsoMyTradeWeb` is selected. Click *Next*.



Web Service

Web Service Deployment Settings

Choose from the list of supported run-time protocols and deployment servers, or use the default settings.

Web service deployment environment selection:

○ Use Defaults
○ Choose run-time protocol first
● Choose server first
○ Explore options

Server Selection:

- Existing Servers
  - StrutsServer
  - Trade Server
- Server Types

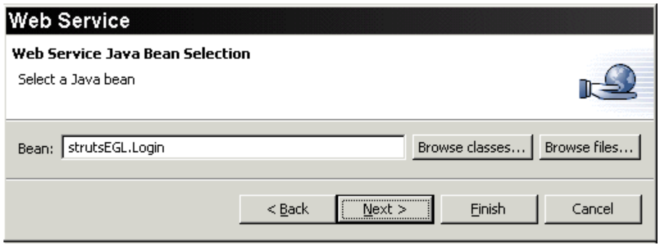
Run-time Protocol Selection:

IBM SOAP version 2.3

Web project: ItsoMyTradeWeb

Figure 12-2 Web service wizard: Deployment Settings

- For JavaBean Selection (Figure 12-3), the `strutsEGL.Login` bean is preselected. Click *Next*.



Web Service

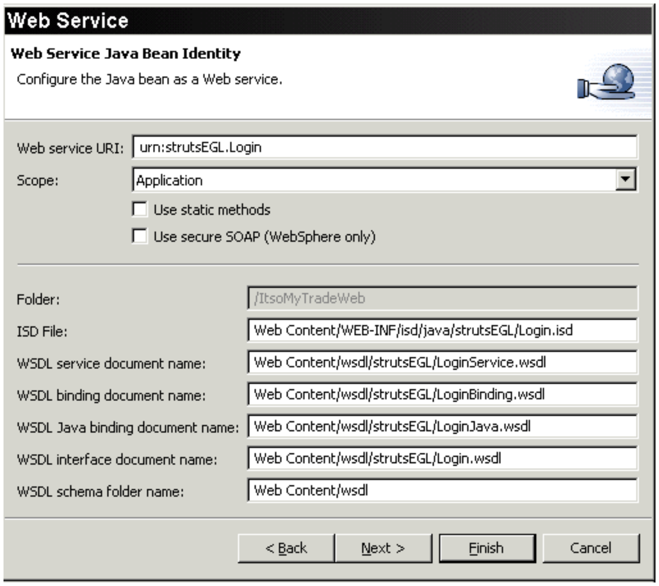
Web Service Java Bean Selection

Select a Java bean

Bean: strutsEGL.Login

Figure 12-3 Web service wizard: JavaBean Selection

- For the JavaBean Identity (Figure 12-4), change the URI to `urn:StrutsEGLLogin`. Leave all other fields unchanged. Click *Next*.



Web Service

Web Service Java Bean Identity

Configure the Java bean as a Web service.

Web service URI: urn:strutsEGL.Login

Scope: Application

□ Use static methods

□ Use secure SOAP (WebSphere only)

Folder: /ItsoMyTradeWeb

ISD File: Web Content/WEB-INF/isd/java/strutsEGL/Login.isd

WSDL service document name: Web Content/wsdl/strutsEGL/LoginService.wsdl

WSDL binding document name: Web Content/wsdl/strutsEGL/LoginBinding.wsdl

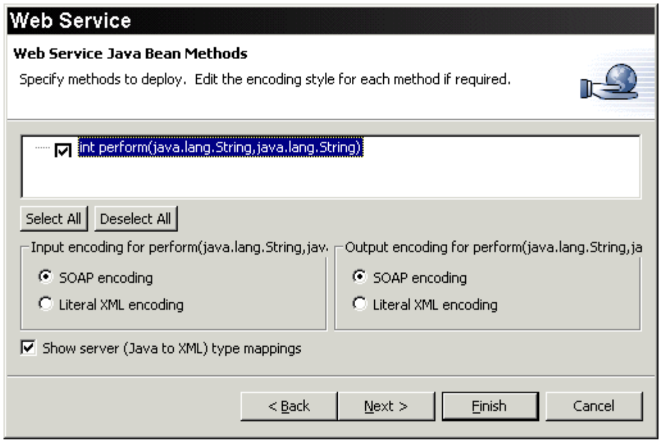
WSDL Java binding document name: Web Content/wsdl/strutsEGL/LoginJava.wsdl

WSDL interface document name: Web Content/wsdl/strutsEGL/Login.wsdl

WSDL schema folder name: Web Content/wsdl

Figure 12-4 Web service wizard: JavaBean Identity

- For JavaBean Methods (Figure 12-5), the `perform` method is preselected and you can see that SOAP encoding is used for parameters and results. Select *Show server (Java to XML) type mappings*. Click *Next*.



Web Service

Web Service Java Bean Methods

Specify methods to deploy. Edit the encoding style for each method if required.

☑ int perform(java.lang.String,java.lang.String)

Input encoding for perform(java.lang.String,jav.

● SOAP encoding

○ Literal XML encoding

Output encoding for perform(java.lang.String,ja

● SOAP encoding

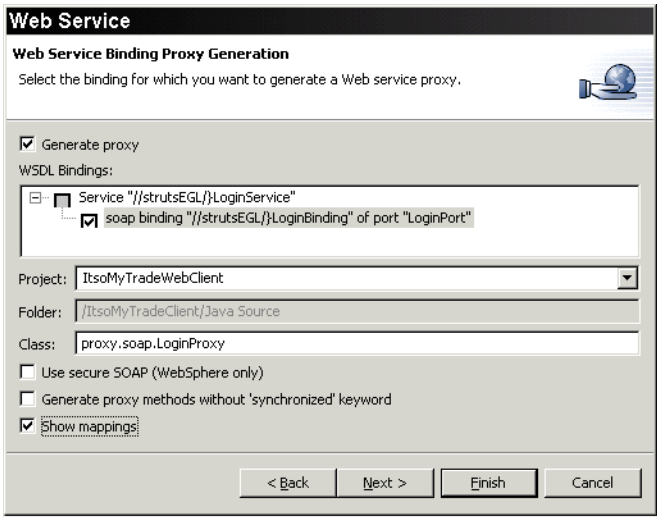
○ Literal XML encoding

☑ Show server (Java to XML) type mappings

Figure 12-5 Web service wizard: JavaBean Methods

- No change is required for Java to XML Mappings. Click *Next*.

- For Binding Proxy Generation (Figure 12-6), *Generate Proxy* is preselected. The project name should be set to `ItsoMyTradeWebClient`. Leave the proxy class as `proxy.soap.LoginProxy`. Select *Show mappings*. Click *Next*.



Web Service

Web Service Binding Proxy Generation

Select the binding for which you want to generate a Web service proxy.

☑ Generate proxy

WSDL Bindings:

- □ Service "//strutsEGL/}LoginService"
  - ☑ soap binding "//strutsEGL/}LoginBinding" of port "LoginPort"

Project: ItsoMyTradeWebClient

Folder: /ItsoMyTradeClient/Java Source

Class: proxy.soap.LoginProxy

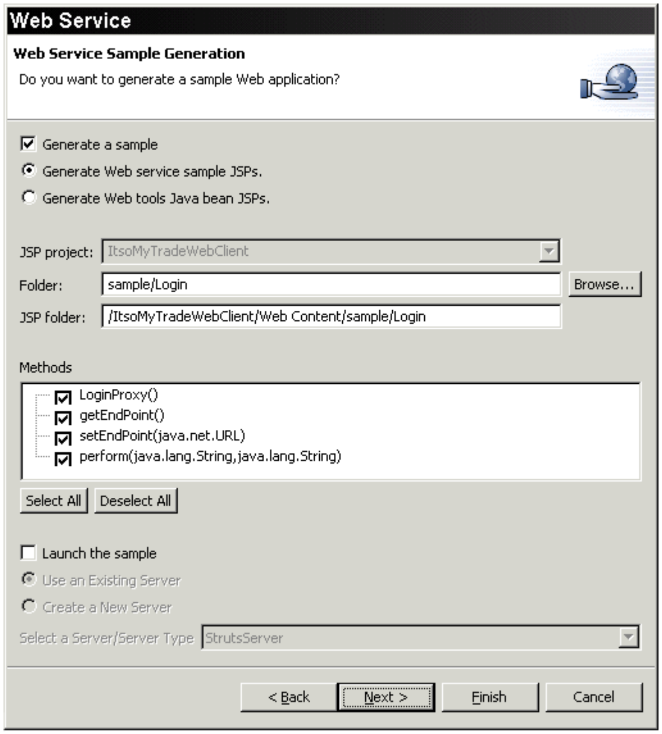
□ Use secure SOAP (WebSphere only)

□ Generate proxy methods without 'synchronized' keyword

☑ Show mappings

Figure 12-6 Web service wizard: Binding Proxy Generation

- No change is required for Java to XML Mappings. Click *Next*.
- No change is required for SOAP Binding Mapping Configuration. Click *Next*.
- Skip the Test Client panel (we do not start it). Click *Next*.
- For Sample Generation (Figure 12-7):
  - *Generate a sample* is preselected.
  - Select *Generate Web service sample JSPs*.
  - Leave the folder as `sample/Login` and the JSP folder as `ItsoMyTradeWebClient/Web Content/sample/Login`.
  - Do not select *Launch the sample*.
  - Click *Next*.



Web Service

Web Service Sample Generation

Do you want to generate a sample Web application?

☑ Generate a sample

● Generate Web service sample JSPs.

○ Generate Web tools Java bean JSPs.

JSP project: ItsoMyTradeWebClient

Folder: sample/Login

JSP folder: /ItsoMyTradeWebClient/Web Content/sample/Login

Methods

☑ LoginProxy()
☑ getEndPoint()
☑ setEndPoint(java.net.URL)
☑ perform(java.lang.String,java.lang.String)

□ Launch the sample

● Use an Existing Server

○ Create a New Server

Select a Server/Server Type: StrutsServer

Figure 12-7 Web service wizard: Sample Generation

- Skip the Publication panel. We will not launch the UDDI explorer.
- Click *Finish*. Be patient..... the code is generated and the server is started.

Generated files

The Web service wizard generates the following files into the Web Content of the project where the Web service is installed (`ItsoMyTradeWeb`):

- Administrative application in the `admin` folder.
- A `Login.isd` file into `WEB-INF\isd\java\strutsEGL`. The ISD file contains the deployment information of the Web service:

```
<isd:service id="urn:strutsEGL.Login" xmlns:isd="......">
  <isd:provider type="java" scope="Application" methods="perform">
    <isd:java class="strutsEGL.Login" static="false"/>
  </isd:provider>
  <isd:mappings>
  </isd:mappings>
</isd:service>
```

- Two JAR files, soapcfg.jar and xsd.bean.runtime.jar, are added to WEB-INF\lib. These files contain the SOAP run-time library and encoding.
- Two SOAP servlets, rpcrouter and messagerouter into the WEB-INF\web.xml file. The rpcrouter servlet will be used to invoke our Web service.
- Four Web Service Description Language (WSDL) files into the wsdl\strustEGL folder. These WSDL files describe the Web service with its name, methods, parameters, and results.
  - The Login.wsdl and LoginBinding.wsdl files contain the specification of the service.
  - The LoginService.wsdl file points to the actual server where the service is installed.
  - The LoginJava.wsdl file is an alternative binding file.
- The SOAP deployment descriptor, dds.xml. This file is the concatenation of all the individual ISD files.
- The SOAP configuration file, soap.xml. This file points to the configuration manager that reads the dds.xml file.

The Web service wizard generates the following files into the client project (ItsoMyTradeWebClient):

- The client proxy class, LoginProxy, into the soap.proxy package under Java Source.
- A test sample into the Web Content\sample\Login folder. The sample consists of four JSPs; the starting point is the TestClient.jsp file.

Creating a Web service from a wrapper class

The wrapper classes that are generated to invoke an EGL program could also be converted into a Web service.

These wrapper classes are, however, not well suited to become Web services. If you analyze the generated code, for example, the LogacWrapper class (in the tradeEGL.genned package) you notice that you require two methods to invoke the EGL program and retrieve the result:

- call(Logws)—execute the EGL program with a Logws record as parameter
- getLogws()—retrieve the Logws to extract the result data

This would result in two Web services interactions. It is therefore easier to combine the processing into a model class, such as the Login class used in “Creating a Web service from a Struts model class” on page 311.

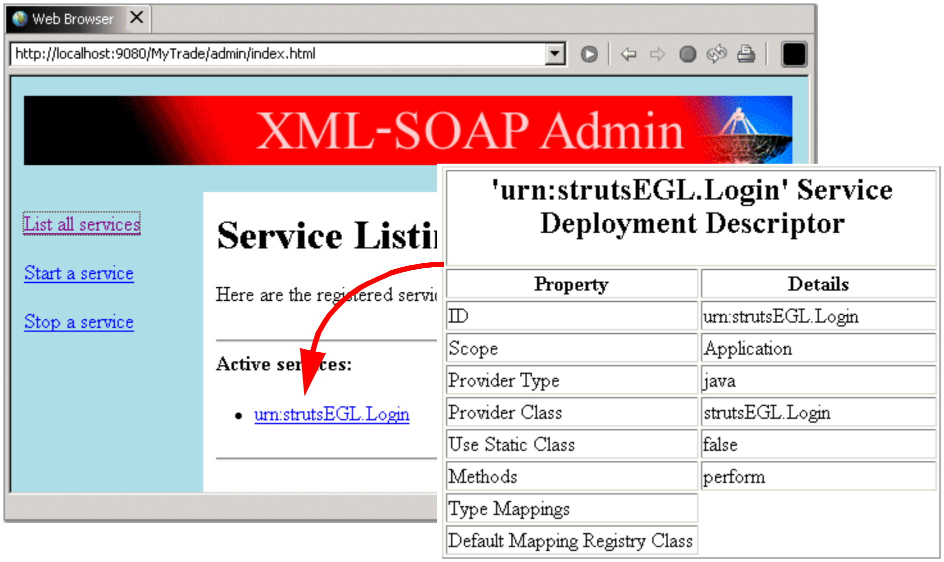
Testing the Web service

Note that the StrutsServer has been started automatically. The Web service has been installed in the server and is running.

SOAP administrative application

First we run the administrative application. It will show us that the Web service is installed and running.

- Select the index.html file in ItsoMyTradeWeb\Web Content\admin and *Run on Server* from the context menu.
- The XML SOAP Admin Web page displays (Figure 12-8).
- Select *List all services* and the urn:strutsEGL.Login service is listed.
- Select the service to see its details.



Here are the registered servi

'urn:strutsEGL.Login' Service Deployment Descriptor

Property	Details
ID	urn:strutsEGL.Login
Scope	Application
Provider Type	java
Provider Class	strutsEGL.Login
Use Static Class	false
Methods	perform
Type Mappings	
Default Mapping Registry Class	

Figure 12-8 SOAP administrative application

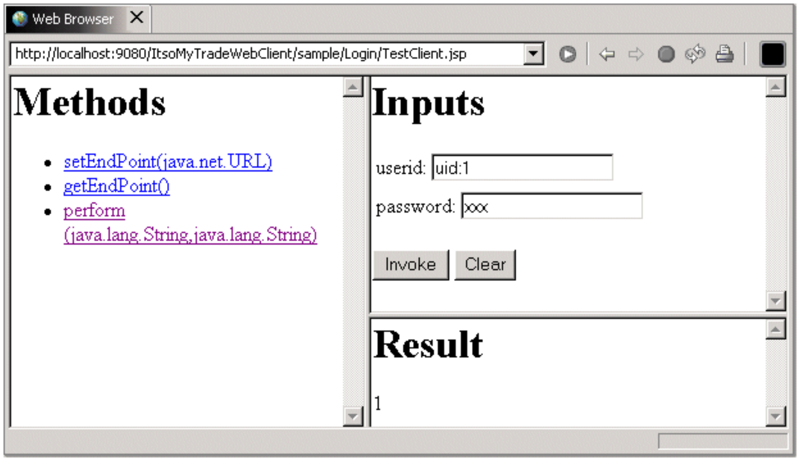
You can use the *Start/Stop a Web service* actions to make Web services available or unavailable. The SOAP server will remember the setting when the WebSphere server is restarted and only make started Web services available to clients.

A stopped Web service returns a SOAP fault to the calling client application.

Sample test client

Next we run the sample test client that was generated for us:

- Select the `TestClient.jsp` in `ItsoMyTradeWebClient\Web Content \sample\Login` and *Run on Server* from the context menu.
- The test client is displayed (Figure 12-9).



http://localhost:9080/ItsoMyTradeWebClient/sample/Login/TestClient.jsp

- setEndPoint(java.net.URL)
- getEndPoint()
- perform (java.lang.String,java.lang.String)

userid: uid:1

password: xxx

1

Figure 12-9 Web service test client

- Select the `perform` method. The input pane displays the parameters. Enter `uid:1` and `xxx` for user ID and password, and click *Invoke*.
- After invoking the Web service, the result pane shows the result (`1`) of a successful login.
- Enter a bad user ID or password and the result is 0.

How does this work?

The work is performed in the `Result.jsp`:

- The proxy class (`LoginProxy`) is allocated as a JavaBean:

```
<jsp:useBean id="id" scope="session" class="proxy.soap.LoginProxy" />
```

> **Note:** You should change the scope to `request` to enable multiple test clients of different Web services to execute in the same run. With session scope, you can encounter class cast errors.

- For the `perform` method, the parameters are extracted from the request block and the Web service is invoked using the `perform` method of the proxy bean.

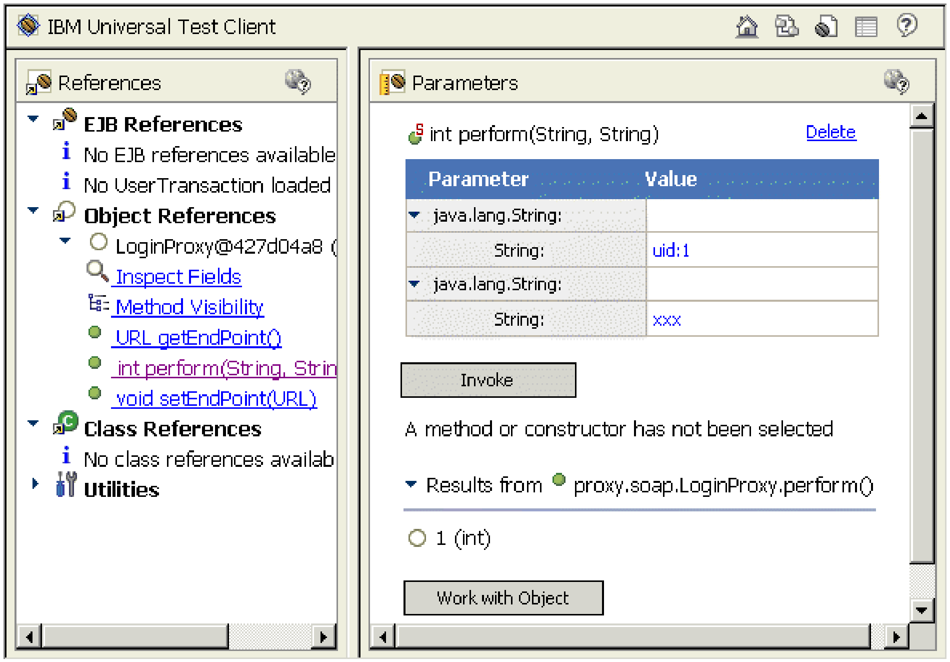
- The proxy bean allocates a SOAP `Call` object that is filled with parameter and encoding information. Finally the invoke method of the `Call` object calls the Web service in the server. The result object is converted to the required return type (`int`) and passed back to the caller (`Result.jsp`).
- The `Result.jsp` displays the return value.

Universal test client

We can also use the universal test client to test the Web service. To run the Web service, we have to instantiate the `LoginProxy` class and invoke its perform method.

After starting the `StrutsServer`, select the `LoginProxy` class in the `proxy.soap` package of the `ItsoMyTradeClient` project and *Launch Universal Test Client* from the context menu.

The universal test client starts and preloads an instance of the `LoginProxy` class. Select the `perform` method, enter `uid:1` and `xxx` as user ID and password, and click *Invoke*. The result of the Web service is displayed (Figure 12-10).



int perform(String, String)

Parameter	Value
java.lang.String:	
String:	uid:1
java.lang.String:	
String:	xxx

A method or constructor has not been selected

Results from proxy.soap.LoginProxy.perform()

1 (int)

Figure 12-10 Web service testing with the universal test client

Using the TCP/IP monitor to see the SOAP messages

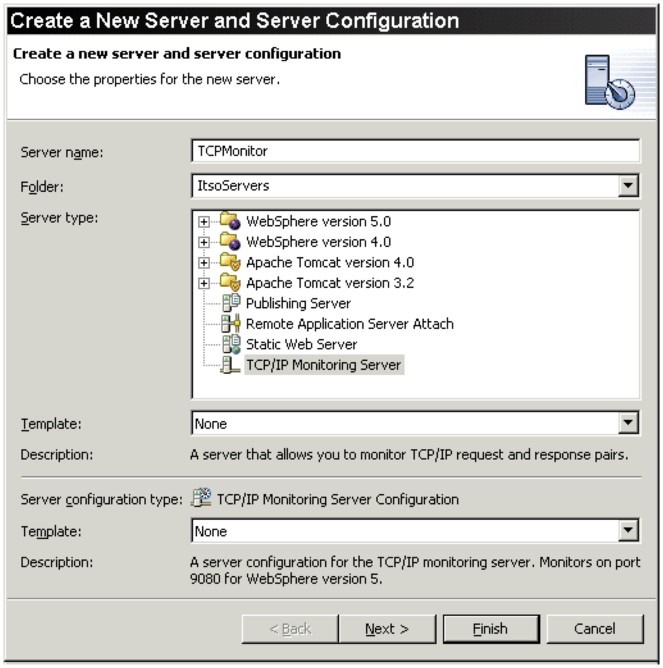
With all the testing methods explored, we wonder what the actual SOAP messages of a Web service interaction look like.

The Enterprise Developer provides a TCPIP monitoring server that can display the traffic.

Configuring a TCP/IP monitoring server

In the Server Configuration view of the Server perspective, select *New -> Server and Server Configuration* from the context menu. In the dialog (Figure 12-11):

- Enter `TCPMonitor` as name
- Select `ItsoServers` as folder (project)
- Select *TCP/IP Monitoring Server* as server type
- Click *Finish*.



Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: TCPMonitor

Folder: ItsoServers

Server type: TCP/IP Monitoring Server

Template: None

Description: A server that allows you to monitor TCP/IP request and response pairs.

Server configuration type: TCP/IP Monitoring Server Configuration

Template: None

Description: A server configuration for the TCP/IP monitoring server. Monitors on port 9080 for WebSphere version 5.

Figure 12-11 Creating a TCP/IP Monitoring Server

The `TCPMonitor` server appears in the Server view.

Running the Web service through the monitor

First we start the `TCPMonitor` server. from the Servers view. The Console displays:

```
Monitoring server started
localhost:9081 -> localhost:9080
```

By default the proxy (`LoginProxy`) sends the SOAP request to the `rpcrouter` servlet using port 9080 (the default WebSphere port):

```
stringURL = "http://localhost:9080/MyTrade/servlet/rpcrouter";
```

To display the traffic, we have to send the request through port 9081 (the default TCP/IP monitor port). There are two ways to accomplish this:

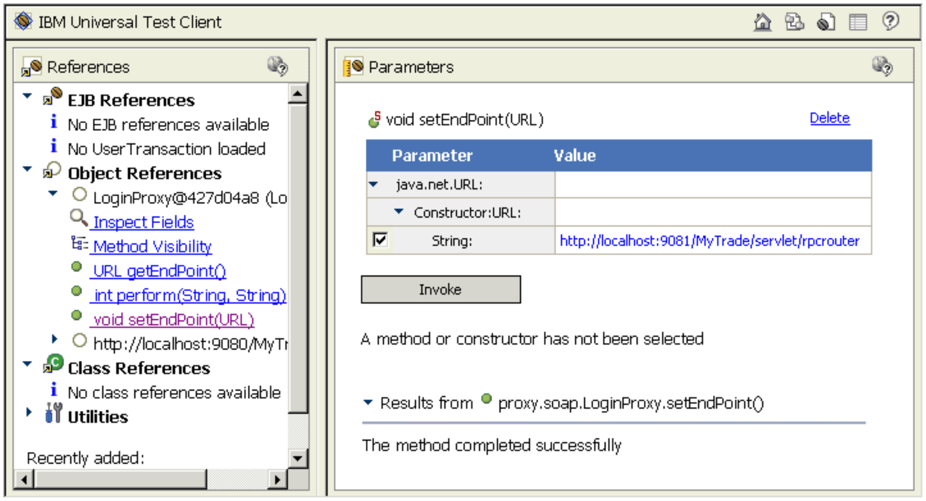
- We can edit the `LoginProxy` class (in `soap.proxy` of `ItsoMyTradeClient`) and change the URL to go through port 9081:

  ```
  stringURL = "http://localhost:9081/MyTrade/servlet/rpcrouter";
  ```

- Alternatively we can change the port dynamically. The proxy class provides `getEndpoint` and `setEndpoint` methods to manipulate the SOAP address.

Changing the port of the SOAP request

We will use the second alternative. In the universal test client, select the `setEndpoint` method, the `URL(string)` constructor for the parameter, enter the new request address, and click *Invoke* (Figure 12-12).



void setEndPoint(URL)

Parameter	Value
java.net.URL:	
Constructor:URL:	
☑ String:	http://localhost:9081/MyTrade/servlet/rpcrouter

A method or constructor has not been selected

Results from proxy.soap.LoginProxy.setEndPoint()

The method completed successfully

Figure 12-12 Changing the SOAP endpoint address

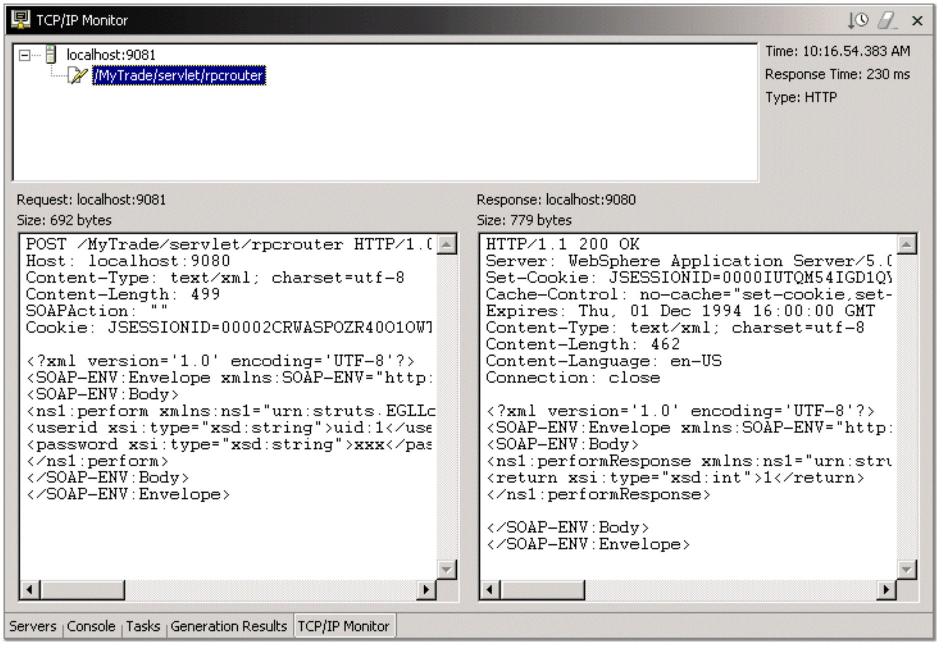
Note that you can use the `getEndpoint` method to retrieve the existing address.

Run the Web service

Select the `perform` method and run the Web service call again with `uid:1` and `xxx` as parameters.

View the SOAP messages

In the Server perspective, open the TCP/IP Monitor view and select the `rpcrouter` request (Figure 12-13).



Time: 10:16.54.383 AM
Response Time: 230 ms
Type: HTTP

Request: localhost:9081
Size: 692 bytes

```
POST /MyTrade/servlet/rpcrouter HTTP/1.(
Host: localhost:9080
Content-Type: text/xml; charset=utf-8
Content-Length: 499
SOAPAction: ""
Cookie: JSESSIONID=00002CRWASPOZR40O1OW1

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http:
<SOAP-ENV:Body>
<ns1:perform xmlns:ns1="urn:struts.EGLLc
<userid xsi:type="xsd:string">uid:1</use
<password xsi:type="xsd:string">xxx</pas
</ns1:perform>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response: localhost:9080
Size: 779 bytes

```
HTTP/1.1 200 OK
Server: WebSphere Application Server/5.(
Set-Cookie: JSESSIONID=0000IUTQM54IGD1Q\
Cache-Control: no-cache="set-cookie,set-
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Content-Type: text/xml; charset=utf-8
Content-Length: 462
Content-Language: en-US
Connection: close

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http:
<SOAP-ENV:Body>
<ns1:performResponse xmlns:ns1="urn:stru
<return xsi:type="xsd:int">1</return>
</ns1:performResponse>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 12-13 TCP/IP Monitor view

In the left pane, you can see the SOAP input message with two parameters:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
   <ns1:perform xmlns:ns1="urn:struts.EGLLogin"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <userid xsi:type="xsd:string">uid:1</userid>
      <password xsi:type="xsd:string">xxx</password>
   </ns1:perform>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In the right pane you can see the SOAP output message with the return value:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
   <ns1:performResponse xmlns:ns1="urn:struts.EGLLogin"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:int">1</return>
   </ns1:performResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Note that SOAP encoding supports the standard data types, such as `string`, `int`, `float`, `double`, and so forth. These data types are defined `xsi:type` in a SOAP XML schema.

Creating a Web service that returns the working storage

The current Struts model class (`Login`) only returns an integer to signal a successful or unsuccessful login. A more interesting approach would be to retrieve the complete working storage record (although it currently does not hold much information).

In addition, the `Logws` carries a number of EGL class references with it. Therefore, it is easier to create a simple JavaBean that hold the `Logws` data.

To illustrate this concept, we create a new service JavaBean (`LoginLogws`) that returns a simple JavaBean (`LogwsBean`). Then we create a Web service from the service JavaBean.

Create the data JavaBean

In the `ItsoMyTradeWeb` project, create a package named `webserv`. In this package, create a simple class named `LogwsBean`:

- The class must implement `java.io.Serializable`.
- Add four properties (strings):

```
private String userid   = "";
private String password = "";
private String action   = "";
private String status   = "";
```

- Create getter and setter methods for the properties. You can do that easily from the Outline view using the context menu on the properties.
- The code of the class should look similar to Figure 12-14.

```
package webserv;

public class LogwsBean implements java.io.Serializable
{
   private String userid   = "";
   private String password = "";
   private String action   = "";
   private String status   = "";

   public java.lang.String getAction( ) { return action; }
   public java.lang.String getPassword( ) { return password; }
   public java.lang.String getStatus( ) { return status; }
   public java.lang.String getUserid( ) { return userid; }

   public void setAction(String action) {
      this.action = action;
   }
   public void setPassword(String password) {
      this.password = password;
   }
   public void setStatus(String status) {
      this.status = status;
   }
   public void setUserid(String userid) {
      this.userid = userid;
   }
}
```

Figure 12-14 LogwsBean data bean as result class (abbreviated)

Create the service JavaBean

Create a copy of the `Login` class in the `webserv` package of `ItsoMyTradeWeb`. (Select the `Login` class and *Copy*, then select the `webserv` package and *Paste*.) Rename the copy `LoginLogws` (select *Rename* from the context menu).

Edit the `LoginLogws` class:

- Change the `perform` method:
  – Change the return type to `webserv.LogwsBean`.
  – Change all `return 0` to `return null`.
  – Return the `webserv.LogwsBean` when successful.

- The code of the `LoginLogws` class is shown in Figure 12-15.

```
package webserv;
import com.ibm.vgj.cso.CSOException;
import com.ibm.vgj.cso.CSOLocalPowerServerProxy;
import com.ibm.vgj.cso.CSOPowerServer;
import tradeEGL.genned.LogacWrapper;
import tradeEGL.genned.Logws;

public class LoginLogws {
   private CSOPowerServer powerServer;

   public webserv.LogwsBean perform(String userid, String password) {
      try {
         powerServer = new CSOLocalPowerServerProxy();
      } catch (CSOException e) {
         e.printStackTrace();
         return null;
      }
      // Create an instance of the wrapper class
      LogacWrapper wrapper = new LogacWrapper(powerServer);
      // Set up parameter Logws
      Logws newLogws = wrapper.getLogws();

      try {
         newLogws.setUserid  (userid);
         newLogws.setPassword(password);
         newLogws.setAction  ("inquire");

         // execute the wrapper
         wrapper.execute();

         LogwsBean logwsbean = new LogwsBean();
         logwsbean.setUserid  ( newLogws.getUserid()   );
         logwsbean.setPassword( newLogws.getPassword() );
         logwsbean.setAction  ( newLogws.getAction()   );
         logwsbean.setStatus  ( newLogws.getStatus()   );
         return logwsbean;
      } catch (Exception e) {
         System.out.println("LoginLogws Exception: " + e.getMessage());
         e.printStackTrace();
         return null;
      }
   }
}
```

Figure 12-15   LoginLogws service bean

Create a Web service from the LoginLogws JavaBean

To create the Web service from the LoginLogws JavaBean, repeat the steps of “Creating a Web service from a Struts model class” on page 311:

- Select the LoginLogws class and *New -> Web Services -> Web Service*.
- Select *Generate a proxy* and *Generate a sample*.
- The URI becomes urn:weserv.LoginLogws
- The proxy becomes LoginLogwsProxy. Make sure the project for the proxy is ItsoMyTradeWebClient.
- The sample client goes into sample/LoginLogws (make sure to select Generate Web service sample JSPs).

Generated files

The Web service wizard generates the following additional files into the Web Content of the project where the Web service is installed (ItsoMyTradeWeb):

- A LoginLogws.isd file into WEB-INF\isd\java\webserv. The ISD file contains the deployment information of the Web service, including information about encoding classes for the JavaBean:

```
<<isd:service id="urn:webserv.LoginLogws"
      xmlns:isd="http://xml.apache.org/xml-soap/deployment">
  <isd:provider type="java" scope="Application" methods="perform">
    <isd:java class="webserv.LoginLogws" static="false"/>
  </isd:provider>
  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="http://webserv/" qname="x:LogwsBean"
      javaType="webserv.LogwsBean"
      java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
      xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
  </isd:mappings>
</isd:service>
```

- Four WSDL files into the wsdl\webserv folder. These WSDL files describe the Web service with its name, methods, parameters, and results. The folder also contains an XML XSD file (LogwsBean.xsd) that describes the JavaBean.
- The SOAP deployment descriptor, dds.xml is updated with the new ISD file.

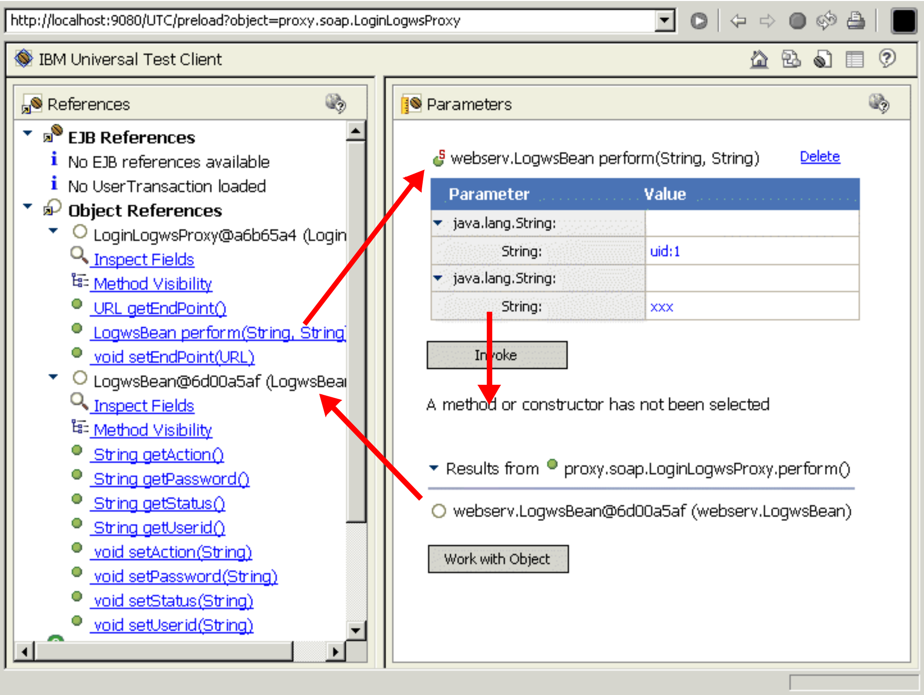
The Web service wizard generates the following files into the client project (ItsoMyTradeWebClient):

- The client proxy class, LoginLogwsProxy, into the soap.proxy package.
- The sample in the sample\LoginLogws folder.
- A copy of the LogwsBean class in a new webserv package.

Use the universal test client

Select the `LoginLogwsProxy` class in the `ItsoMyTradeWebClient` project and *Launch Universal Test Client* (context). The test client starts and loads an instance of the proxy bean:

- Select the `perform` method, enter `uid:1` and `xxx` as parameters and click *Invoke*.
- The result is a `LogwsBean`. Click *Work with Object* to add the result to Object References (Figure 12-16).
- Expand the `LogwsBean` and run its `getStatus` (and other get methods) method to retrieve the result data values.



http://localhost:9080/UTC/preload?object=proxy.soap.LoginLogwsProxy
IBM Universal Test Client
References
Parameters

EJB References
No EJB references available
No UserTransaction loaded
Object References
LoginLogwsProxy@a6b65a4 (Login
Inspect Fields
Method Visibility
URL getEndPoint()
LogwsBean perform(String, String)
void setEndPoint(URL)
LogwsBean@6d00a5af (LogwsBea
Inspect Fields
Method Visibility
String getAction()
String getPassword()
String getStatus()
String getUserid()
void setAction(String)
void setPassword(String)
void setStatus(String)
void setUserid(String)

webserv.LogwsBean perform(String, String) Delete

Parameter	Value
java.lang.String:	
String:	uid:1
java.lang.String:	
String:	xxx

A method or constructor has not been selected

Results from proxy.soap.LoginLogwsProxy.perform()

webserv.LogwsBean@6d00a5af (webserv.LogwsBean)

Figure 12-16 Universal test client run with JavaBean result

Use the generated sample

We will run the generated sample later (see “Running the sample client” on page 335). You can select the `sample\LoginLogws\TestClient.jsp` and *Run on Server*, then select the `perform` method, enter values for user ID and password and click *Invoke*. The result data of the `LogwsBean` is displayed as shown in Figure 12-22 on page 335.

Use the TCP/IP monitoring server

When you use the TCP/IP monitoring server for the LoginLogwsProxy, the SOAP output displays as:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
   <ns1:performResponse xmlns:ns1="urn:webserv.LoginLogws"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xmlns:ns2="http://webserv/" xsi:type="ns2:LogwsBean">
         <status xsi:type="xsd:string">1</status>
         <userid xsi:type="xsd:string">uid:1</userid>
         <password xsi:type="xsd:string">xxx</password>
         <action xsi:type="xsd:string">inquire</action>
      </return>
   </ns1:performResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Notice how the LogwsBean has been serialized into XML. The four fields of the JavaBean are clearly visible in the SOAP XML output.

On the client side, the JavaBean is reconstructed from the XML.

Creating a Web service client

Let us create a small Web client that uses the two Web services. This client consist of an HTML file (LoginClient.html) with a form for user ID and password. From the form a servlet (LoginServlet) is invoked. The servlet calls the Web services using the appropriate proxy class and displays the result.

Create the HTML page with an input form

Create a new HTML file in ItsoMyTradeWebClient\Web Content (select *New -> HTML/XHTML file*). Name the file LoginClient.html.

Edit the code and replace the source with the file shown in Figure 12-17.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=WINDOWS-1252">
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>LoginClient.html</TITLE>
</HEAD>
<BODY>
<H1> Web Service Login Client </H1>
<FORM action="LoginServlet">
<TABLE border="0">
   <TR>
      <TD>Enter the user ID:</TD>
      <TD><INPUT type="text" name="userid" size="10" maxlength="10"></TD>
   </TR><TR>
      <TD>Enter the password:</TD>
      <TD><INPUT type="password" name="password" size="10"
                  maxlength="10"></TD>
   </TR><TR>
      <TD>&nbsp;</TD> <TD>&nbsp;</TD>
   </TR><TR>
      <TD><INPUT type="submit" name="logws" value="LoginLogws"></TD>
      <TD><INPUT type="submit" name="login" value="Login"></TD>
   </TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Figure 12-17 Web service client HTML

Notice that the form has two buttons to invoke either of the two Web services we created.

Create the servlet to invoke the Web services

In `ItsoMyTradeWebClient\Java Source`, create a new package named `client`. In the `client` package, create a servlet named `LoginServlet` (*New -> Servlet*) and add it to the `web.xml` file.

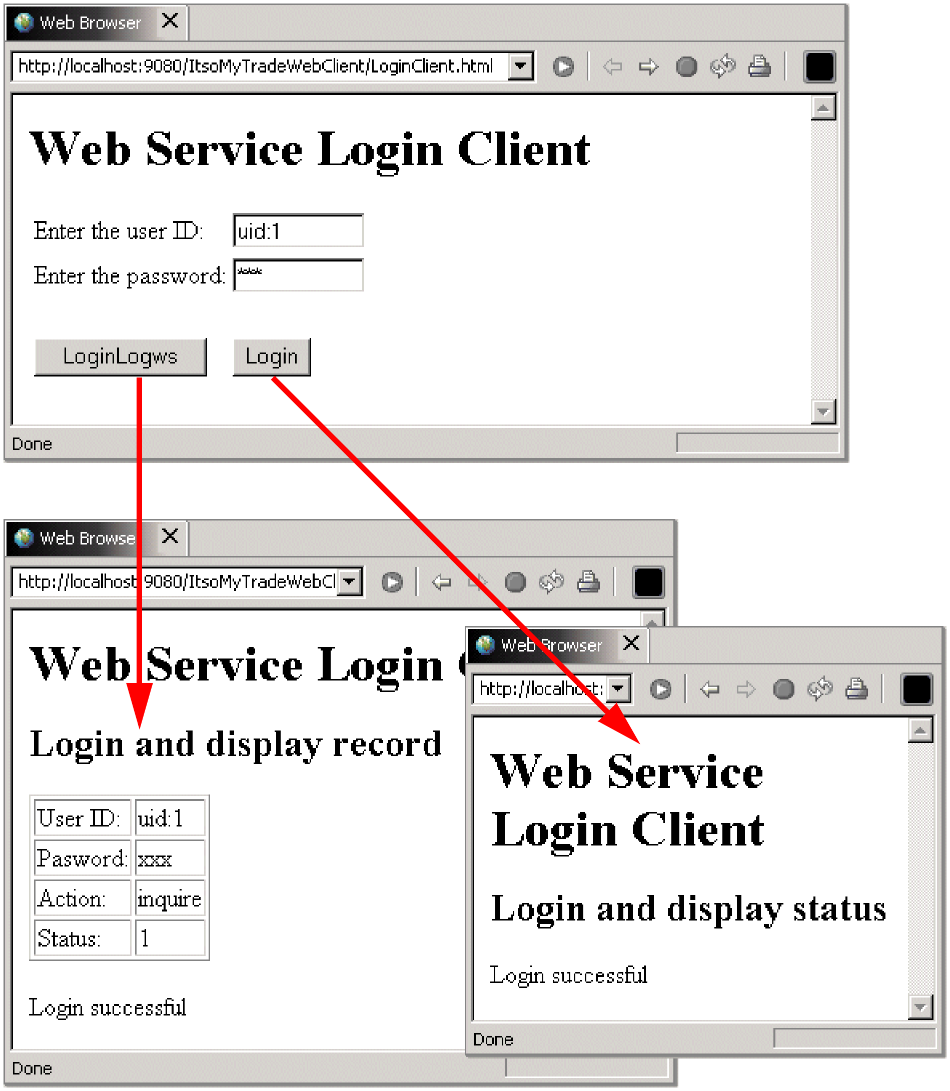
Edit the code and replace the source with the file shown in Figure 12-18.

```
package client;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import java.io.PrintWriter;
import proxy.soap.*;
import webserv.LogwsBean;

public class LoginServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
   doPost(req, resp);
}
public void doPost(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
   PrintWriter out = resp.getWriter();
   out.println("<html><body><h1>Web Service Login Client</h1>");
   String loginBut  = req.getParameter("login");
   String logwsBut  = req.getParameter("logws");
   String userid    = req.getParameter("userid");
   String password  = req.getParameter("password");
   try {
      if (loginBut  != null) {                                  // Login Web service
         out.println("<h2> Login and display status </h2>");
         LoginProxy loginProxy = new LoginProxy();
         int status = loginProxy.perform(userid, password);
         if (status == 1)
            out.println("<p>Login successful");
         else out.println("<p>Login failed");
      }
      if (logwsBut  != null) {                             // LoginLogws Web service
         out.println("<h2> Login and display record </h2>");
         out.println("<TABLE border=\"1\"><TBODY>");
         LoginLogwsProxy logwsProxy = new LoginLogwsProxy();
         LogwsBean logws = logwsProxy.perform(userid, password);
         out.println("<tr><td>User ID:</td><td>"+logws.getUserid()+"</td></tr>");
         out.println("<tr><td>Pasword:</td><td>"+logws.getPassword()+"</td></tr>");
         out.println("<tr><td>Action: </td><td>"+logws.getAction()+"</td></tr>");
         out.println("<tr><td>Status: </td><td>"+logws.getStatus()+"</td></tr>");
         out.println("</TBODY></TABLE>");
         if (logws.getStatus().trim().equals("1") )
            out.println("<p>Login successful");
         else out.println("<p>Login failed");
      }
      out.println("</body></html>");
   } catch (Exception ex) { ex.printStackTrace();
      out.println("<p>Error in Login Web Service: "+ex.getMessage());
   }
}}
```

Figure 12-18   Web service client servlet

Test the Web service client

Start the `StrutsServer`, select the `LoginClient.html` file and *Run on Server*. Enter `uid:1` and `xxx` and run either of the Web services. The output of a sample run is shown in Figure 12-19.



Web Browser
http://localhost:9080/ItsoMyTradeWebClient/LoginClient.html
LoginLogws
Login
Done
Web Browse
http://localhost:9080/ItsoMyTradeWebCl
Web Browser
http://localhost:
Done
Done

Web Service Login Client

Enter the user ID: uid:1

Enter the password: ***

Web Service Login

Login and display record

User ID:	uid:1
Pasword:	xxx
Action:	inquire
Status:	1

Login successful

Web Service Login Client

Login and display status

Login successful

Figure 12-19 Web service client application run

Using a Web service in a Struts action

A Struts action or model class can use a Web service is the same way the sample client servlet uses a Web service.

Such an action would only make sense when the Web service is remote, that is, running in a different server. Calling one of our own Web services in the same server would only add overhead to the process.

To call a Web service in a Struts action, use similar code as in the client servlet:

```
LoginLogwsProxy logwsProxy = new LoginLogwsProxy();
LogwsBean logws = logwsProxy.perform(userid, password);
```

From this example, you realize that you require a proxy class and you must know the data type that is returned.

Outline of required actions

To generate a proxy class for a Web service, you require the description of the Web service, that is, the WSDL files.

The general sequence of actions is as follows:

- Have a Web project ready from which you want to call the Web service.
- Get the WSDL files. You require the complete set of files: the specification, the binding, the service implementation, and the XML schema of the result.

  For our LoginLogws example:

  ```
  LoginLogws.wsdl
  LoginLogwsBinding.wsdl
  LoginLogwsService.wsdl
  LogwsBean.xsd
  ```

  You would get these files either from the service provider directly, or by using a UDDI registry that has pointers to the files.
- Run the Web service wizard from the LoginLogwsService.wsdl implementation file to generate the proxy class.
- Use the proxy class in a Struts action.

Prepare a Web project

To illustrate the process, we will use our existing client project, ItsoMyTradeWebClient.

Get the WSDL files

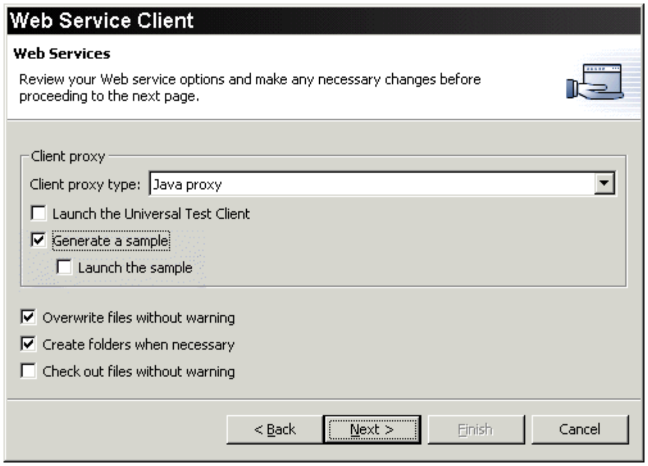
It does not matter how you get the WSDL files. In our case, we just copy the files from the `ItsoMyTradeWeb` project to the `ItsoMyTradeWebClient` project:

- Create a folder named `wsdl` in `ItsoMyTradeWebClient\Web Content`.
- Select the files in `ItsoMyTradeWeb\Web Content\wsdl\webserv` and *Copy*.
- Select the `ItsoMyTradeWebClient\Web Content\wsdl` folder and *Paste*.

Generating a proxy class for a Web service

Run the Web service wizard to generate the proxy:

- Select the `LoginLogwsService.wsdl` file (in `ItsoMyTradeWebClient`) and *New -> Web Services -> Web Service Client*.
- In the dialog (Figure 12-20), all the defaults are fine. We can also generate a sample (but we do not require one). Click *Next*.


Web Service Client

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Client proxy

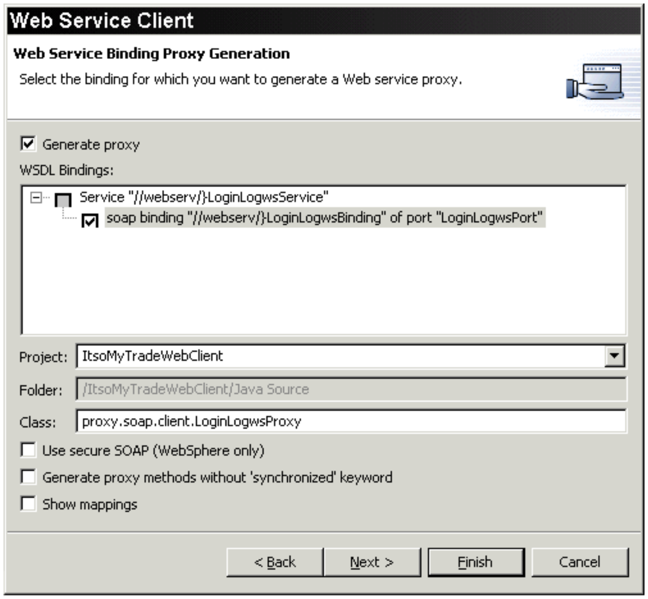
Client proxy type: Java proxy

☐ Launch the Universal Test Client
☑ Generate a sample
☐ Launch the sample

☑ Overwrite files without warning
☑ Create folders when necessary
☐ Check out files without warning

Figure 12-20 Web service client proxy generation

- For the WSDL file selection make sure that the `LoginLogwsService.wsdl` file is selected. Click *Next*.
- For Binding Proxy Generation (Figure 12-21), make sure that the `ItsoMyTradeWebClient` project is selected. Change the proxy class name to `proxy.soap.`**`client`**`.LoginLogwsProxy`. We do not want to overwrite the existing proxy class we used for testing. Click *Next*.



Web Service Client

Web Service Binding Proxy Generation

Select the binding for which you want to generate a Web service proxy.

☑ Generate proxy

WSDL Bindings:

- ☐ Service "//webserv/}LoginLogwsService"
  - ☑ soap binding "//webserv/}LoginLogwsBinding" of port "LoginLogwsPort"

Project: ItsoMyTradeWebClient

Folder: /ItsoMyTradeWebClient/Java Source

Class: proxy.soap.client.LoginLogwsProxy

☐ Use secure SOAP (WebSphere only)

☐ Generate proxy methods without 'synchronized' keyword

☐ Show mappings

Figure 12-21 Web service client proxy generation class

- Skip the universal test client and click *Next*.
- For Sample Generation, select *Generate Web service sample JSPs* and set the JSP folder as:

  /ItsoMyTradeWebClient/Web Content/sample/**client**/LoginLogws

  We do not want to overwrite the sample generated in “Create a Web service from the LoginLogws JavaBean” on page 326. (Actually the code is identical.)
- Click *Finish*.

Generated files

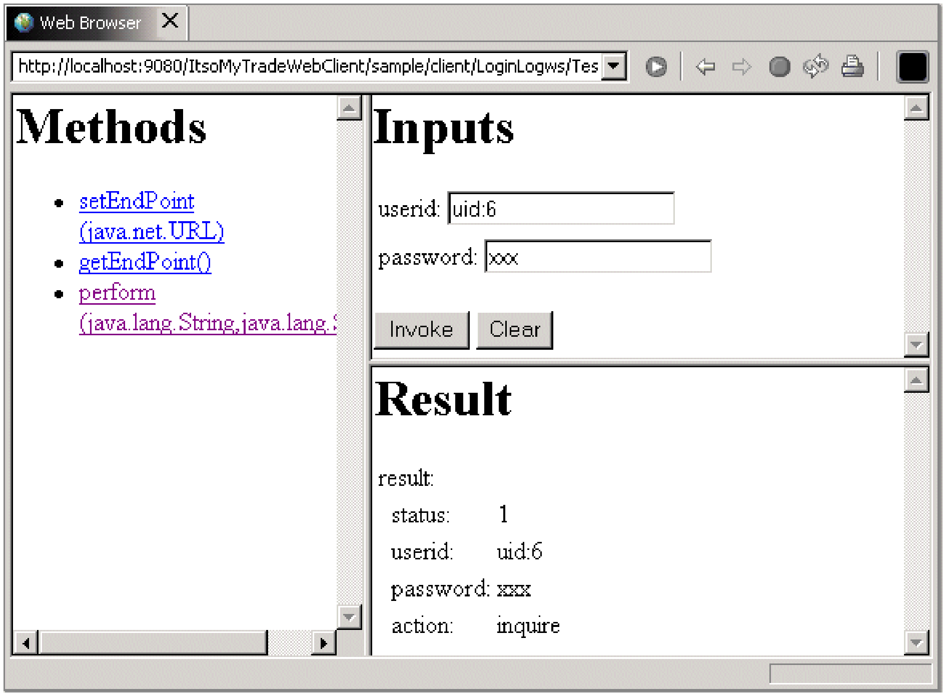
The generated files include:

- The proxy class `proxy.soap.client.LoginLogwsProxy`
- The sample client JSPs in `Web Content\sample\client\LoginLogws`
- The result JavaBean, `webserv.LogwsBean`, is copied to the project (in our case it is already there)

Running the sample client

Start the `StrutsServer`, select the `sample\client\LoginLogws\TestClient.jsp` and *Run on Server*. The sample client is displayed.

Select the `perform` method, enter appropriate user ID and password, and click *Invoke*. A sample run is shown in Figure 12-22.


Web Browser
http://localhost:9080/ItsoMyTradeWebClient/sample/client/LoginLogws/Tes
Invoke
Clear

Methods

- setEndPoint (java.net.URL)
- getEndPoint()
- perform (java.lang.String,java.lang.S

Inputs

userid: uid:6

password: xxx

Result

result:
status: 1
userid: uid:6
password: xxx
action: inquire

Figure 12-22 Web service sample client run

Use the proxy bean in a Struts action

This step is not illustrated here.

Once you have a proxy for a Web service, invocation of the Web service becomes very easy. Just allocate the proxy class, call the method with appropriate parameters, and retrieve the result.

The result can be a simple Java type (`int`, as illustrated in the `Login` example), a JavaBean (`LogwsBean`, as illustrated in the `LoginLogws` example), an array of JavaBeans, or an XML document in memory (an `org.w3c.dom.Element`).

13

Deploying applications

In this chapter we discuss the deployment of the sample application into a WebSphere Application Server.

Part of the deployment is the configuration of the WebSphere Application Server. For our purposes we only use the base server, one server in one machine (node).

Deployment steps

Deployment of the sample application includes these activities:

- Creating the enterprise application archive (EAR) file from the Enterprise Developer
- Configuring the WebSphere Application Server for DB2 and CICS resource adapters
- Installing the enterprise application in a WebSphere Application Server

Creating the EAR file

Make sure that the enterprise application and its modules are error free and that the deployment descriptor of the `ItsoMyTradeWeb` project is configured for the J2C resource (see "Configuring the Web application for J2C" on page 303).

Select *File -> Export -> EAR file*. In the export dialog select the `ItsoMyTradeEAR` application. For the destination enter any location, for example:

```
d:\itsomytrade.ear
```

Do not select any other options and click *Finish*.

Copy the file to the WebSphere Application Server directory for installable applications on the server machine, for example:

```
d:\WebSphere\AppServer\installableApps
```

Configuring the WebSphere Application Server

In this section, we configure a WebSphere Application Server Version 5 for our sample application. We assume that a Version 5 server has been installed on a server machine.

Start the server and the administrative console

Start the WebSphere server (select *Start -> Programs -> IBM WebSphere -> Application Server V5.0 -> Start the server*).

Start the administrative console (select *Start -> Programs -> IBM WebSphere -> Application Server V5.0 -> Administrative Console*). This opens a browser with:

```
http://127.0.0.1:9090/admin
```

You can also start the administrative console from another machine by pointing to:

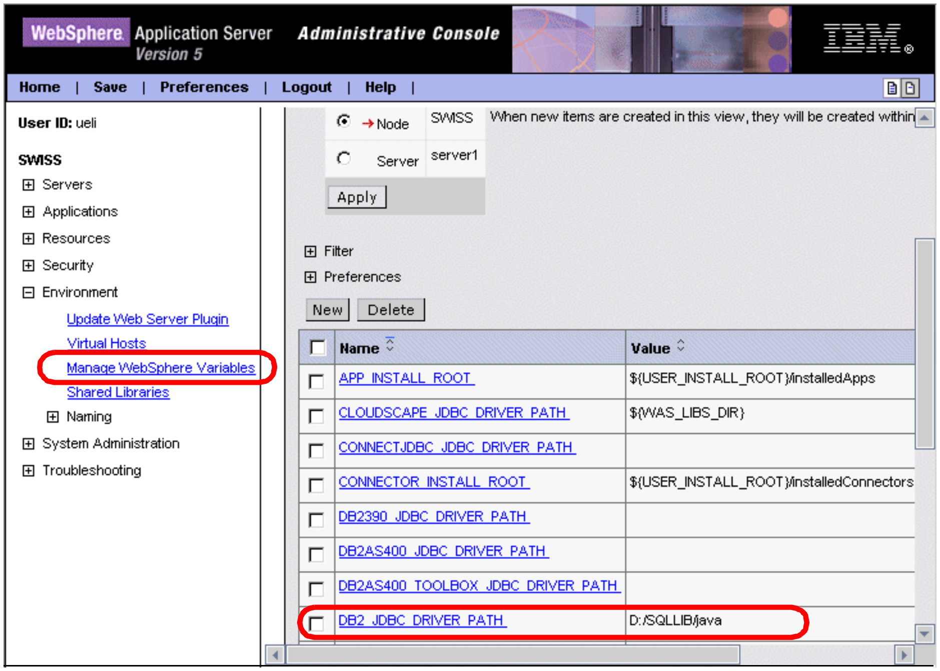
```
http://yourservermachine:9090/admin
```

You are prompted for a user ID that is used to track the changes. This section assumes that security has not been enabled and that we only have one node with one server called `server1`.

Configuring the data source for the TRADEDB

When configuring a data source, environment variables are used to point to installation directories. First we check that a variable is configured pointing to the DB2 JDBC directory:

Expand *Environment* and select *Manage WebSphere Variables*. For the Node (top), scroll down to the variable *DB2 JDBC Driver Path* and make sure that it points to the `java` subdirectory of the DB2 installation. If the variable value is empty, select the *DB2 JDBC Driver Path* variable and enter the correct path in the dialog that is displayed. Click *OK* and save the configuration.



● Node	SWISS	When new items are created in this view, they will be created within
○ Server	server1	

□	Name	Value
□	APP_INSTALL_ROOT	${USER_INSTALL_ROOT}/installedApps
□	CLOUDSCAPE_JDBC_DRIVER_PATH	${WAS_LIBS_DIR}
□	CONNECTJDBC_JDBC_DRIVER_PATH	
□	CONNECTOR_INSTALL_ROOT	${USER_INSTALL_ROOT}/installedConnectors
□	DB2390_JDBC_DRIVER_PATH	
□	DB2AS400_JDBC_DRIVER_PATH	
□	DB2AS400_TOOLBOX_JDBC_DRIVER_PATH	
□	DB2_JDBC_DRIVER_PATH	D:/SQLLIB/java

Figure 13-1 WebSphere variable for DB2 JDBC drivers

Define a DB2 JDBC driver

Expand *Resources* and select *JDBC Providers*. For the scope (top), select *Server* and click *Apply*.

If you do not have a DB2 JDBC Provider, click *New*. Select *DB2 JDBC Provider* from the pull-down and click *OK*. In the dialog, check that the classpath points to `${DB2_JDBC_DRIVER_PATH}/db2java.zip` and the implementation class points to `COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource`. Click *OK*.

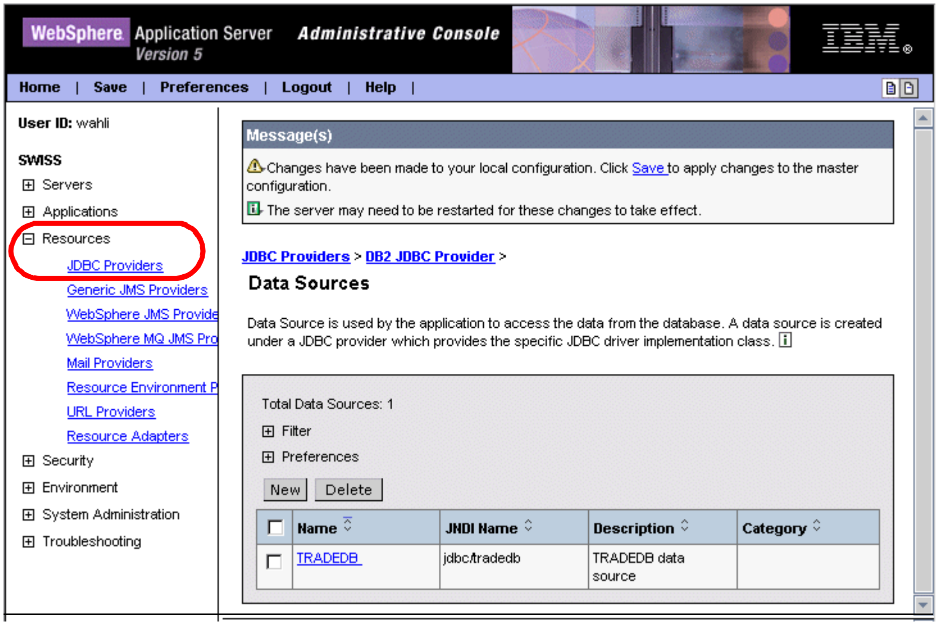
Define the data source for the TRADEDB

Click the new or existing DB2 JDBC Provider. In the dialog, scroll down and select *Data Sources*.

Click *New* to define a data source. Enter `TRADEDB` as the name, `jdbc/tradedb` as the JNDI name, and optionally include a description. Check that the data store helper class name is `com.ibm.websphere.rsadapter.DB2DataStoreHelper`. Click *Apply*.

The dialog is expanded at the bottom. Click *Custom Properties*. Click *databaseName* and enter `TRADEDB` as value (the other properties are optional). Select *Data Sources* in the top line.

Figure 13-2 shows the `TRADEDB` data source defined. Save the configuration.



Message(s)

⚠ Changes have been made to your local configuration. Click Save to apply changes to the master configuration.

ℹ The server may need to be restarted for these changes to take effect.

JDBC Providers > DB2 JDBC Provider >

Data Sources

Data Source is used by the application to access the data from the database. A data source is created under a JDBC provider which provides the specific JDBC driver implementation class. ℹ

Total Data Sources: 1

⊞ Filter

⊞ Preferences

□	Name	JNDI Name	Description	Category
□	TRADEDB	jdbc/tradedb	TRADEDB data source	

Figure 13-2 Defining the TRADEDB data source

Configuring the J2C connector

To configure the J2C connector, we have to install the resource adapter archive file (RAR) and then define the connection to CICS.

The resource adapter archive file for CICS is `cicseci.rar`. You can get that file from the CICS Transaction Gateway or from the Enterprise Developer at:

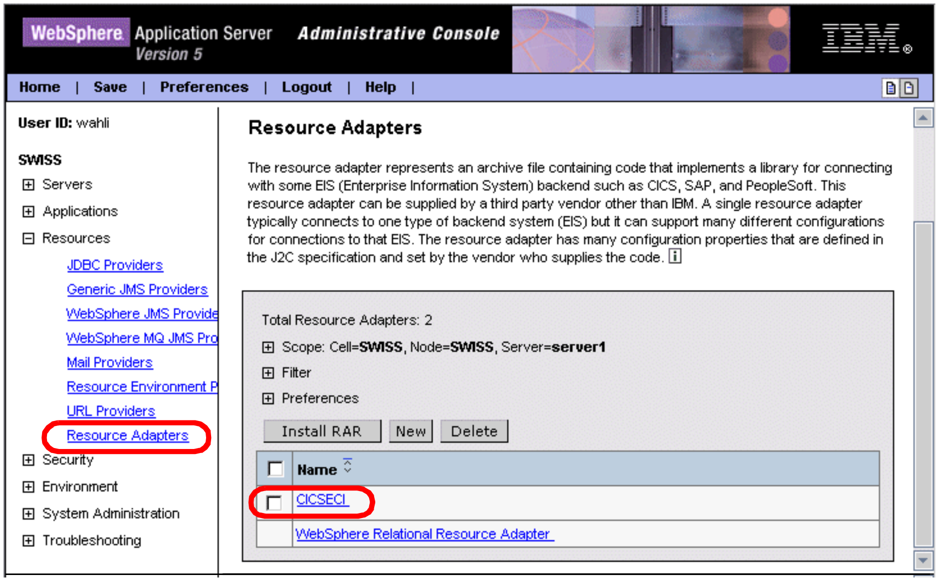
```
<WSED-Home>\wstools\eclipse\plugins\com.ibm.etools.ctc.binding.eis_0.5.0\
                                                                   runtime
```

Install the resource adapter archive (RAR) file

Select *Resources -> Resource Adapters*. For the scope, select *Server* and click *Apply*. Click *Install RAR*.

Select *Local path* and click *Browse*. Locate and select the `cicseci.rar` file. Click *Next*. Enter `CICSECI` as the name and click *OK*.

The `CICSECI` adapter appears in the list (Figure 13-3).



Resource Adapters

The resource adapter represents an archive file containing code that implements a library for connecting with some EIS (Enterprise Information System) backend such as CICS, SAP, and PeopleSoft. This resource adapter can be supplied by a third party vendor other than IBM. A single resource adapter typically connects to one type of backend system (EIS) but it can support many different configurations for connections to that EIS. The resource adapter has many configuration properties that are defined in the J2C specification and set by the vendor who supplies the code.

Total Resource Adapters: 2

⊞ Scope: Cell=**SWISS**, Node=**SWISS**, Server=**server1**

⊞ Filter

⊞ Preferences

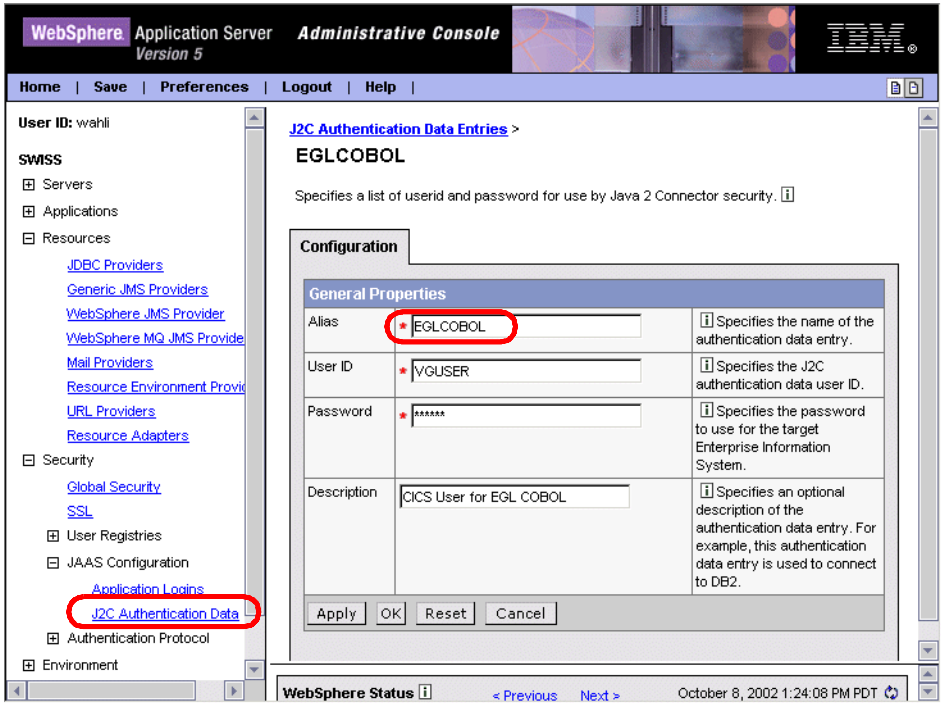
□	Name
□	CICSECI
	WebSphere Relational Resource Adapter

Figure 13-3 CICS ECI resource adapter

Configuring container-managed authentication

We have to configure authentication for the J2C connector for the container because we selected container authentication in the Web application (see “Configuring the Web application for J2C” on page 303).

Expand *Security -> JAAS Configuration -> J2C Authentication Data*. Click *New* to define a new user ID. Enter any alias name, for example, EGLCOBOL, and enter a user ID and password that are allowed to run the CICS EGL transaction (Figure 13-4).



J2C Authentication Data Entries >

EGLCOBOL

Specifies a list of userid and password for use by Java 2 Connector security.

Alias	* EGLCOBOL	Specifies the name of the authentication data entry.
User ID	* VGUSER	Specifies the J2C authentication data user ID.
Password	* ******	Specifies the password to use for the target Enterprise Information System.
Description	CICS User for EGL COBOL	Specifies an optional description of the authentication data entry. For example, this authentication data entry is used to connect to DB2.

October 8, 2002 1:24:08 PM PDT

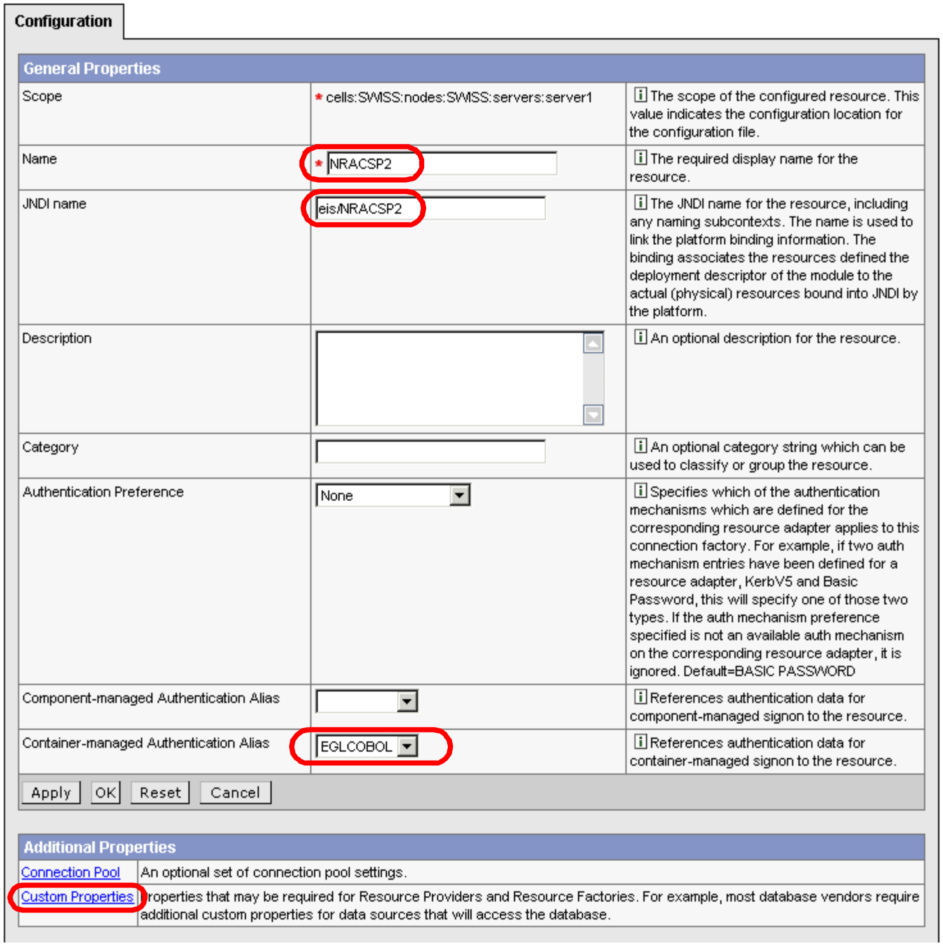
Figure 13-4 J2C authentication

Configuring the CICS ECI resource adapter

Expand *Resources - Resource Adapters* and select the *CICSECI* resource adapter. At the bottom of the dialog, click *J2C Connection Factories* (under the heading Additional Properties).

Click *New* to define a connection factory. In the dialog (Figure 13-5):

- Enter `NRACSP2` as the display name.
- Enter `eis/NRACSP2` as the JNDI name (to match the Web application).
- For the authentication preference, select *None*.
- For the Container-managed Authentication Alias, select `EGLCOBOL` from the pull-down.
- Click *Apply*.



General Properties

Scope	* cells:SWSS:nodes:SWSS:servers:server1	The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* NRACSP2	The required display name for the resource.
JNDI name	eis/NRACSP2	The JNDI name for the resource, including any naming subcontexts. The name is used to link the platform binding information. The binding associates the resources defined the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.
Description		An optional description for the resource.
Category		An optional category string which can be used to classify or group the resource.
Authentication Preference	None	Specifies which of the authentication mechanisms which are defined for the corresponding resource adapter applies to this connection factory. For example, if two auth mechanism entries have been defined for a resource adapter, KerbV5 and Basic Password, this will specify one of those two types. If the auth mechanism preference specified is not an available auth mechanism on the corresponding resource adapter, it is ignored. Default=BASIC PASSWORD
Component-managed Authentication Alias		References authentication data for component-managed signon to the resource.
Container-managed Authentication Alias	EGLCOBOL	References authentication data for container-managed signon to the resource.

Additional Properties

Connection Pool	An optional set of connection pool settings.
Custom Properties	roperties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database.

Figure 13-5 J2c connection factory

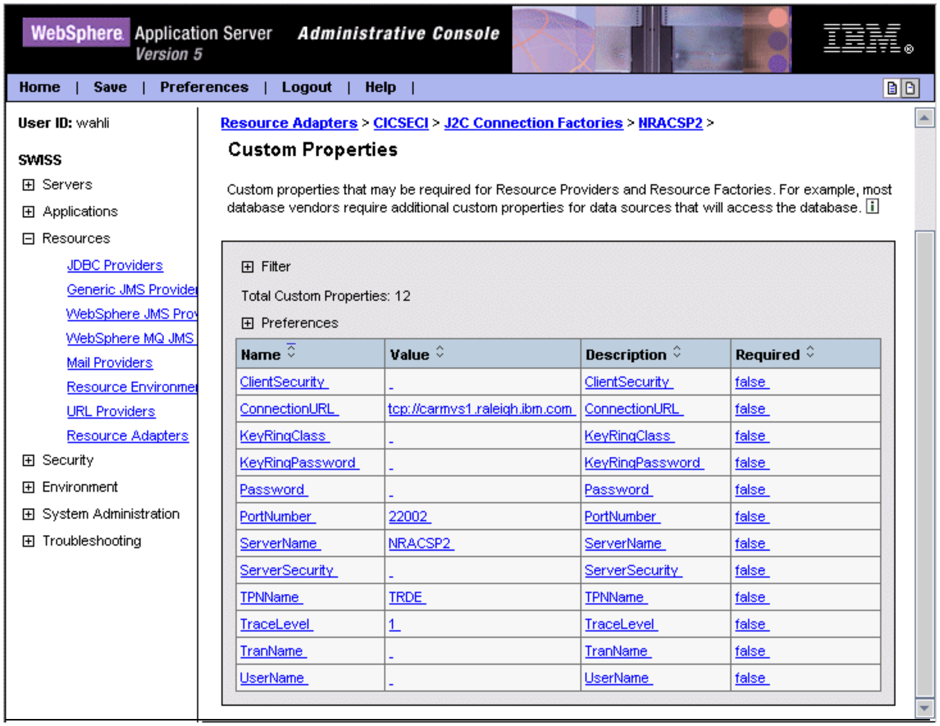
Select *Custom Properties* at the bottom. Select these properties and define their values (Figure 13-6). Save the configuration when done.

ConnectionUrl `tcp://carmvs1.raleigh.ibm.com`—your CICS Transaction Gateway (CTG) node

PortNumber `22002`—the port of the CTG

ServerName `NRACSP2`—name of the CICS server

TPNName `TRDE`—CICS transaction for the DB2 plan



Custom Properties

Custom properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database.

Total Custom Properties: 12

Name	Value	Description	Required
ClientSecurity		ClientSecurity	false
ConnectionURL	tcp://carmvs1.raleigh.ibm.com	ConnectionURL	false
KeyRingClass		KeyRingClass	false
KeyRingPassword		KeyRingPassword	false
Password		Password	false
PortNumber	22002	PortNumber	false
ServerName	NRACSP2	ServerName	false
ServerSecurity		ServerSecurity	false
TPNName	TRDE	TPNName	false
TraceLevel	1	TraceLevel	false
TranName		TranName	false
UserName		UserName	false

Figure 13-6 Properties of the CICS ECI connector

Note that *UserName* and *Password* are not required because we selected container authentication. These properties can be specified when selecting component authentication and they overwrite any specification done for the connection factory Component-managed Authentication Alias.

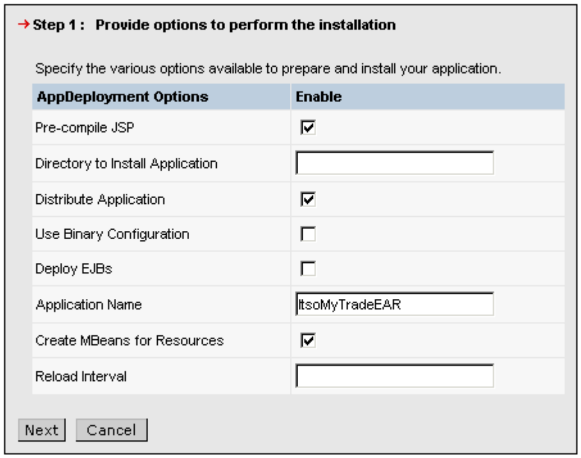
Installing the enterprise application

In this section, we install the `ItsoMyTradeEAR` application into the application server.

In the administrative console, expand Applications and select *Install New Application*:

- In the dialog, select *Local path* and click *Browse* to locate the `itsomytrade.ear` file (for example, `WebSphere\AppServer\installableApps`). Click *Next*.
- Leave all defaults on the second preparation panel and click *Next*.

- Step 1: Provide installation options. Select *Pre-compile JSP* and leave all other defaults. (EJBs have already been deployed in the Enterprise Developer.)

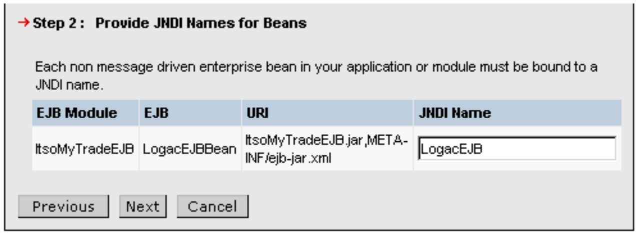


→ **Step 1 : Provide options to perform the installation**

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	☑
Directory to Install Application	
Distribute Application	☑
Use Binary Configuration	□
Deploy EJBs	□
Application Name	ItsoMyTradeEAR
Create MBeans for Resources	☑
Reload Interval	

- Step 2: JNDI names for beans (no change). The JNDI name of the session EJB generated by EGL is visible.

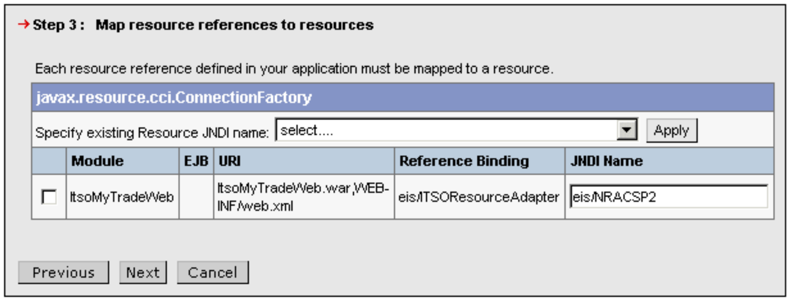


→ **Step 2 : Provide JNDI Names for Beans**

Each non message driven enterprise bean in your application or module must be bound to a JNDI name.

EJB Module	EJB	URI	JNDI Name
ItsoMyTradeEJB	LogacEJBBean	ItsoMyTradeEJB.jar,META-INF/ejb-jar.xml	LogacEJB

- Step 3: Map resource references (no change). The reference of the J2C connection and JNDI name defined in the `web.xml` file is visible.



→ **Step 3 : Map resource references to resources**

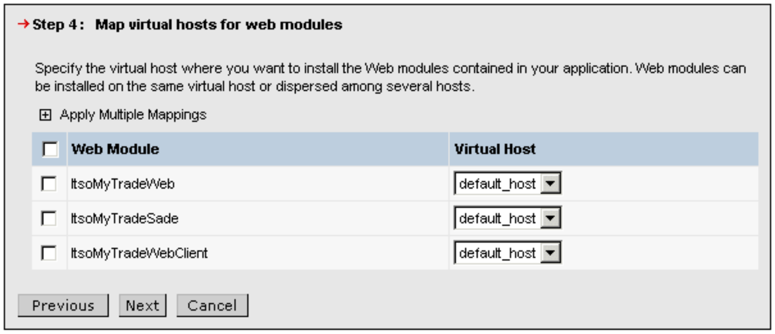
Each resource reference defined in your application must be mapped to a resource.

javax.resource.cci.ConnectionFactory

Specify existing Resource JNDI name: select....

	Module	EJB	URI	Reference Binding	JNDI Name
□	ItsoMyTradeWeb		ItsoMyTradeWeb.war,WEB-INF/web.xml	eis/ITSOResourceAdapter	eis/NRACSP2

- Step 4: Map virtual hosts for Web modules (no change). All modules map to *default_host*.



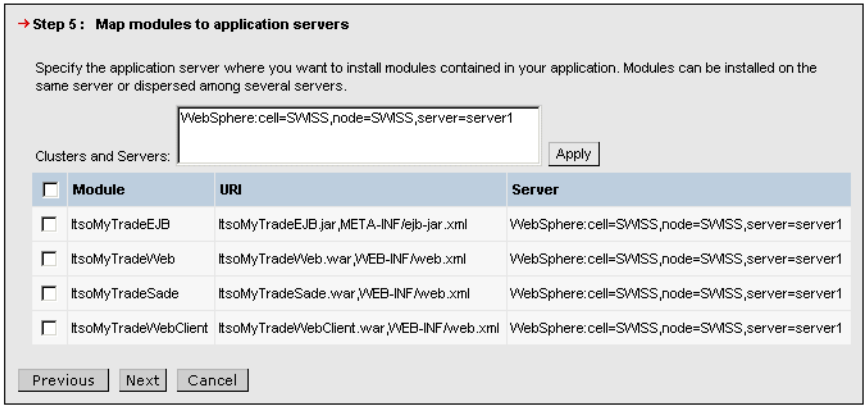
→ **Step 4 : Map virtual hosts for web modules**

Specify the virtual host where you want to install the Web modules contained in your application. Web modules can be installed on the same virtual host or dispersed among several hosts.

⊞ Apply Multiple Mappings

□	Web Module	Virtual Host
□	ItsoMyTradeWeb	default_host
□	ItsoMyTradeSade	default_host
□	ItsoMyTradeWebClient	default_host

- Step 5: Map modules to application servers (no change). All modules run on the only server (`server1`).



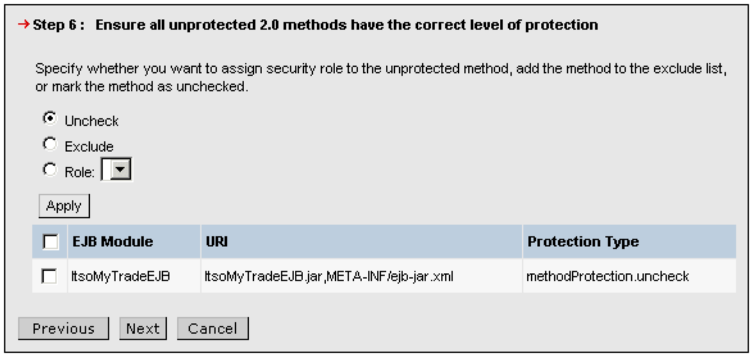
→ **Step 5 : Map modules to application servers**

Specify the application server where you want to install modules contained in your application. Modules can be installed on the same server or dispersed among several servers.

Clusters and Servers: WebSphere:cell=SWISS,node=SWISS,server=server1

□	Module	URI	Server
□	ItsoMyTradeEJB	ItsoMyTradeEJB.jar,META-INF/ejb-jar.xml	WebSphere:cell=SWISS,node=SWISS,server=server1
□	ItsoMyTradeWeb	ItsoMyTradeWeb.war,WEB-INF/web.xml	WebSphere:cell=SWISS,node=SWISS,server=server1
□	ItsoMyTradeSade	ItsoMyTradeSade.war,WEB-INF/web.xml	WebSphere:cell=SWISS,node=SWISS,server=server1
□	ItsoMyTradeWebClient	ItsoMyTradeWebClient.war,WEB-INF/web.xml	WebSphere:cell=SWISS,node=SWISS,server=server1

- Step 6: Method protection (no change).



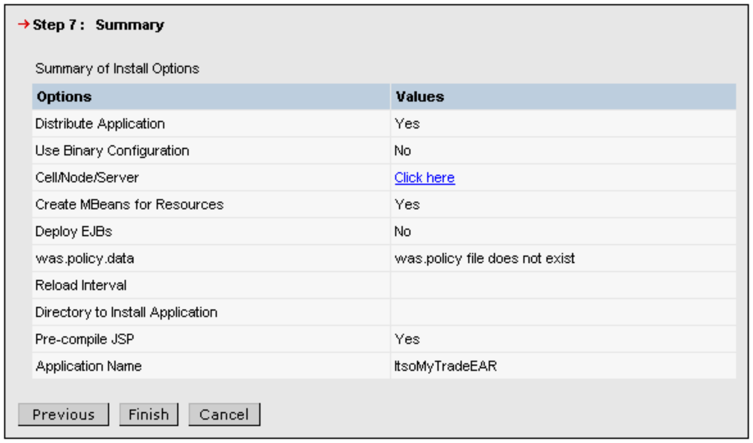
→ **Step 6 : Ensure all unprotected 2.0 methods have the correct level of protection**

Specify whether you want to assign security role to the unprotected method, add the method to the exclude list, or mark the method as unchecked.

● Uncheck
○ Exclude
○ Role:

□	EJB Module	URI	Protection Type
□	ItsoMyTradeEJB	ItsoMyTradeEJB.jar,META-INF/ejb-jar.xml	methodProtection.uncheck

- Step 7: Summary (no change).



→Step 7 : Summary

Summary of Install Options

Options	Values
Distribute Application	Yes
Use Binary Configuration	No
Cell/Node/Server	Click here
Create MBeans for Resources	Yes
Deploy EJBs	No
was.policy.data	was.policy file does not exist
Reload Interval	
Directory to Install Application	
Pre-compile JSP	Yes
Application Name	ItsoMyTradeEAR

- Click *Finish*. Wait for the message *Application ItsoMyTradeEAR installed successfully*. Click *Save to master configuration*.
- Stop the server and then start the server.

Setting up the TRADEDB database

To test the application on an application server, you require at least the TRADEREGISTRYBEAN table in the TRADEDB database with the user IDs loaded.

To load the table with the sample data, you require the exported data. Copy the traderegistrybean.ixf file from:

```
<WSED>\wstools\eclipse\plugins\com.ibm.etools.examples.trade_x.x.x\scripts
```

Run these commands in a DB2 command window to define and load the database and table:

```
db2 create database tradedb
db2 connect to tradedb
db2 create table db2admin.traderegistrybean
   ( userid varchar(251) not null, password varchar(251), status integer,
          primary key (userid))
db2 import from traderegistrybean.ixf of ixf insert into
       db2admin.traderegistrybean
db2 connect reset
```

If you used a user ID other than db2admin in your application, be sure to change the prefix in the commands.

Testing the Web application

Start a browser with the URL of the Web application:

```
http://servmachine:9080/MyTrade
```

Note that we can use the HTTP server that is built into WebSphere Application Server. To use an external HTTP server, we have to regenerate the plug-in and stop/start the HTTP server.

In the welcome page enter values into the bottom form to execute the EGL program in different configurations:

```
uid:1       Struts -> Java wrapper -> EGL program
uid:11      Struts -> Java wrapper -> session EJB -> EGL program
uid:111     Struts -> session EJB -> EGL program
uid:222     Struts -> Java wrapper -> J2C -> CICS COBOL EGL program
```

Part 4

Development environment for z/OS

In Part 4, we describe the z/OS development environment that is provided by the Enterprise Developer.

14

Developing for z/OS

This chapter discusses and illustrates the functionality behind the perspectives that are geared at supporting the development of artifacts that eventually will reside on z/OS.

We illustrate the functionality using a COBOL sample. The focus is on the features offered within Enterprise Developer to support the remote-edit-compile-debug (RECD) scenario. However, we also illustrate some of the local capabilities that are instrumental in support of the RECD scenario.

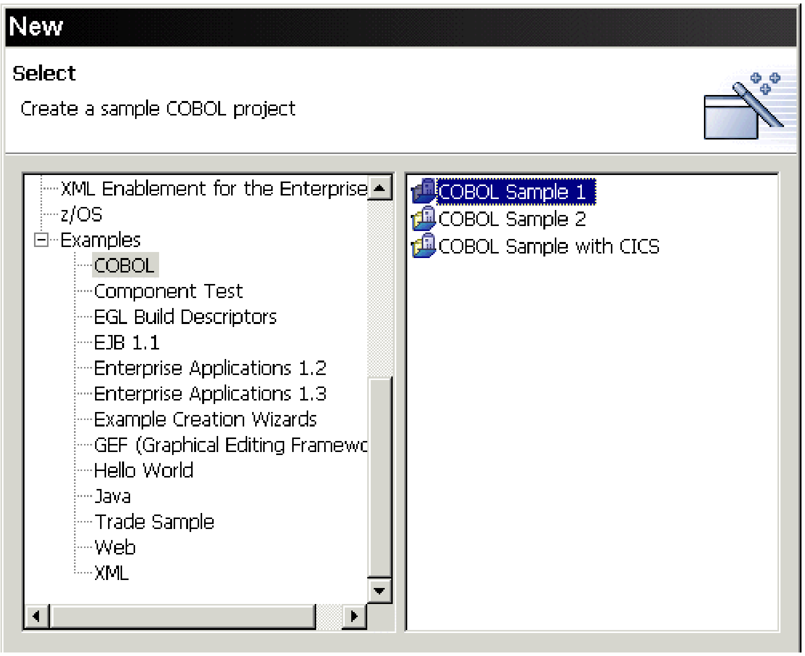
Local project

In this section we will create, load and work with a local project. This local project will subsequently move to z/OS to illustrate the RECD capabilities. It is likely that z/OS customer would use Enterprise Developer to maintain existing systems residing on the mainframe. Therefore, we will not discuss the local capabilities at great length.

Creating a local project

Starting from a z/OS Projects perspective, you create a (local) project using sample code that is shipped with Enterprise Developer by performing these steps:

- Select *File -> New -> Other*. The New wizard dialog opens (Figure 14-1).



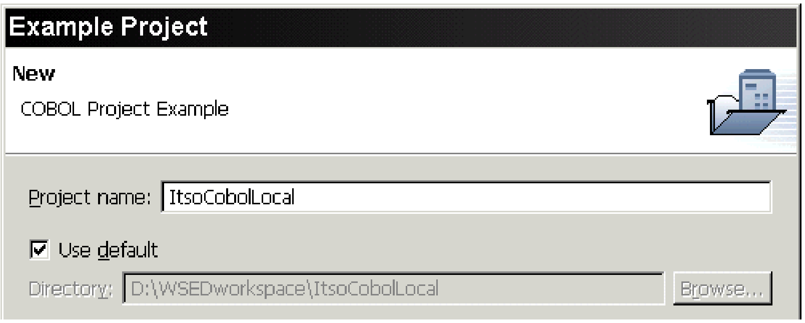
New

Select

Create a sample COBOL project

Figure 14-1 New Wizard dialog to create Cobol sample project

- Select *Examples -> COBOL* and in the right-hand pane *COBOL Sample1* and click *Next*, after which the COBOL sample project dialog is displayed (Figure 14-2).



Example Project

New

COBOL Project Example

Project name: ItsoCobolLocal

☑ Use default

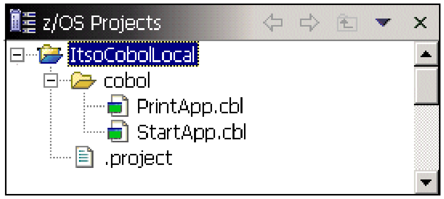
Directory: D:\WSEDworkspace\ItsoCobolLocal

Browse...

Figure 14-2   Cobol Sample project dialog

- ► Enter `ItsoCobolLocal` as the project name and accept the default workspace as the location where the project data will be stored.
- ► Click *Next*. For now we bypass the compile options and builder choices panels and click *Finish*.
- ► After processing has completed, the z/OS Projects view should look like Figure 14-3 and contain two COBOL sources and a `.project` file.

Important: You might have noticed that a `.project` file is created. Do not touch the `.project` files, which are part of each project.



z/OS Projects

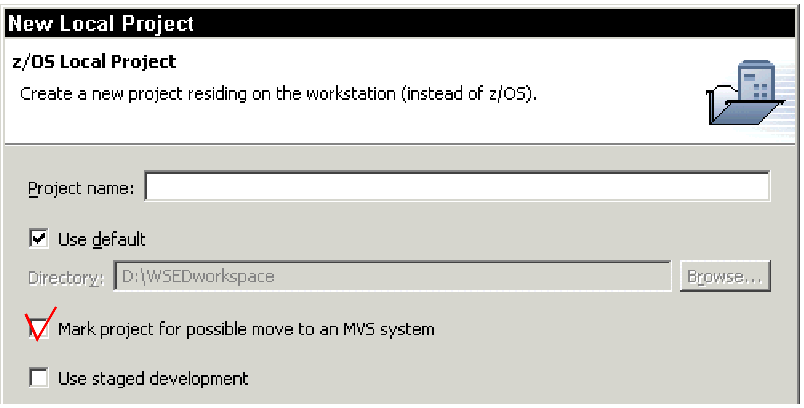
```
ItsoCobolLocal
    cobol
        PrintApp.cbl
        StartApp.cbl
    .project
```

Figure 14-3   z/OS project after creation

Tip: If you are interested only in the remote-edit-compile-debug (RECD) scenario, go directly to “Remote project introduction” on page 364.

Local project for possible move to MVS

When defining a new MVS local project from scratch, the user is offered an interesting option on the first panel (Figure 14-4).



New Local Project

z/OS Local Project

Create a new project residing on the workstation (instead of z/OS).

Project name:

☑ Use default

Directory: D:\WSEDworkspace

☐ Mark project for possible move to an MVS system

☐ Use staged development

Figure 14-4 Local project dialog for possible move to MVS

If you select *Mark project for possible move to an MVS syste*m, the Enterprise Developer enforces certain rules within that local project as if it were a remote project. An example would be that file names can only be 8 characters long.

Building the local project

In order to be able to run and debug the programs that are part of the `ItsoCobolLocal` project, we exploit the capabilities of a (local) distributed build server. For a local project, we use a build server on Windows (NT, 2000), which can be on the same machine or on another machine in the network.

Based upon a so-called build plan, which is generated when a rebuild for a project is requested, the build server does know what processes must be invoked in a particular order using the various sources and artifacts created during build processing.

For our projec, this means that the build plan must reflect that two compiles have to be performed and afterwards a link is performed to create an executable. Moreover, the build server will report back to the requesting build client what has been done and if requested brings the generated output artifacts (various listings) back into the project as well (Figure 14-5).

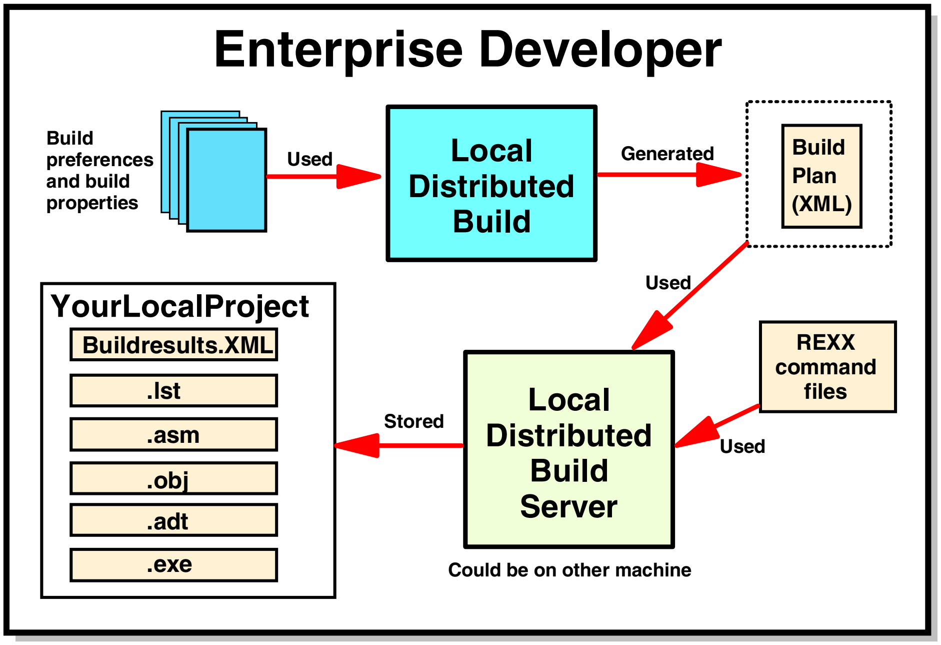


Figure 14-5 Local build server overview

Defining and starting a local build server

To start the build server, run this command in a Command Prompt window:

```
ccublds -p 2345 -V
```

Note that the system PATH must include the Enterprise Developer distributed build directory and the COBOL compile command:

```
d:<WSED>\wstools\eclipse\plugins\com.ibm.etools.egl.distributedbuild_5.0.0\bin
d:\IBM\vac\BIN            <=== this is missing in the PATH
```

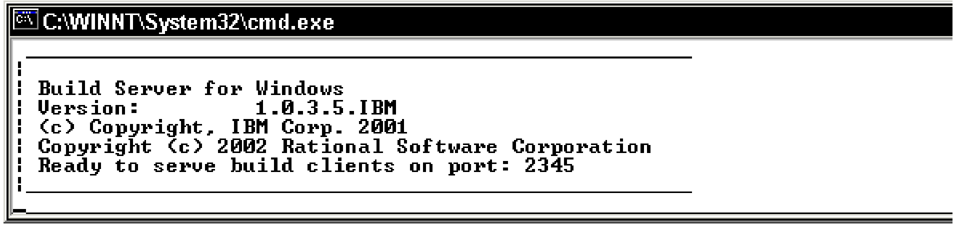
You can also create a command file to set the PATH and start the build server (Figure 14-6).

```
set path=D:\WSED\wstools\eclipse\plugins\
     com.ibm.etools.egl.distributedbuild_5.0.0\bin;D:\IBM\vac\BIN;%path%
ccublds -p 2345 -V
```

Figure 14-6 Command to start the local build server

Note that the port 2345 is just a number. As long as this number is not in use as the port for another process, one can choose any number. In the services file (`c:\winnt\system32\drivers\etc\services`), you can see which port numbers are potentially in use. We suggest you update this file to reflect the fact that 2345 will be used for the local build server.

After starting the build server, the command window should look like Figure 14-7. The build server is waiting for work!

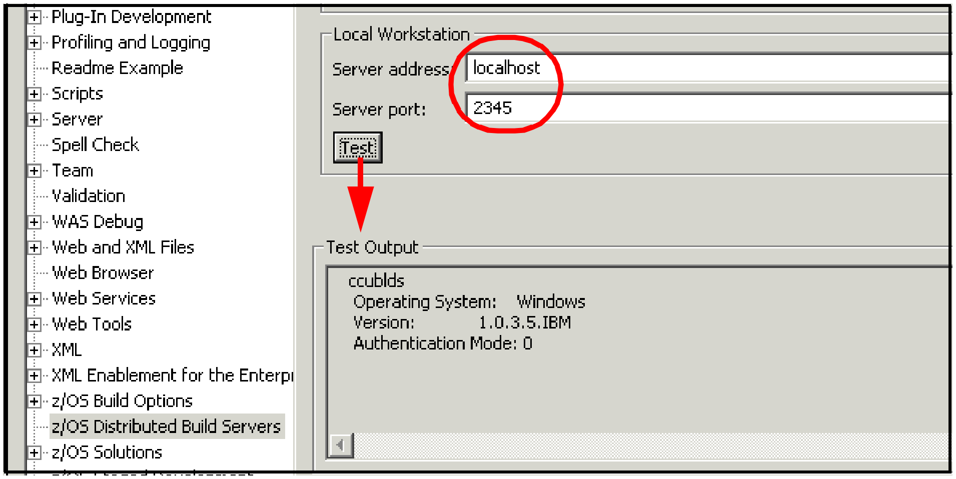


```
 Build Server for Windows
 Version:           1.0.3.5.IBM
 (c) Copyright, IBM Corp. 2001
 Copyright (c) 2002 Rational Software Corporation
 Ready to serve build clients on port: 2345
```

Figure 14-7 Local build server up and listening

When the build server is up and running, it should be defined to the development environment. The (local) build server is defined as follows:

- *Select Windows -> Preferences -> z/OS Distributed Build Servers* (Figure 14-8):



Server address: localhost

Server port: 2345

```
ccublds
 Operating System:   Windows
 Version:         1.0.3.5.IBM
 Authentication Mode: 0
```

Figure 14-8 Defining (local) build server in the preferences file

- For a local workstation, enter `localhost` as server address and `2345` as port.
- Click *Test* to verify that the server is running and click *OK* to save the preferences.

Performing the build

Now let's perform the build operation.

Fixing the system PATH

The build failed in our system because the system PATH does not include the COBOL compile command file d:\IBM\vac\BIN\iwzvcomp.cmd.

Important: Open the environment variables (select *Advanced* on system properties) and add the directory to the WSEDPATH variable:

```
D:\WSED\runtimes\base_v5\java\bin;D:\WSED\wstools\eclipse\plugins\com.ibm
.etools.egl.distributedbuild_5.0.0\bin;D:\IBM\vac\BIN;
```

Note that WSEDPATH is appended to the system PATH. You must reopen a command window and restart the build processor (ccublds).

Invoking the build

In order to create an accurate load module, the main COBOL programs must be flagged as being a MAIN program. In our case the StartApp program is a MAIN program. After that we can perform the build:

- Select StartApp.cbl and *Set As Main* from the context menu.
- Select the ItsoCobolLocal project and *Rebuild Project* from the context menu.

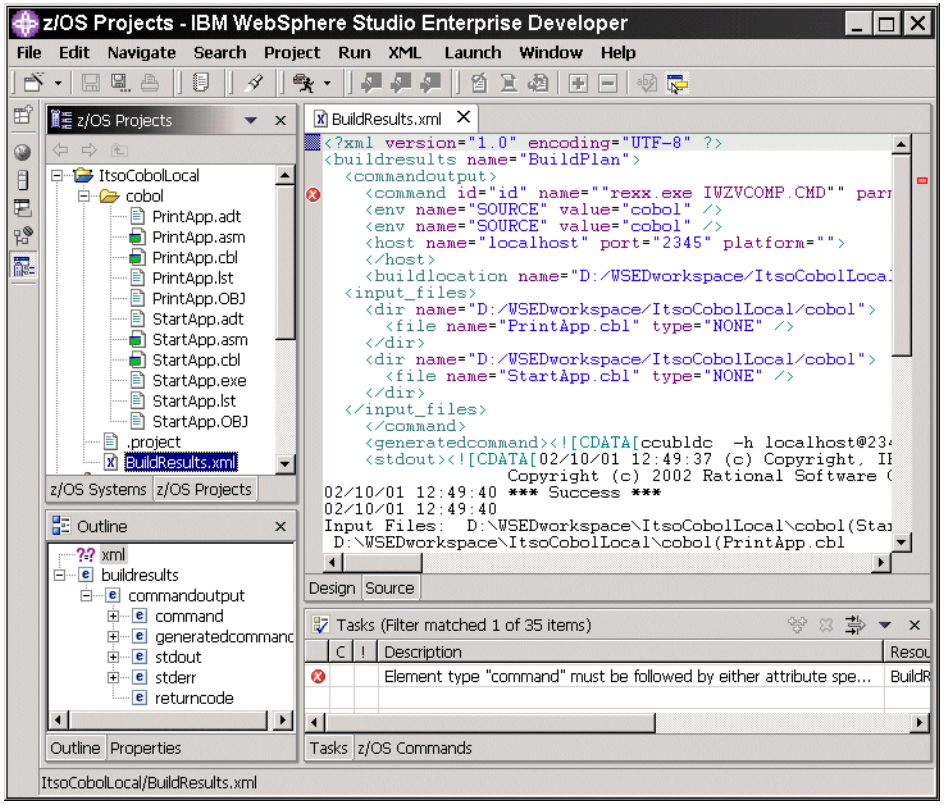
Checking the build output

After the build completes, all progress windows have closed, and after performing a refresh action on the project (select *Refresh* from the context menu), open the BuildResults.xml file to view the results of the processing (Figure 14-9).

Notice the generated files in the cobol folder:

- Compiler listings (.lst)
- Intermediate files (.adt, .asm, .OBJ)
- Executable (.exe)

Note that although an error is displayed for the XML file, the executable is in good shape. The Outline view is useful to navigate through the BuildResults.xml file.



```
<?xml version="1.0" encoding="UTF-8" ?>
<buildresults name="BuildPlan">
  <commandoutput>
    <command id="id" name=""rexx.exe IWZVCOMP.CMD"" parr
    <env name="SOURCE" value="cobol" />
    <env name="SOURCE" value="cobol" />
    <host name="localhost" port="2345" platform="">
    </host>
    <buildlocation name="D:/WSEDworkspace/ItsoCobolLoca]
  <input_files>
    <dir name="D:/WSEDworkspace/ItsoCobolLocal/cobol">
      <file name="PrintApp.cbl" type="NONE" />
    </dir>
    <dir name="D:/WSEDworkspace/ItsoCobolLocal/cobol">
      <file name="StartApp.cbl" type="NONE" />
    </dir>
  </input_files>
    </command>
    <generatedcommand><![CDATA[ccubldc  -h localhost@234
    <stdout><![CDATA[02/10/01 12:49:37 (c) Copyright, II
                  Copyright (c) 2002 Rational Software (
02/10/01 12:49:40 *** Success ***
02/10/01 12:49:40
Input Files:  D:\WSEDworkspace\ItsoCobolLocal\cobol(Sta1
 D:\WSEDworkspace\ItsoCobolLocal\cobol(PrintApp.cbl
```

C	!	Description	Resou
✕		Element type "command" must be followed by either attribute spe...	BuildR

Figure 14-9   COBOL project after build processing

The build output shows a successful completion of the build process:

```
02/10/01 12:49:40 *** Success ***
02/10/01 12:49:40
Input Files:  D:\WSEDworkspace\ItsoCobolLocal\cobol(StartApp.cbl
 D:\WSEDworkspace\ItsoCobolLocal\cobol(PrintApp.cbl
Command: rexx.exe IWZVCOMP.CMD -qTEST,LIST -b/de -PP&quot; &quot;
-main:StartApp PrintApp.cbl StartApp.cbl
***************** Build Script Output Follows *****************
PP 5639-I44 IBM VisualAge for COBOL (Windows)  3.0 in progress ...
End of compilation 1,  program PRINTAPP,  no statements flagged.
PP 5639-I44 IBM VisualAge for COBOL (Windows)  3.0 in progress ...
End of compilation 1,  program STARTAPP,  no statements flagged.
COBOL compile complete, return code = 0.
***************** End Of Build Script Output *****************
```

Alternative way to perform build

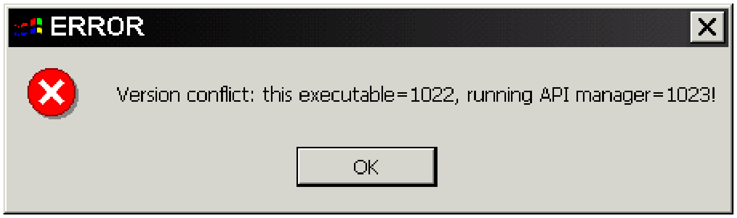
For a local project it is possible to perform build action (compile and link) from the z/OS Projects view by selecting a COBOL source and *Build* in the context menu.

A build started like this has a scope of only a single source file and does not exploit the distributed build architecture. Behind the build action is a REXX script issuing a command-line invocation of the distributed COBOL compiler (`cob2`) and linker.

> **Restriction:** Performing a build operation from the COBOL file context menu will only create an executable for stand-alone programs. The distributed build discussed earlier is capable of handling programs that call other programs. Note that a project is restricted to having only one main program.

Potential REXX conflict

When running builds or a syntax check, you may encounter a REXX version conflict if you already have Object REXX installed on your system. The error message shows up in a pop-up dialog as shown in Figure 14-10.



Version conflict: this executable=1022, running API manager=1023!

Figure 14-10 REXX version conflict

To resolve this conflict, open the Windows Task Manager, find the `rxapi.exe` process, and click *End Process* to kill that task. Then retry the operation.

Running the local project

To see what the sample program is all about, run the executable:

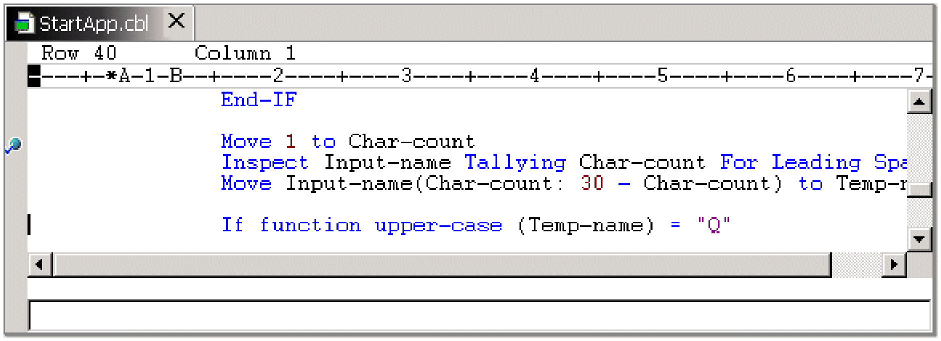
- Double-click `StartApp.exe` from the z/OS Project view and enter a name when prompted. A sample output is shown below:

```
Enter a name or Q to quit:
Gert Hekkenberg
Thanks to Gert Hekkenberg for succeeding!

Enter a name or Q to quit:
q
```

Debugging the local project

To debug the local project, we set a breakpoint in the COBOL program, open the Debug perspective, and start the debug session:

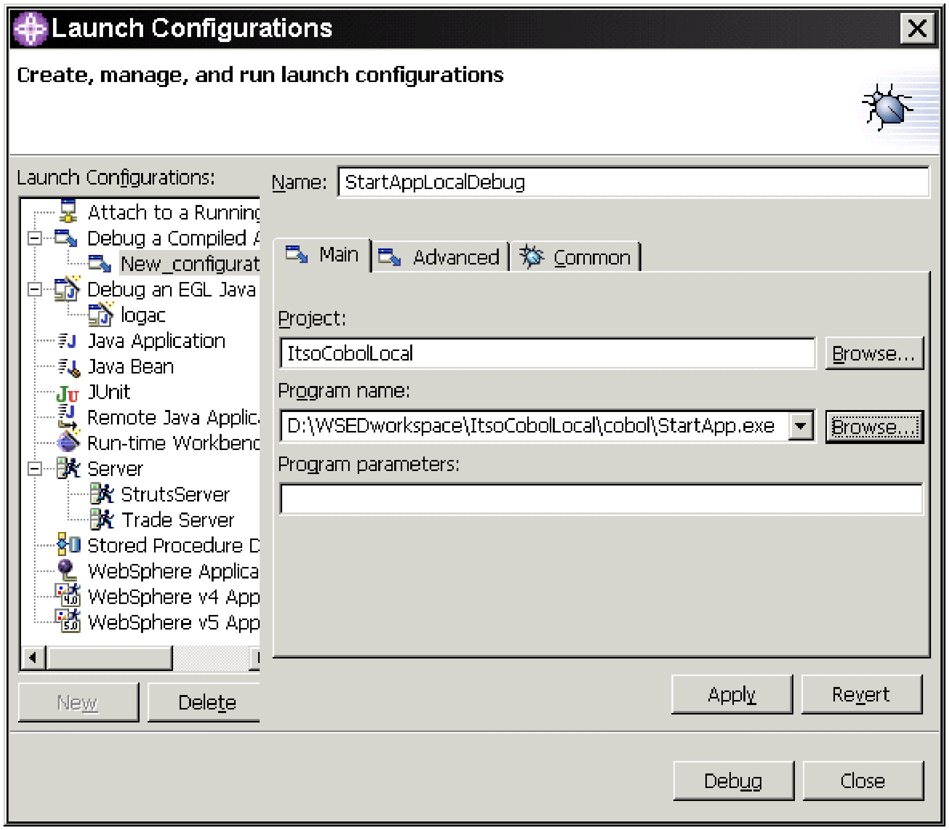
- Open the `StartApp.cbl` program. Scroll to the line `Move 1 to Char-count` and double-click in the grey left border to set the breakpoint (Figure 14-11).



```
Row 40      Column 1
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-
               End-IF

               Move 1 to Char-count
               Inspect Input-name Tallying Char-count For Leading Spa
               Move Input-name(Char-count: 30 - Char-count) to Temp-r

               If function upper-case (Temp-name) = "Q"
```

Figure 14-11 COBOL program with breakpoint set

- Open the Debug perspective by selecting *Window -> Open Perspective -> Other -> Debug* and click *OK* (or use the shortcut through the open perspectives icon).
- Now we start a debug session for debugging the COBOL program. Select *Run -> Debug* and the Launch Configuration dialog opens (Figure 14-12):
  - Select *Debug a Compiled Application*.
  - Click *New* to define a new configuration.
  - Overtype the name with `StartAppLocalDebug`.
  - For the project, click *Browse* and select the `ItsoCobolLocal` project.
  - For the program name, click *Browse* and navigate to the `StartApp.exe` in the workspace.
  - Click *Apply*.



Launch Configurations

Create, manage, and run launch configurations

Name: StartAppLocalDebug

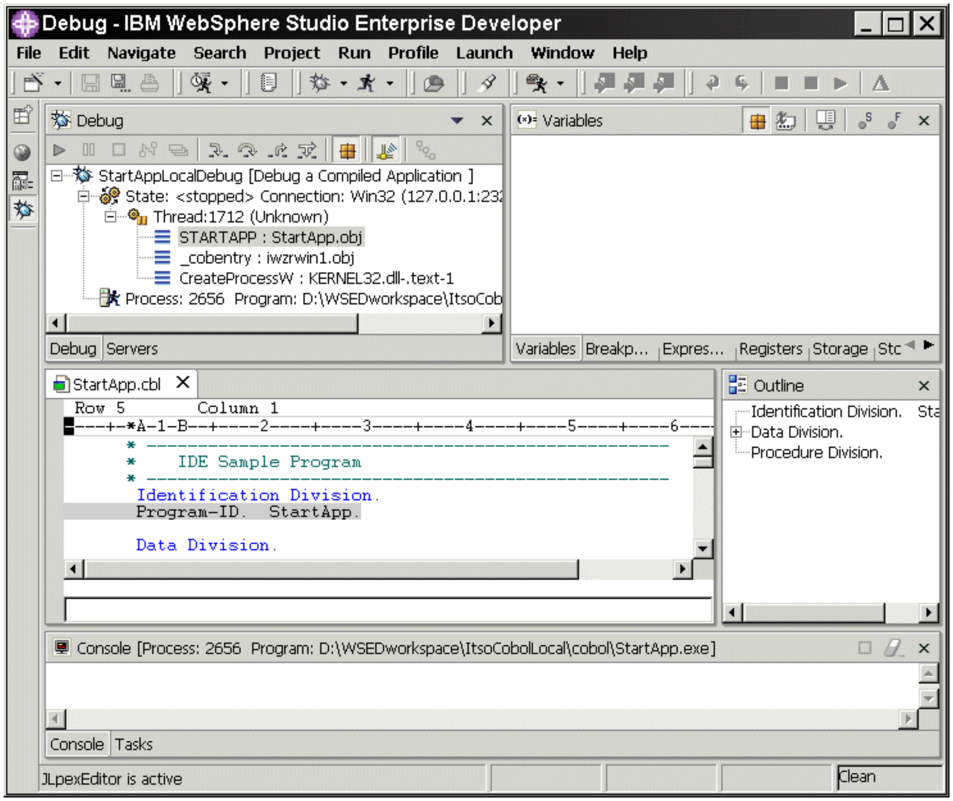
Project: ItsoCobolLocal

Program name: D:\WSEDworkspace\ItsoCobolLocal\cobol\StartApp.exe

Program parameters:

Figure 14-12 Launch debug configuration for local debug

- Click *Debug* to start debugging the COBOL program. The COBOL source opens in the Debug view at the start of the program (Figure 14-13).


Debug - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Run Profile Launch Window Help
Debug
Variables
Debug Servers
Variables Breakp... Expres... Registers Storage Stc
StartApp.cbl
Outline
Identification Division. Sta
Data Division.
Procedure Division.
Console Tasks
JLpexEditor is active
Clean

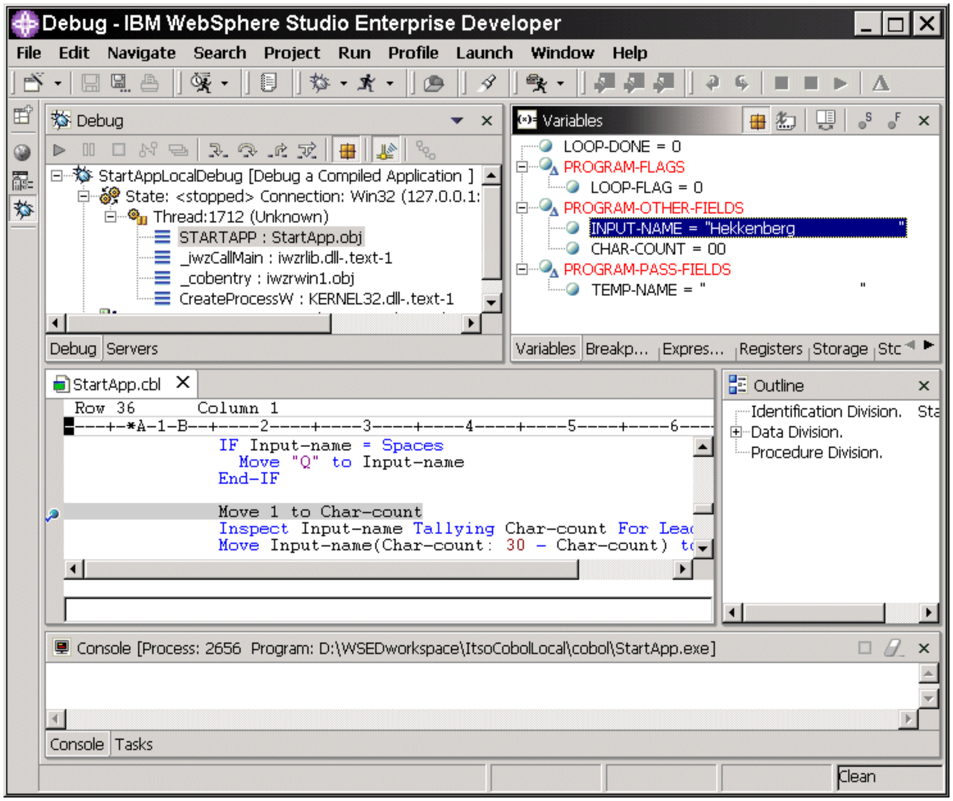
```
StartAppLocalDebug [Debug a Compiled Application ]
  State: <stopped> Connection: Win32 (127.0.0.1:23
    Thread:1712 (Unknown)
      STARTAPP : StartApp.obj
      _cobentry : iwzrwin1.obj
      CreateProcessW : KERNEL32.dll-.text-1
  Process: 2656  Program: D:\WSEDworkspace\ItsoCob
```

```
 Row 5            Column 1
---+-*A-1-B--+----2----+----3----+----4----+----5----+----6----
      * ----------------------------------------------------------
      *     IDE Sample Program
      * ----------------------------------------------------------
       Identification Division.
       Program-ID.  StartApp.

       Data Division.
```

Console [Process: 2656 Program: D:\WSEDworkspace\ItsoCobolLocal\cobol\StartApp.exe]

Figure 14-13 Debug perspective: Local Debug session started

- Note that the application has opened a command window. We have to enter input in the command window.
- Click the *Resume* icon ▶ in the Debug view to run the program to the breakpoint.
- Enter `Hekkenberg` as the name in the command window of the running program.
- Expand the program variables in the Variables view (Figure 14-14).


Debug - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Run Profile Launch Window Help
Debug
Variables
Debug Servers
Variables Breakp... Expres... Registers Storage Stc
StartApp.cbl
Outline
Identification Division.
Data Division.
Procedure Division.
Sta
Console Tasks
Clean

StartAppLocalDebug [Debug a Compiled Application]
State: <stopped> Connection: Win32 (127.0.0.1:
Thread:1712 (Unknown)
STARTAPP : StartApp.obj
_iwzCallMain : iwzrlib.dll-.text-1
_cobentry : iwzrwin1.obj
CreateProcessW : KERNEL32.dll-.text-1

LOOP-DONE = 0
PROGRAM-FLAGS
LOOP-FLAG = 0
PROGRAM-OTHER-FIELDS
INPUT-NAME = "Hekkenberg "
CHAR-COUNT = 00
PROGRAM-PASS-FIELDS
TEMP-NAME = " "

```
 Row 36          Column 1
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----
              IF Input-name = Spaces
                Move "Q" to Input-name
              End-IF

              Move 1 to Char-count
              Inspect Input-name Tallying Char-count For Lea
              Move Input-name(Char-count : 30 - Char-count) t
```

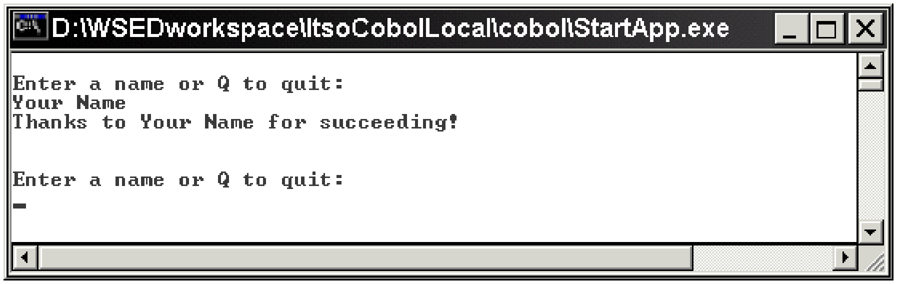
Console [Process: 2656 Program: D:\WSEDworkspace\ItsoCobolLocal\cobol\StartApp.exe]

Figure 14-14 Local Debug At Breakpoint with variables expanded

- Select the `INPUT-NAME` variable and *Change Variable Value* (context). Change the string "`Hekkenberg`" to "`Your Name`" (including quotes).
- Step through the program using the icon. Watch the variables change in the Variables view.
- When you arrive at the line:

  `Call 'PrintApp' using Program-pass-fields`

  use the icon to step into the `PrintApp.cbl` subroutine.
- Step through the subroutine until the output is displayed in the command window of the running application and a new prompt is issued (Figure 14-15).
- Enter `q` to terminate the application. You must click the *Resume* icon to terminate the debug session.



```
Enter a name or Q to quit:
Your Name
Thanks to Your Name for succeeding!


Enter a name or Q to quit:
_
```

Figure 14-15 Application output after changing the name

Obviously this has been a very simple example, but you have seen the debugger at work. We will come back to the debugger when discussing the remote project.

Remote project introduction

From a customer perspective, this is the most likely scenario to use. All the advantages offered from a workstation platform are leveraged while maintaining the actual development location on the host.

Among the advantages are:

- Multithreading (multiple sessions)
- Visual orientation
- Color capabilities
- Mouse orientation
- Faster development cycle (RECD) than using 3270 interface
- Host software configuration management in place
- No need to replicate subsystems in distributed environment
- Integration with other development tooling/perspectives

In this section, we touch upon many of the aspects of working according to the RECD scenario. Given that this chapter is meant to provide a starter introduction, it is not geared solely at the scenario but we also want to provide some background on the system environment and the mechanics of things. Especially the nomenclature of the workstation requires discussion and the mapping onto the nomenclature familiar to an ISPF-based developer. After a discussion of the prerequisites and its configuration, we illustrate the functionality on bringing the local project towards z/OS.

Prerequisites and configuration

In this section, we look at the prerequisites in regard to products, workstation, and z/OS.

Product prerequisites for Enterprise Developer on z/OS

These prerequisites are described in *The Program Directory for WebSphere Studio Enterprise Developer Options for z/OS*, GI10-3242-00. We only mention some aspects of the prerequisites. Please make sure that all products that your environment and your applications require are installed and configured.

IBM Debug Tool for z/OS and OS/390 V3.1

Enterprise Developer relies on this product to provide the mainframe debug engine for performing remote debug. The release mentioned is the lowest that will work with the workstation debug engine that comes with Enterprise Developer. It can be obtained as a stand-alone product or as being part of the full function offering of the IBM Enterprise COBOL for z/OS V3.1 or higher product.

UNICODE support

This support is needed when you want to exploit the XMI support that comes with the V3 level of COBOL. Information on how to obtain the code can be found in the *Program Directory for WebSphere Studio Enterprise Developer Options for z/OS*, GI10-3242-00 on page 13 and 14.

COBOL compiler

Although IBM Cobol for OS/390 and VM V2.x is a supported environment, both the error feedback function and the COBOL XML support will not be available. If you would like to have access to all functionality, you will need IBM Enterprise COBOL for z/OS V3.1 or later.

REXX/370

Make sure that this prerequisite is in place, because Enterprise Developer uses the REXX executable. IBM SAA REXX/370 Alternate Library is actually shipped with Enterprise Developer as a separate install. So if your installation does not happen to have REXX in place, it can be installed from Enterprise Developer media.

Note: If you are going to use the shipped version of REXX, the performance of the REXX will be less than when using the full library. In this case, the REXX will run interpreted.

Workstation prerequisites

You must install and enable the **Microsoft Loopback Adapter**. For instructions on how to do this, see Chapter 4 of the *Enterprise Developer Installation Guide.*

WebSphere Studio Enterprise Developer Options for z/OS

In this section, we briefly discuss the options that are needed from the z/OS development perspective.

z/OS build server

This server provides the capabilities to perform remote builds for either native code or EGL-generated COBOL. It potentially performs the following tasks, depending on what needs to be done:

- Receives build requests and associated files
- Performs character conversions
- Runs builds within the destination environment
- Optionally collects and returns results to the client
- Reports back what has been done

Error feedback modules

These modules are required to provide error feedback exits. In order to exploit these, one needs to point at the appropriate library (`SCCULOAD`). If these routines are available, the user will get feedback from preprocessors and alike that might be used during compile without them having a separate job step.

OS/390 components for RECD

The next sections provide a short description of the components. More information and configuring information can be found in the `INST390.pdf` file, which is titled *Installing and Configuring OS/390 Components for Remote ECD*, SC18-7046-00.

Foreign file system server

The foreign file system (FSS) server provides Enterprise Developer users with transparent access to their MVS data sets. Users can connect their MVS systems to their workstations and explore, edit, copy, delete, and create sequential data sets, partitioned data sets and partitioned data set members. The FFS server functions as an extension of the IBM HTTP Server. The FSS client on the workstation functions as an installable file system.

You can verify that the FFS server has been installed correctly without going through Enterprise Developer by entering the following address from a browser:

```
http://systemname:webserverport/FFSDS/
```

Here `systemname` is the TCP/IP host name or address of the OS/390 system and `webserverport` is the port number for the IBM HTTP Server (standard is 80; on the system we used it was 4080). Upon successful access, you should be able to see the home page of the foreign file system server.

Important: The `FFSDS/` must be uppercase and the final slash must be used. You will be prompted for user ID and password as well.

Job monitor server

The job monitor server provides Enterprise Developer users with function similar to SDSF: the ability to view job status, view job output and purge, cancel, and release jobs and job output. It is installed together with FFS.

TSO remote command server

This optional feature allows users to issue TSO commands from the workstation and receive their feedback.

Figure 14-16 provides an overview of the software required to have all functions in working order. It does not show the components of Enterprise Developer.

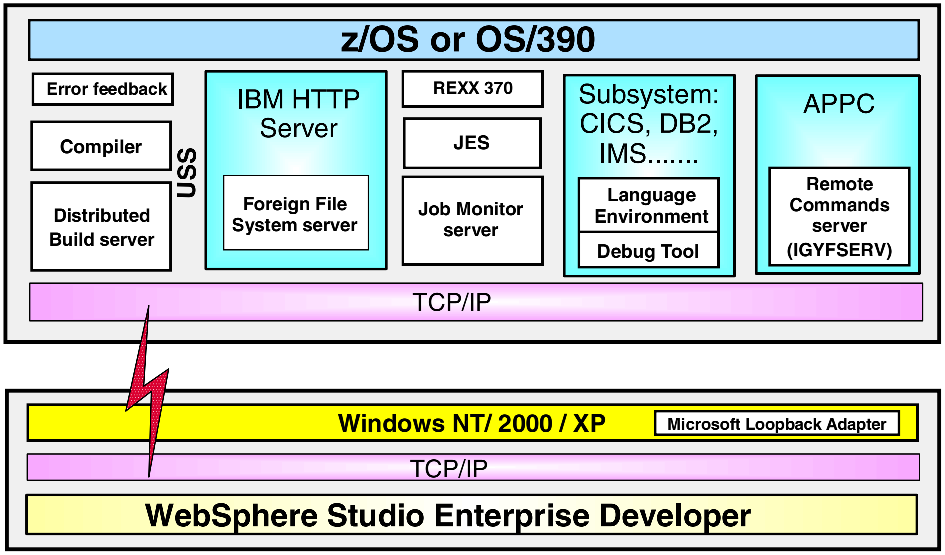


Figure 14-16 Overview of software required

z/OS sample data sets

In order for our sample to be moved to z/OS, we have to create several partitioned data sets. The reader could certainly use existing data sets, but it is probably a good idea to create a separate set of data sets to get familiar with the RECD approach of Enterprise Developer, thus separating the sample from production.

Please create the data sets using ISPF according to the specifications that can be found in Table 14-1, where userID is substituted with your user ID.

Table 14-1 z/OS data sets for a user

PDS name	Record format	Record length	Block size
userID.ENTDEV.COBOL	FB	80	6160
userID.ENTDEV.COPYLIB	FB	80	6160
userID.ENTDEV.JCL	FB	80	6160
userID.ENTDEV.LISTING	FBA	133	2660
userID.ENTDEV.OBJECT	FB	80	3200
userID.ENTDEV.LOAD	U	0	6233

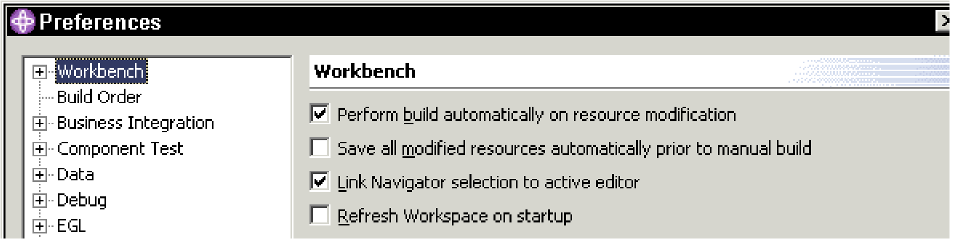
Setting Enterprise Developer preferences

Preferences can be set on various levels within the Enterprise Developer product. At this point, we describe only a few preferences that have to be set on the Workbench level and are relevant for our project. When we define our remote project, we will define overrides to the Workbench-level preferences specific to a remote project.

Tip: Default preferences can be set by an administrator and imported by the various users. Users can also export their settings in order to share them or for backup purposes. For that purpose, the *Preferences* dialog has an *Import* and *Export* button. The suffix of the file is always `.epf`.

Restriction: These preferences are not moved when a project is moved to a software configuration and version management tool. A procedure should be in place to export and position the `.epf` file within project as a file.

- The first preference option we want to change is shown after selecting *Window -> Preferences -> Workbench* (Figure 14-17).



Workbench

☑ Perform build automatically on resource modification

□ Save all modified resources automatically prior to manual build

☑ Link Navigator selection to active editor

□ Refresh Workspace on startup

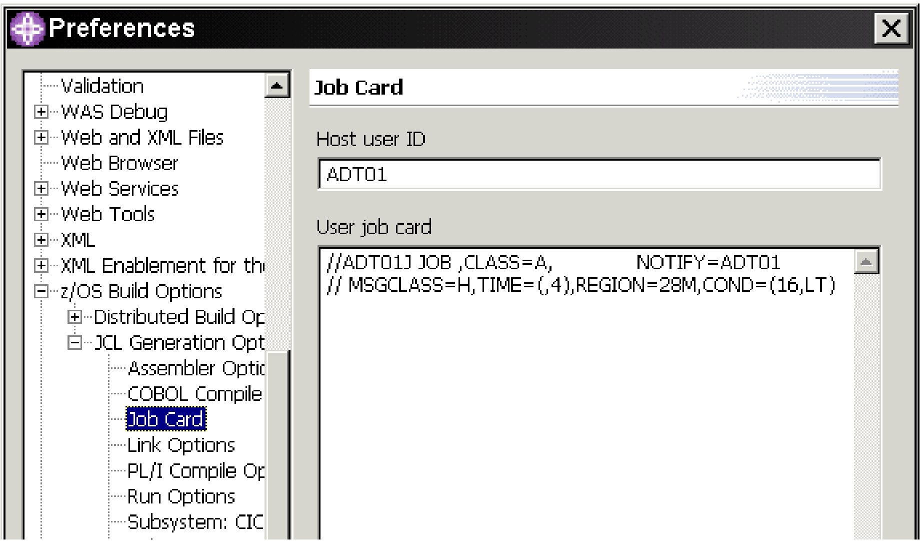
Figure 14-17 Workbench preferences

- We advise the developers working on remote projects to deselect *Perform build automatically on resource modification*. This way rebuilds will not start automatically when a change is applied to an artifact.

> **Tip:** In the text behind the check boxes, one character has an underscore. When you press the `Alt` key and this character on your keyboard, the check mark will switch.

- The next preference option is under *z/OS Build Options -> JCL Generation Options -> Job Card.* After the changes, the job card should resemble Figure 14-18 where `ADT01` is your user ID.

  This job card will be used by Enterprise Developer when it generates JCL either on request or as part of its operations.



Job Card

Host user ID: ADT01

User job card

```
//ADT01J JOB ,CLASS=A,         NOTIFY=ADT01
// MSGCLASS=H,TIME=(,4),REGION=28M,COND=(16,LT)
```

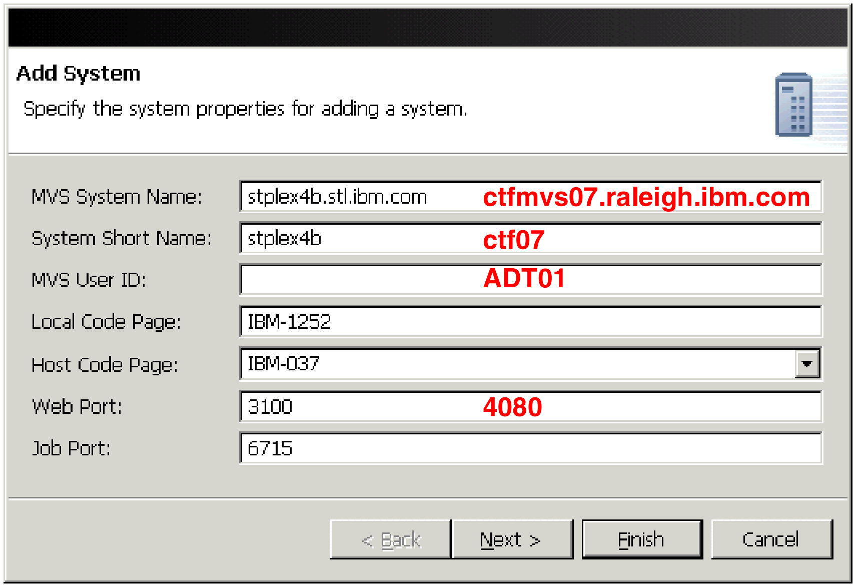
Figure 14-18 Job card preferences

Define and connect to a remote system

We have to define one or more remote systems for which we are going to use data sets in our remote projects. It is more convenient to define the system(s) prior to the definition of the remote project(s), because this will enable easy referral to the system(s) when defining the project(s).

To define a remote system:

- Open (or switch to) a z/OS Systems perspective.
- Select *Remote Systems* in the z/OS Systems view and *Add System* (context). The Add System dialog (Figure 14-19) is displayed.



Add System

Specify the system properties for adding a system.

MVS System Name:	stplex4b.stl.ibm.com	ctfmvs07.raleigh.ibm.com
System Short Name:	stplex4b	ctf07
MVS User ID:		ADT01
Local Code Page:	IBM-1252	
Host Code Page:	IBM-037	
Web Port:	3100	4080
Job Port:	6715	

Figure 14-19 Defining a z/OS system for Enterprise Developer

- MVS System Name—should reflect the fully qualified TCP/IP name of the z/OS system. From a command window, you should be able to ping the system (`ping fullqualifiedname`) and should receive an answer.
- System Short Name—is yours to choose; it will be used when Enterprise Developer lists the system.
- MVS User ID—should be your user ID.
- Local Code Page—represent the active code page of your workstation.
- Host Code Page—This should be provided to you by the z/OS system support people.

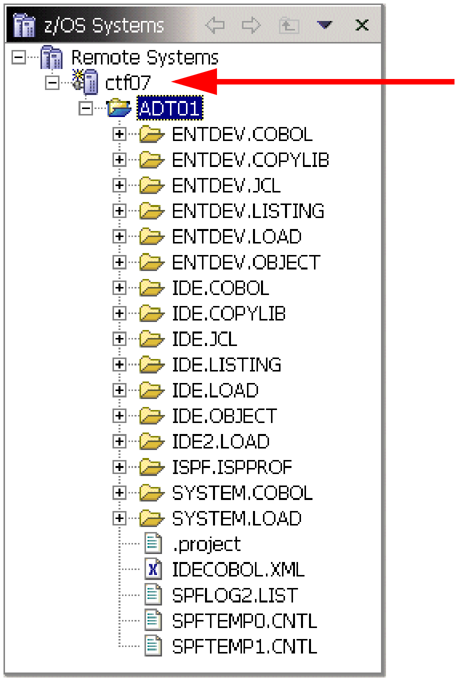
- Web Port—should be the same as discussed in “Foreign file system server” on page 366.

  > **Tip:** We suggest that your installation provides a separate HTTP server for the exclusive use of Enterprise Developer users. This way the server can be started and stopped without interfering with other HTTP applications.

- Job Port—obtain the number from your systems support people; most installations tend to keep the default (6715).

- Click *Finish*. You are prompted to connect to the system. Click *Yes* and *OK* in the pop-up dialog asking you if it is OK to close all existing connections before attempting to connect to the newly defined system.
- Fill in your password and click *OK.*

Now your remote system is defined! Figure 14-20 shows the expanded z/OS Systems view. As you can see, the data sets under your user ID are listed.



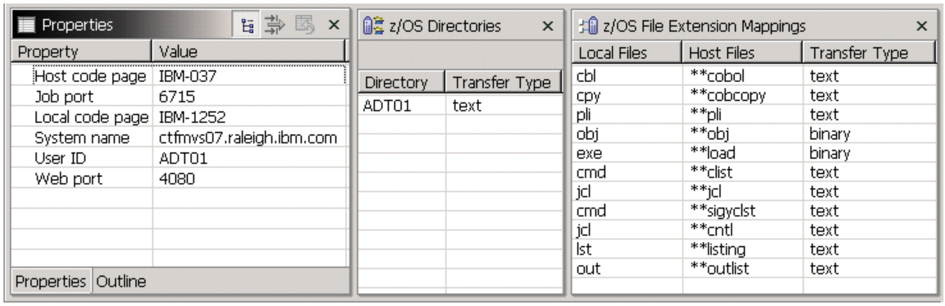
```
z/OS Systems
Remote Systems
  ctf07
    ADT01
      ENTDEV.COBOL
      ENTDEV.COPYLIB
      ENTDEV.JCL
      ENTDEV.LISTING
      ENTDEV.LOAD
      ENTDEV.OBJECT
      IDE.COBOL
      IDE.COPYLIB
      IDE.JCL
      IDE.LISTING
      IDE.LOAD
      IDE.OBJECT
      IDE2.LOAD
      ISPF.ISPPROF
      SYSTEM.COBOL
      SYSTEM.LOAD
      .project
      IDECOBOL.XML
      SPFLOG2.LIST
      SPFTEMP0.CNTL
      SPFTEMP1.CNTL
```

You can disconnect from the system.

On reconnect you are prompted for the password.

Figure 14-20 Remote system in connected state

Select the system you just defined and you should see the Properties, z/OS Directories, and z/OS File Extension Mappings views at the bottom of the Enterprise Developer window (Figure 14-21).



Property	Value
Host code page	IBM-037
Job port	6715
Local code page	IBM-1252
System name	ctfmvs07.raleigh.ibm.com
User ID	ADT01
Web port	4080

Directory	Transfer Type
ADT01	text

Local Files	Host Files	Transfer Type
cbl	**cobol	text
cpy	**cobcopy	text
pli	**pli	text
obj	**obj	binary
exe	**load	binary
cmd	**clist	text
jcl	**jcl	text
cmd	**sigyclst	text
jcl	**cntl	text
lst	**listing	text
out	**outlist	text

Figure 14-21 z/OS systems perspective for a connected system

- The **Properties** view speaks for itself.
- The **z/OS Directories** view shows your user ID as the high-level qualifier.

  Enterprise Developer is using the high-level qualifier to see what data sets and sequential files are available. These data sets can then be mapped onto the project to indicate that this project can access the associated artifacts. Directory can be regarded as being the workstation’s analogy for the high-level qualifier.

  There are dialogs available to create additional directories so that projects can map to a variety of data sets as needed. These dialogs are available from the context menu of the defined system.
- The **z/OS File Extensions Mappings** view displays how host files in partitioned data set members are mapped onto workstation files.

  For example, the members of our data set `ADT01.ENTDEV.COBOL` (**COBOL in this pane) will have workstation names `*.cbl` and are transferred in text format. Load modules are mapped to `.exe` and are transferred in binary.

  Thus far we have not identified the analogy for partitioned data sets as a means to group multiple members of the same type. The analogy here is folder. Workstation files with the suffix `.cbl` will be grouped into folders that can be seen in the dialogs of Enterprise Developer.

Important: Although the directories, folders, and files can be seen on the various dialogs, this does not mean that these artifacts are physically present in the file system. The workstation representation merely acts as a view on the remote artifacts. Likewise, ISPF dialogs provide a view into the physical artifacts within the host system.

Creating and configuring a remote project

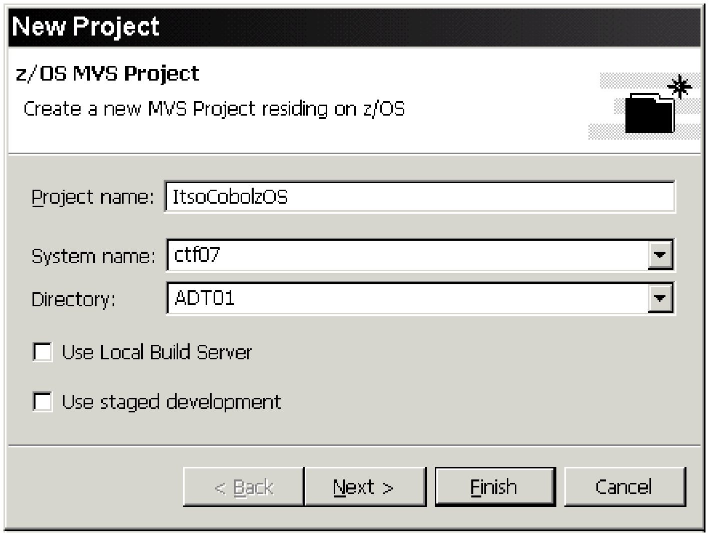
In this section we define a remote project to map to the remote MVS system. Then we set up COBOL files in the remote data sets.

Creating the project

A project can be regarded as being a view upon a collection of artifacts that represent a development effort. From a z/OS perspective, this more often is a collection of artifacts representing a program, transaction, or system. It should be possible to operate upon a project as being an entity. Operations, such as build for instance, should be able to run on the project level.

Let us define a remote project to work with our z/OS COBOL artifacts:

- Open the z/OS Projects perspective (you can close the z/OS Systems perspective because the z/OS Systems view is also part of the z/OS Projects perspective).
- Select File -> *New* -> z/OS -> *MVS Project* (or *New* -> *MVS Project* from the context of the z/OS Project view).
- Enter `ItsoCobolzOS` as project name and select `ctf07` and `ADT01` from the pull-downs for system name and directory (Figure 14-22). Obviously you have to select your system's short name and high-level qualifier, but there is little choice if this is your first encounter with Enterprise Developer.


New Project

z/OS MVS Project

Create a new MVS Project residing on z/OS

Project name: ItsoCobolzOS

System name: ctf07

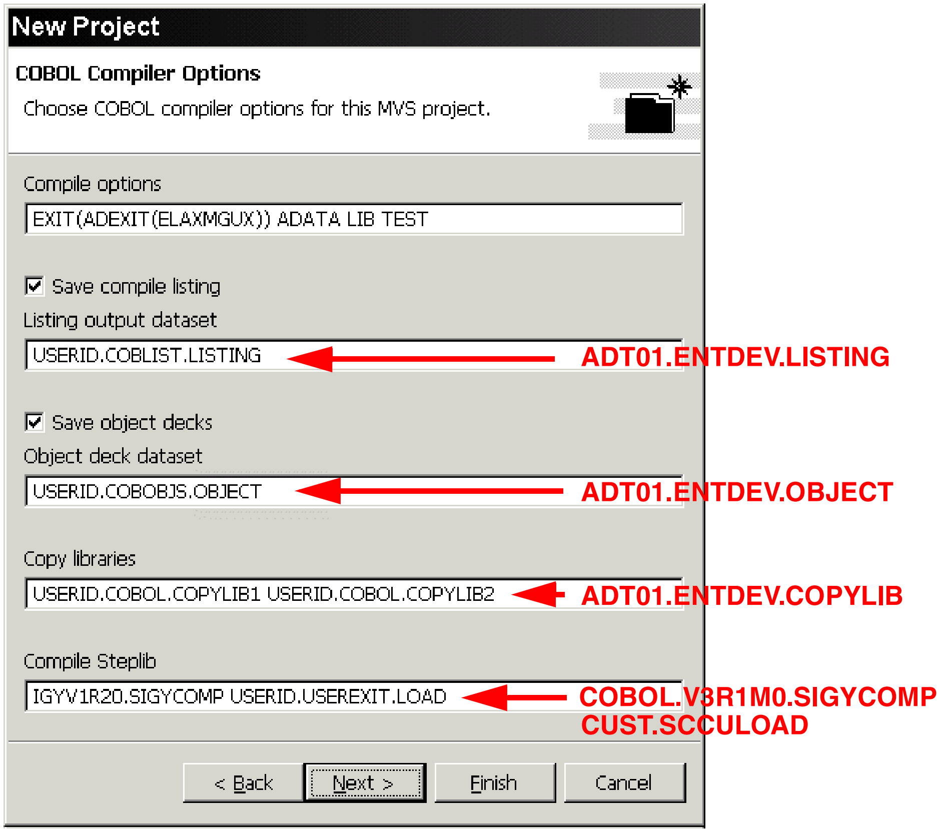
Directory: ADT01

□ Use Local Build Server

□ Use staged development

Figure 14-22 New remote project dialog

- After three clicks on *Next* (we can skip those panels), the CICS Subsystem Options dialog is displayed. Given that our project is not using CICS, we can skip this dialog as well. The same is true for the next two dialogs for DB2 and IMS.
- After these three clicks on *Next*, the COBOL Compiler Options dialog is displayed. We modify the options to match our MVS system (Figure 14-23).


New Project

COBOL Compiler Options

Choose COBOL compiler options for this MVS project.

Compile options: EXIT(ADEXIT(ELAXMGUX)) ADATA LIB TEST

☑ Save compile listing

Listing output dataset: USERID.COBLIST.LISTING

☑ Save object decks

Object deck dataset: USERID.COBOBJS.OBJECT

Copy libraries: USERID.COBOL.COPYLIB1 USERID.COBOL.COPYLIB2

Compile Steplib: IGYV1R20.SIGYCOMP USERID.USEREXIT.LOAD

ADT01.ENTDEV.LISTING

ADT01.ENTDEV.OBJECT

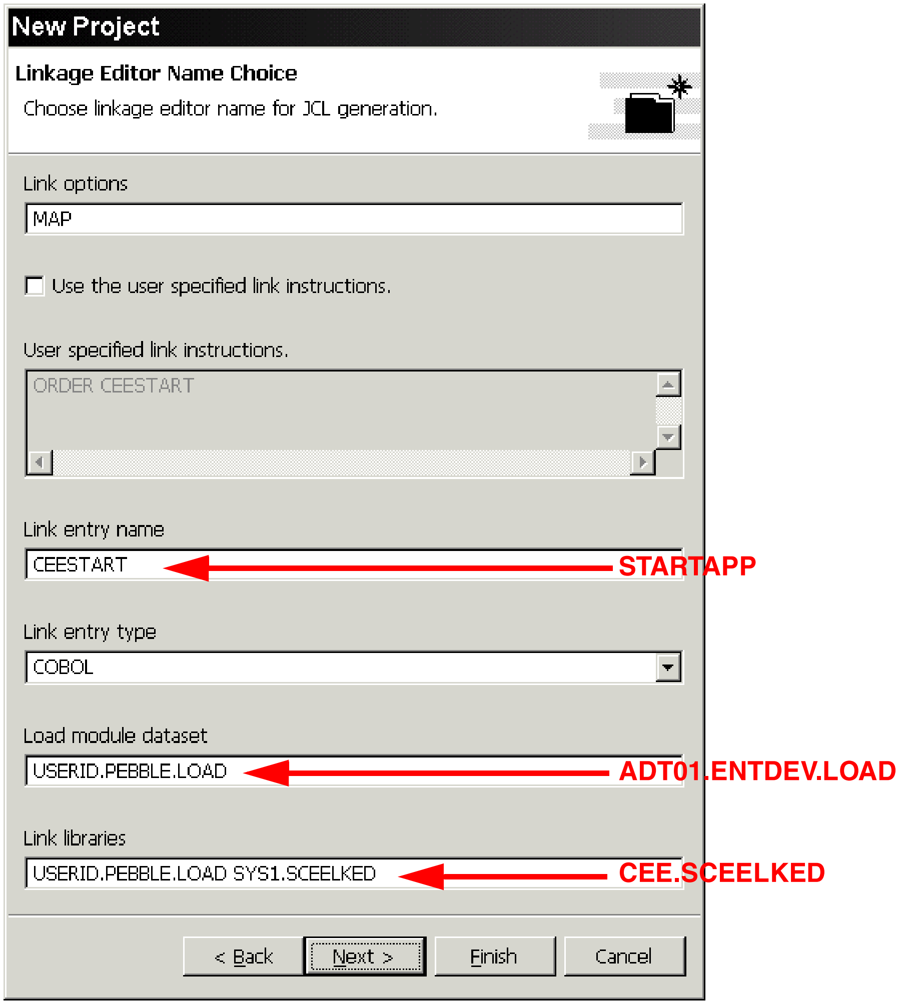
ADT01.ENTDEV.COPYLIB

COBOL.V3R1M0.SIGYCOMP
CUST.SCCULOAD

Figure 14-23 New project COBOL compiler options

In this dialog you can indicate whether or not compile listings and object decks have to be saved. Also note that for the Compile Steplib the error routine load library (`SCCULOAD`) is added.

- Click *Next* three times (we skip the PLI and Assembler panels) and we come to the Linkage Editor Name Choice dialog. We modify the options to match our MVS system (Figure 14-24).


New Project

Linkage Editor Name Choice

Choose linkage editor name for JCL generation.

Link options: MAP

☐ Use the user specified link instructions.

User specified link instructions.

```
ORDER CEESTART
```

Link entry name: CEESTART ← STARTAPP

Link entry type: COBOL

Load module dataset: USERID.PEBBLE.LOAD ← ADT01.ENTDEV.LOAD

Link libraries: USERID.PEBBLE.LOAD SYS1.SCEELKED ← CEE.SCEELKED

Figure 14-24   New project linkage editor options

Note that we specify `STARTAPP` as being the entry point for this project's program load module.

- Click *Finish.* We skip the Run Options page. We will adapt these options in "Debugging the remote executable" on page 389.

The project appears in the z/OS Projects view with a composed name of the project, system, and high-level qualifier (`ItsoCobolzOS-ctf07.ADT01`).

Project properties

All the preferences we set in the New wizard can be viewed and modified by selecting the project and *Properties* (context). These values are project overrides to the Workbench preferences that we set up in “Setting Enterprise Developer preferences” on page 368 (see Figure 14-17).

Tips:

- Properties can be set at the project (high-level qualifier), folder (PDS), and file (PDS member) level. In a production development environment, it is likely that you would prefer to define the link options at the file level.
- You can save considerable time when configuring projects if the Workbench administrator defines appropriate defaults for the preferences at the Workbench level.

Map data sets for the remote project

Now we have to map our data sets to the project by adding appropriate folders, which represent the partitioned data sets, to the project:

- In the z/OS Systems view, select the data sets for our COBOL project (use the `Ctrl` key for multiple selections). Select *Add To Project* (context), in the dialog select the `ItsoCobolzOS` project, and click *Finish* (Figure 14-25). Notice that the folders actually disappear from the z/OS Systems view and are added to the project in the z/OS Projects view.

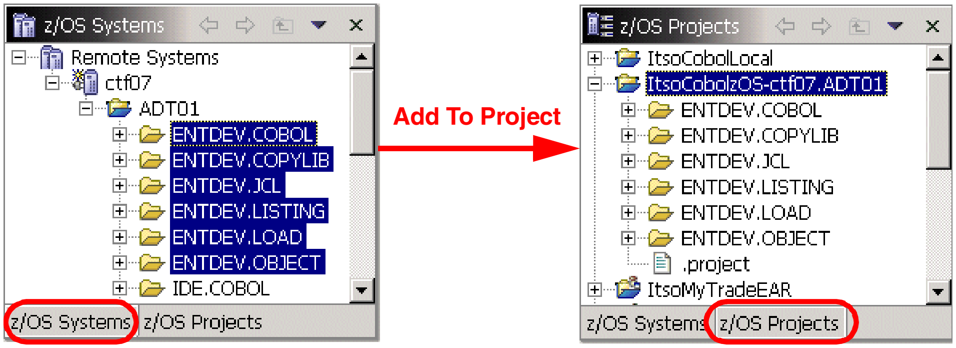


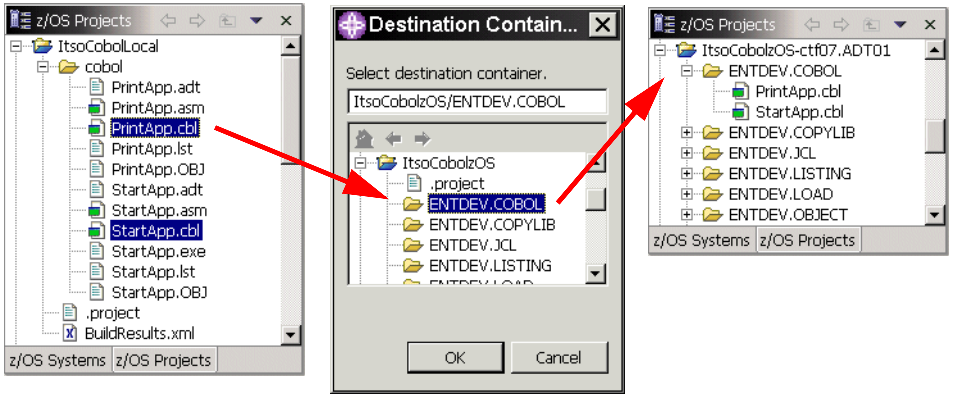
Figure 14-25 Add folders to the remote project

Important: If the connection to the remote system is closed or you stop Enterprise Developer, the remote projects are closed as well. They can only be opened (by hand or automatically depending on the situation) after you have restored the connection.

Copying files from local to remote project

Instead of writing new COBOL programs, we use our existing local project and copy the COBOL programs to the remote project:

- Expand the `cobol` folder in the `ItsoLocalCobol` project. Select `PrintApp.cbl` and `StartApp.cbl` and *Copy* (context). In the destination container dialog, select the `ENTDEV.COBOL` folder in the remote project and click *OK*. The two programs are copied to the z/OS system and they appear in the remote project (Figure 14-26).



Select destination container.

ItsoCobolzOS/ENTDEV.COBOL

Figure 14-26 Copy files from a local to a remote project

- Note that the cobol programs are actually moved to z/OS during this operation. If you don't believe us, log on to the MVS system and have a look!
- The `StartApp.cbl` flag still carries the MAIN notation from the local project. You can select a program and *Set As Main* (context).

Operations on members

You can select a member in the z/OS folder (`ENTDEV.COBOL`) and perform operations such as:

- Delete the member.
- Rename the member.
- Syntax check.
- Open with an editor. The default editor is JLPEX, a Java LPEX editor that is integrated into the Workbench.

Distributed build architecture

Figure 14-27 is a representation of the way the distributed build operates when used for building remote z/OS projects.

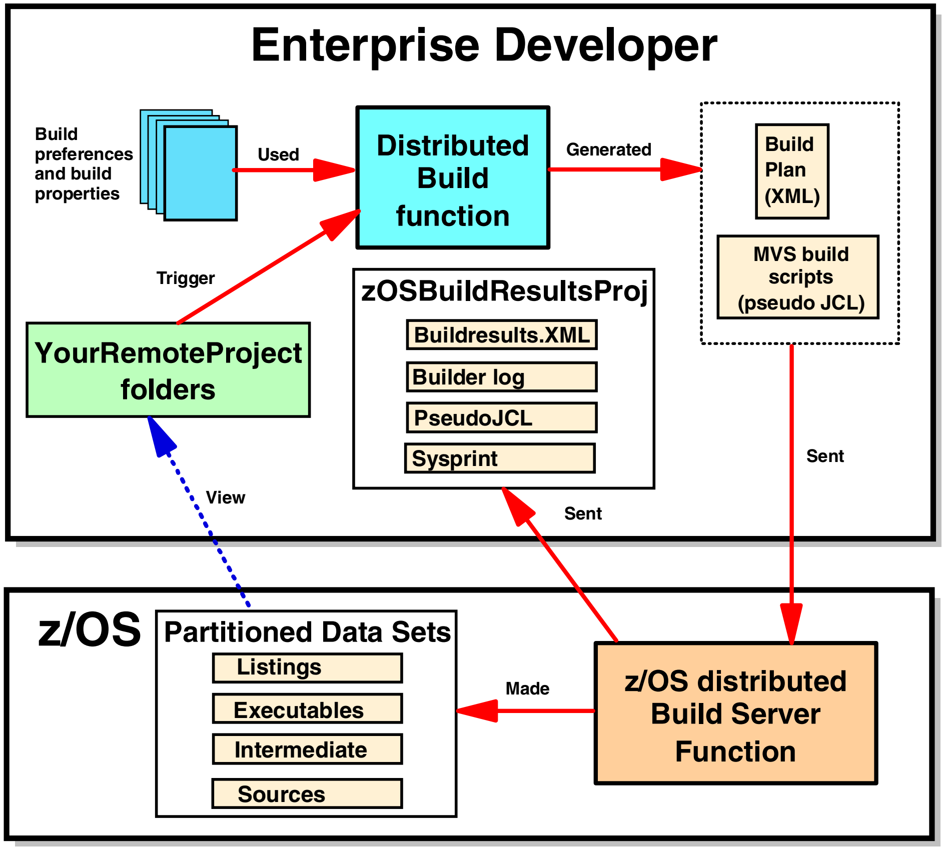


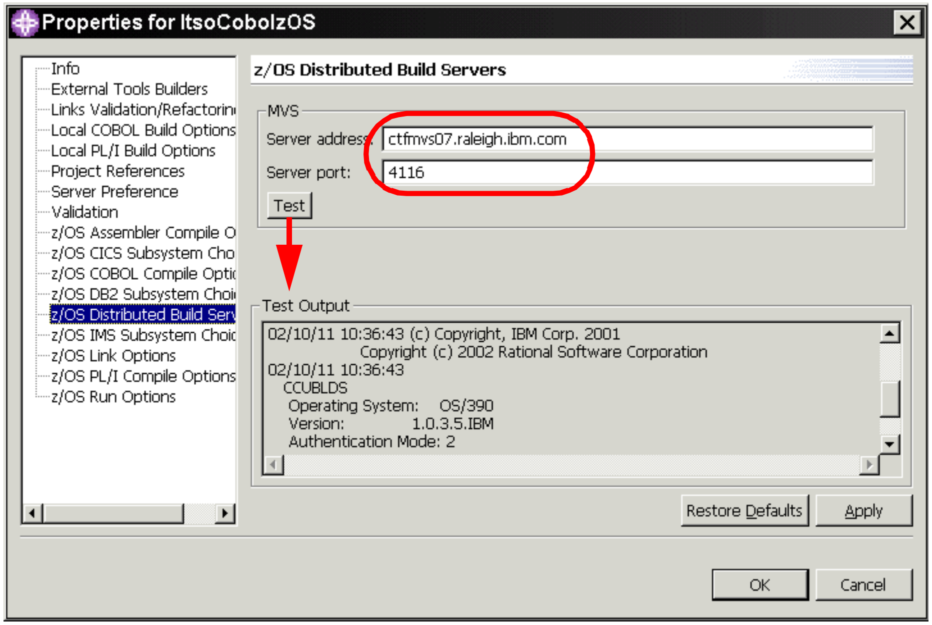
Figure 14-27 Distributed build for z/OS development

Define a distributed build server

The distributed build server that will handle build requests must be defined and up and running for the build clients to use it:

- Select the `ItsoCobolzOS` project in the z/OS Projects view and *Properties -> z/OS Distributed Build Servers*.
- Your development support staff should provide you the information specific to your installation, being the port number for the build server and the fully

qualified name of the host system. Enter the server address and port, then click *Test* to test the connection (Figure 14-28).



Server address: ctfmvs07.raleigh.ibm.com

Server port: 4116

```
02/10/11 10:36:43 (c) Copyright, IBM Corp. 2001
                Copyright (c) 2002 Rational Software Corporation
02/10/11 10:36:43
  CCUBLDS
   Operating System:    OS/390
   Version:             1.0.3.5.IBM
   Authentication Mode: 2
```

Figure 14-28 Defining and testing a distributed build server

- Click *OK* and you will be able to exploit the capabilities of the distributed build server.

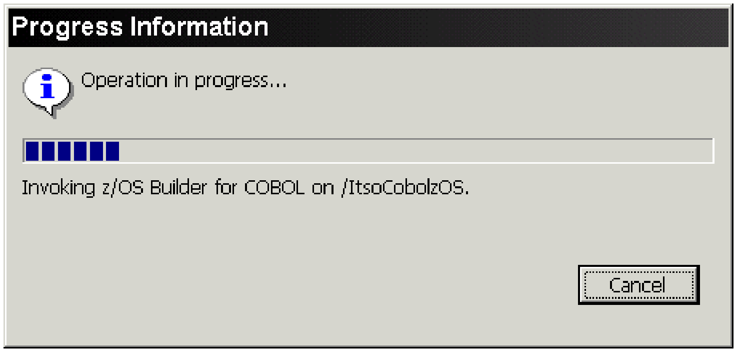
Note: In the test output pane, you can see that the build server is configured to run with Authentication Mode: 2. This actually can differ from what you are seeing, depending on your installation preference. More information on the possibilities here can be found in the help.

Building the remote project

Building the project with all the setup done is quite simple.

Starting a distributed build

Select the `ItsoCobolzOS` project in the z/OS Projects view and *Rebuild Project* (context). Soon thereafter, the progress indicator of the distributed build is displayed (Figure 14-29).



Operation in progress...

Invoking z/OS Builder for COBOL on /ItsoCobolzOS.

Figure 14-29 Progress indicator for distributed build

Important: While performing the build, a command window with the title CCU Security Manager might be started on your machine (Figure 14-30). You have to close this window; otherwise the build will never finish. Note that this can occur multiple times and that the window is sometimes hidden or minimized. We expect that this will be fixed.

Figure 14-30 CCU Security Manager

Distributed build results

After the build is finished, a new project called `zOSBuildResultsProj` appears in the z/OS Projects view.

Important: In some circumstances you may get an additional project named `zOSOutputProj`. This seems to happen if JCL errors are encountered during the build. Check the build results, fix the problem, delete the generated project, and rerun the build.

Select the new project and the `ItsoCobolzOS` project and *Refresh* (context) to see the new files that were created.

Figure 14-31 shows the z/OS Projects view with the projects expanded.

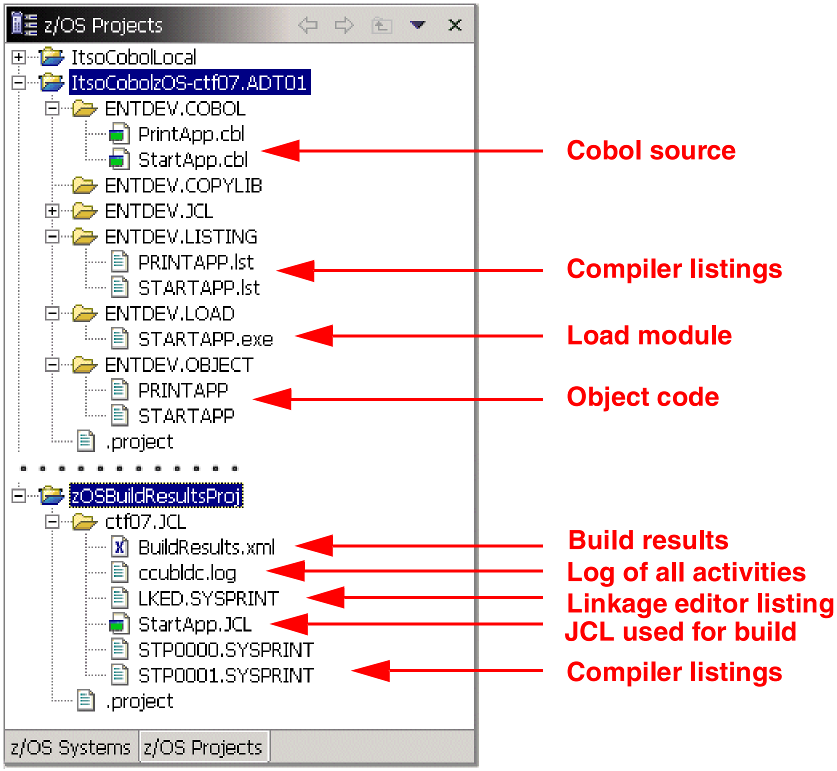


Figure 14-31 z/OS projects after distributed build

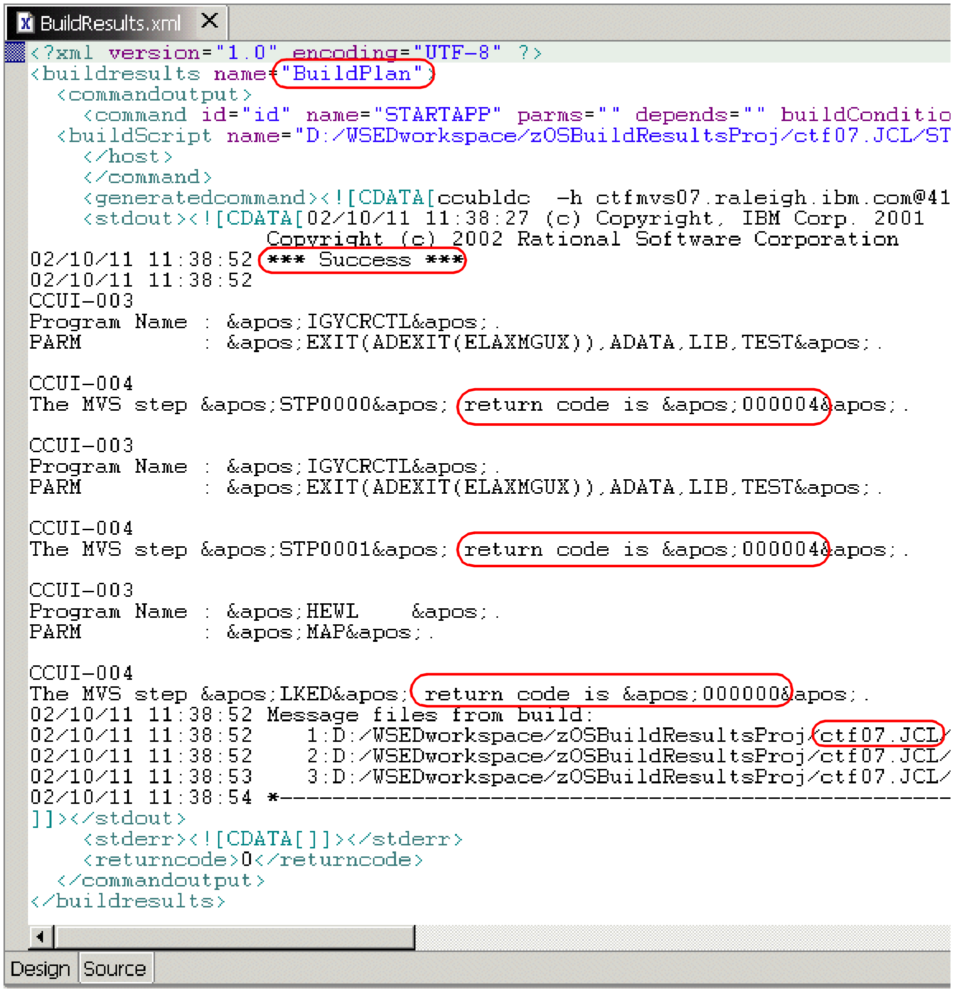
Now let us discuss some of the files in more detail.

BuildResults.xml

This file is always created on your workstation when a distributed build has run. Normally it is enough to look at this file to know what happened during the build (Figure 14-32).

The red ovals are positioned in the editor display at various places of interest:

- `BuildPlan`: the top part of this file actually reflects the build plan.
- Success: the build plan was executed successfully within the error limits set.
- Return code 4 (twice): these are the return codes of both compiles.
- Return code 0: this is the return code of the linkage editor.
- `ctf07.JCL` is the folder where the full output of the translators is held (three `*.sysprint` files); which are in effect local copies of the members in the listing data set (link edit listing is not kept on the host system).



```
<?xml version="1.0" encoding="UTF-8" ?>
<buildresults name="BuildPlan">
  <commandoutput>
    <command id="id" name="STARTAPP" parms="" depends="" buildConditio
  <buildScript name="D:/WSEDworkspace/zOSBuildResultsProj/ctf07.JCL/ST
    </host>
    </command>
    <generatedcommand><![CDATA[ccubldc  -h ctfmvs07.raleigh.ibm.com@41
    <stdout><![CDATA[02/10/11 11:38:27 (c) Copyright, IBM Corp. 2001
                  Copyright (c) 2002 Rational Software Corporation
02/10/11 11:38:52 *** Success ***
02/10/11 11:38:52
CCUI-003
Program Name : &apos;IGYCRCTL&apos;.
PARM         : &apos;EXIT(ADEXIT(ELAXMGUX)),ADATA,LIB,TEST&apos;.

CCUI-004
The MVS step &apos;STP0000&apos; return code is &apos;000004&apos;.

CCUI-003
Program Name : &apos;IGYCRCTL&apos;.
PARM         : &apos;EXIT(ADEXIT(ELAXMGUX)),ADATA,LIB,TEST&apos;.

CCUI-004
The MVS step &apos;STP0001&apos; return code is &apos;000004&apos;.

CCUI-003
Program Name : &apos;HEWL    &apos;.
PARM         : &apos;MAP&apos;.

CCUI-004
The MVS step &apos;LKED&apos; return code is &apos;000000&apos;.
02/10/11 11:38:52 Message files from build:
02/10/11 11:38:52    1:D:/WSEDworkspace/zOSBuildResultsProj/ctf07.JCL/
02/10/11 11:38:52    2:D:/WSEDworkspace/zOSBuildResultsProj/ctf07.JCL/
02/10/11 11:38:53    3:D:/WSEDworkspace/zOSBuildResultsProj/ctf07.JCL/
02/10/11 11:38:54 *-------------------------------------------------
]]></stdout>
    <stderr><![CDATA[]]></stderr>
    <returncode>0</returncode>
  </commandoutput>
</buildresults>
```

Figure 14-32 BuildResults.xml after distributed build

ccubldc.log

The `ccubldc.log` file is a continuous log of all the builds. It contains parts of the `BuildResults.XML` file.

ENTDEV folders

In the respective folders of the `ItsoCobolzOS` project, you can find the load module, the object decks, and the translator listings.

StartApp.JCL

StartApp.JCL is the actual generated pseudo JCL that was used by the build server. The job card is not included in the file. Note that this file is referred to as *buildscript* in the BuildResults.xml file.

> **Restriction:** If you issue another rebuild project (for instance to build with changed parameters), a new JCL deck is generated named STARTAP.JCL. However, this new deck is not picked up by the build plan, which still points to StartApp.JCL.
>
> Currently the new STARTAP.JCL has an error in the member name of the SYSLMOD statement.
>
> Therefore, to use a new JCL deck, fix the error, then rename the new deck to StartApp.JCL.

You can modify the generated JCL. For example, the generated JCL has this statement for the linkage editor listing:

```
//SYSPRINT DD CCUEXT=CCUOUT
```

- Changing the JCL as shown here will save the output in the LISTING data set:

```
//SYSPRINT DD CCUEXT=CCUOUT,DISP=SHR,DSN=ADT01.ENTDEV.LISTING(LKED)
```

- Changing the JCL as shown here will not send the listing to the workstation:

```
//SYSPRINT DD DISP=SHR,DSN=ADT01.ENTDEV.LISTING(LKED)
```

Incremental build

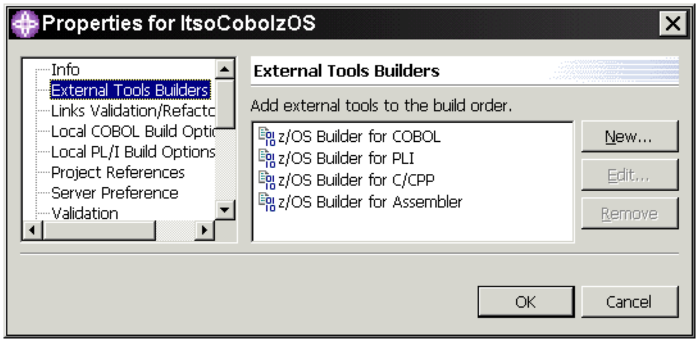
The *Build Project* action of a project performs an incremental build and will only build the artifacts that have changed. The difference in our project is not a lot (saving one compile) but can be substantial for larger projects.

Speeding up the remote build process

You may have noticed that when you build the project, the progress window shows COBOL first, then PLI, then CPP, then ASM, so actually four build processors are running.

The build processors used in a project can be seen in the project properties on the External Tool Builders page (Figure 14-33).

This dialog does not allow you to remove any of the builders.



Properties for ItsoCobolzOS

- Info
- External Tools Builders
- Links Validation/Refacto
- Local COBOL Build Optic
- Local PL/I Build Options
- Project References
- Server Preference
- Validation

External Tools Builders

Add external tools to the build order.

- z/OS Builder for COBOL
- z/OS Builder for PLI
- z/OS Builder for C/CPP
- z/OS Builder for Assembler

Figure 14-33 External tool builders for the remote project

We know that our project only includes COBOL. To remove the unwanted builders, edit the `.project` file in the `ItsoCobolzOS` project.

Remove the PLI, CPP, and ASM tool builders as shown in Figure 14-34.

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>ItsoCobolzOS</name>
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
            <name>com.ibm.ftt.ui.views.navigator.pbCobolCompileAndLink</name>
            <arguments></arguments>
        </buildCommand>
        ~~<buildCommand>~~
            ~~<name>com.ibm.ftt.ui.views.navigator.pbbuilderpli</name>~~
            ~~<arguments></arguments>~~
        ~~</buildCommand>~~
        ~~<buildCommand>~~
            ~~<name>com.ibm.ftt.ui.views.navigator.pbbuildercpp</name>~~
            ~~<arguments></arguments>~~
        ~~</buildCommand>~~
        ~~<buildCommand>~~
            ~~<name>com.ibm.ftt.ui.views.navigator.pbbuilderasm</name>~~
            ~~<arguments></arguments>~~
        ~~</buildCommand>~~
    </buildSpec>
    <natures>
        ....
    </natures>
</projectDescription>
```

Figure 14-34 Removing unwanted build operations

JCL generation

Apart from the facilities of the distributed build server, Enterprise Developer is offering other means to build programs.

Building a single file

If you select `PrintApp.cbl` and *Generate JCL*, three options are provided (Figure 14-35). The first option generates JCL to compile the program, the second option generates JCL to compile and link, and the third option generates JCL to compile, link and run.

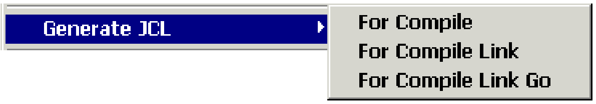


Figure 14-35 Generate JCL options menu

Notes:

- The user should have application knowledge to know which option is meaningful. In our case only the first option is meaningful because this is a called module that cannot run as a stand-alone application.
- (Build) properties can also be set at the file (member) level, thus allowing you to perform compile and link operations with other options if needed for test/debug purposes.

Let us generate JCL for *Compile* for `PrintApp.cbl` at this point.

Another project named `zOSGeneratedJCLProj` is generated. After the *Refresh* action on the project, you can see the JCL file created. Actually this JCL is a local file in the file system. This is probably OK for test and trial JCL. If you want it to be present on the host, then you have to copy or move it into a remote project.

You can look at the generated JCL by opening an editor.

Building a partial project

JCL can also be generated on the folder level and as such you can generate JCL for building a partial project. In our case, we do not have more folders and modules in our project, but this has potential use in larger projects. The same three options shown in Figure 14-35 are offered here.

Tips:

- Properties can be set at the folder level, thus completing the full range of places where one can set properties: Workbench preferences, project, folder and file level properties.
- Options have to be carefully set on file and folder levels to generate JCL that will produce the wanted results.
- Only one JCL member with the same name can exist, so only the last request is reflected in the generated JCL.

Let us generate JCL for the `ENTDEV.COBOL` folder for *Compile Link Go*. After refresh, you can see the newly created `ENTDEV.JCL` member. From the name, you can see that this is JCL for a partial project build. All files within the folder are built when this JCL is submitted. Furthermore, a run step is created to run the application.

Note: In the JCL, the newly created load module is named `ENTDEV`, although from a functional perspective it is the `STARTAPP` application.

Job and command interactions with z/OS

In this section we look at interactions with the z/OS system, such as submitting jobs, monitoring jobs, retrieving job output, and issuing TSO commands.

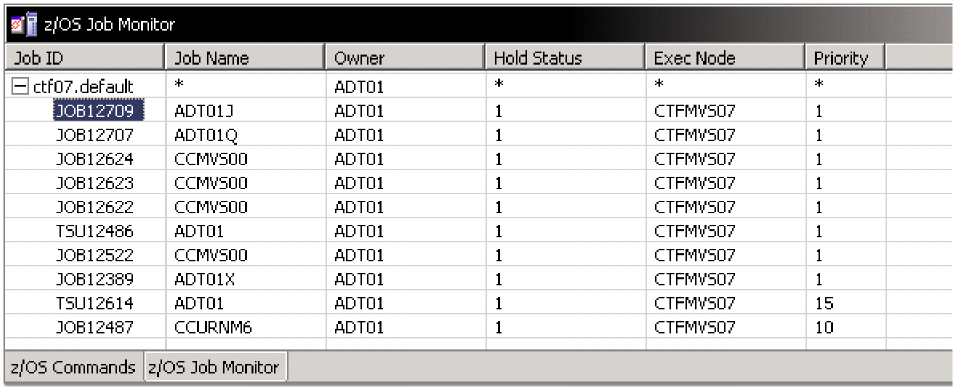
Submitting and monitoring jobs

After the creation of JCL, we would like to be able to submit the JCL for execution on the host. Select the `ENTDEV.JCL` and *Submit* (context).

Restriction: This currently does not work. You can move the job to MVS yourself and submit it from a TSO user ID.

To see the jobs that were executed, open the z/OS Job Monitor view (select *Window -> Show View -> Other -> z/OS Projects Views -> z/OS Job Monitor*).

Figure 14-36 shows the z/OS Job Monitor view with the jobs owned by user ID `ADT01`. This view is not automatically refreshed, so click the *Refresh* icon to see the latest status.



Job ID	Job Name	Owner	Hold Status	Exec Node	Priority
ctf07.default	*	ADT01	*	*	*
JOB12709	ADT01J	ADT01	1	CTFMVS07	1
JOB12707	ADT01Q	ADT01	1	CTFMVS07	1
JOB12624	CCMVS00	ADT01	1	CTFMVS07	1
JOB12623	CCMVS00	ADT01	1	CTFMVS07	1
JOB12622	CCMVS00	ADT01	1	CTFMVS07	1
TSU12486	ADT01	ADT01	1	CTFMVS07	1
JOB12522	CCMVS00	ADT01	1	CTFMVS07	1
JOB12389	ADT01X	ADT01	1	CTFMVS07	1
TSU12614	ADT01	ADT01	1	CTFMVS07	15
JOB12487	CCURNM6	ADT01	1	CTFMVS07	10

Figure 14-36 Job monitor view

You can select a job and *Output* or *Purge* (context). If you select *Purge*, it is obvious what will happen. If you select *Output*, the job output is downloaded into a new project named `zOSOutputProj` and an editor is opened on the output file.

For the compile, link, go job the output displays all the steps, and at the bottom the actual output of the execution:

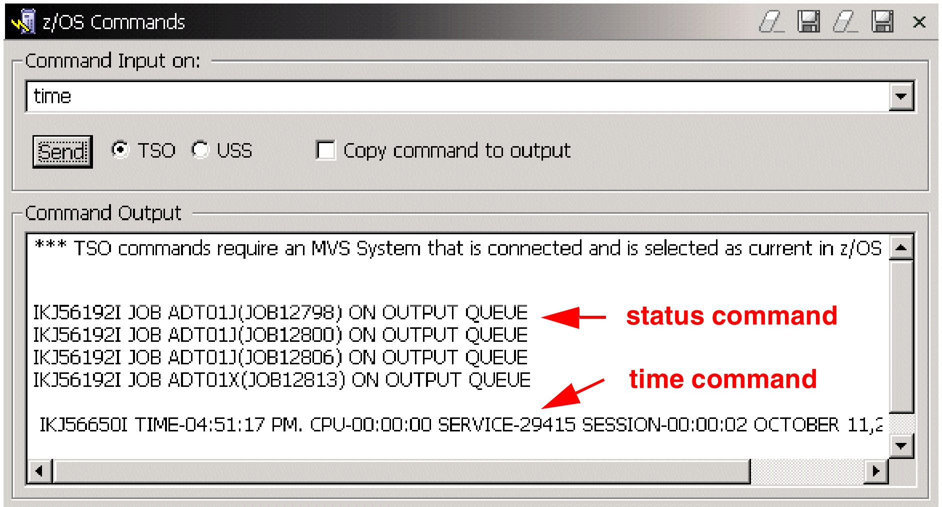
```
Enter a name or Q to quit:
 Hello   HEKKENBERG You did it !

  Enter a name or Q to quit:
```

Tip: The job monitor can use a filter to limit the number of jobs shown. In our case, the filter is set to show all jobs that we own. The job filter can be defined for the system in the z/OS Systems view, but only if you are not connected.

Issuing z/OS commands

One of the views available to the user that we did not discuss thus far is the z/OS Commands view, which is part of the z/OS Projects perspective. From this view, you can issue z/OS commands and see the command output (Figure 14-37).

Notice that you can save the command output into a local file by clicking 💾. This file (`tsox.out`) is placed into a new project named `LocalProj`.



```
*** TSO commands require an MVS System that is connected and is selected as current in z/OS

IKJ56192I JOB ADT01J(JOB12798) ON OUTPUT QUEUE
IKJ56192I JOB ADT01J(JOB12800) ON OUTPUT QUEUE
IKJ56192I JOB ADT01J(JOB12806) ON OUTPUT QUEUE
IKJ56192I JOB ADT01X(JOB12813) ON OUTPUT QUEUE

 IKJ56650I TIME-04:51:17 PM. CPU-00:00:00 SERVICE-29415 SESSION-00:00:02 OCTOBER 11,2
```

Figure 14-37 z/OS Commands view

Remote debugging

In this section, we look at the remote debugger. First we set up the run-time options, then we start the remote program and debug it.

Preparation

Before we can start a debug session, we have to set up z/OS run-time options. We suggest you do that at the project level by selecting the `ItsoCobolzOS` project and *Properties -> z/OS Run Options*:

- Select *Run in the debugger*.
- If required specify run-time environment libraries.

  We select `CUST.V2R10M0.SCEERUN`, the Language Environment V2.1 run-time library.
- If required, specify run-time environment options (nothing specified in our case).
- Code additional JCL in the appropriate space. Besides specifying steplib(s) required at execution time, you have to specify the program input (Figure 14-38).

```
//******ADDITIONAL JCL HERE*******
// DD DSN=CUST.V2R10M0.SCEERUN,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 HEKKENBERG
 Q
/*
```

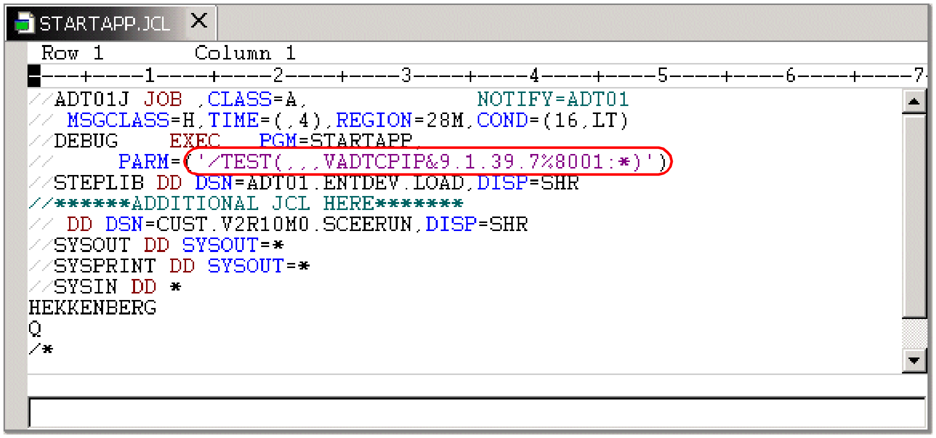
Figure 14-38 Additional JCL in z/OS run-time options

Debugging the remote executable

A debug session on our executable can now be started by:

- Selecting the `STARTAPP.exe` file in `ENTDEV.LOAD` and *Debug Application* (context).
- In the background, this results in submitting `STARTAPP.JCL`, which is generated into the `zOSGeneratedJCLProj` project (Figure 14-39).

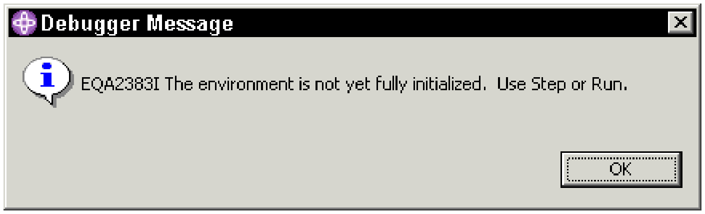
  Notice the `/TEST` option with your own TCP/IP address and the port number (`8001`), and the additional JCL that we provided. The port number is set by selecting *Window -> Preferences -> Debug -> Debug Daemon*.



```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7
//ADT01J JOB ,CLASS=A,               NOTIFY=ADT01
// MSGCLASS=H,TIME=(,4),REGION=28M,COND=(16,LT)
//DEBUG    EXEC   PGM=STARTAPP,
//        PARM=('/TEST(,,,VADTCPIP&9.1.39.7%8001:*)')
//STEPLIB DD DSN=ADT01.ENTDEV.LOAD,DISP=SHR
//******ADDITIONAL JCL HERE*******
// DD DSN=CUST.V2R10M0.SCEERUN,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
HEKKENBERG
Q
/*
```

Figure 14-39 JCL to start the remote debug session

- Clicking *OK* in the dialog that is displayed (Figure 14-40)

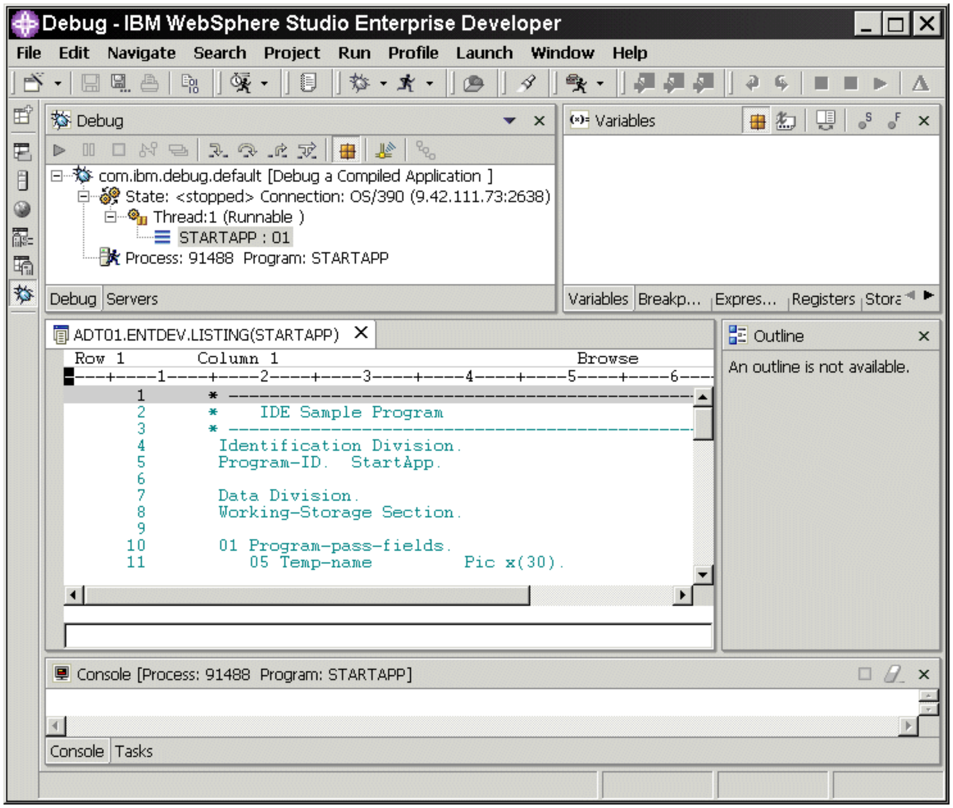


EQA2383I The environment is not yet fully initialized. Use Step or Run.

Figure 14-40 Debugger message on startup

Debugging the program

Now the debugger should be started with the listing of `StartApp` opened (Figure 14-41).



```
com.ibm.debug.default [Debug a Compiled Application ]
  State: <stopped> Connection: OS/390 (9.42.111.73:2638)
    Thread:1 (Runnable )
      STARTAPP : 01
  Process: 91488  Program: STARTAPP
```

An outline is not available.

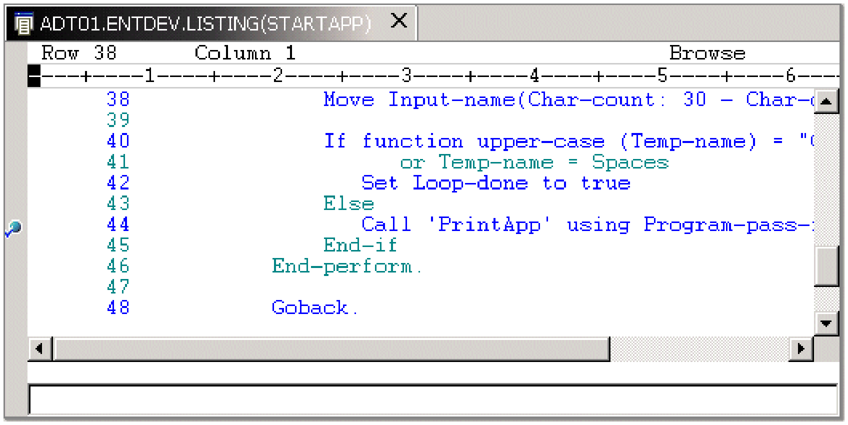
```
Row 1        Column 1                             Browse
----+----1----+----2----+----3----+----4----+----5----+----6----
      *  ---------------------------------------------------
      *     IDE Sample Program
      *  ---------------------------------------------------
       Identification Division.
       Program-ID.  StartApp.

       Data Division.
       Working-Storage Section.

       01 Program-pass-fields.
          05 Temp-name           Pic x(30).
```

Console [Process: 91488 Program: STARTAPP]

Figure 14-41 Remote debug: after startup (batch debug)

Follow the next steps to debug the program:

- Set a breakpoint in the source on line 44 by double-clicking in the grey area beside the line (Figure 14-42).

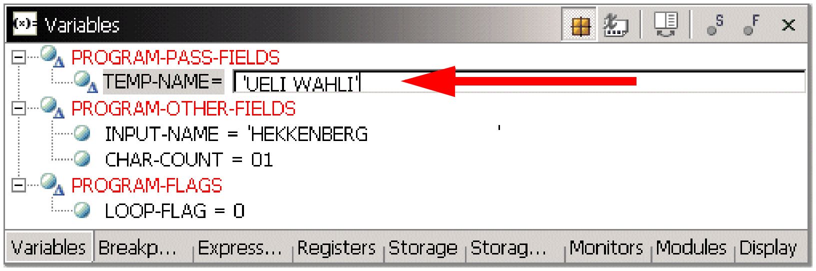


ADT01.ENTDEV.LISTING(STARTAPP)

```
Row 38          Column 1                                     Browse
----+----1----+----2----+----3----+----4----+----5----+----6----
      38                 Move Input-name(Char-count : 30 - Char-(
      39
      40                 If function upper-case (Temp-name) = "(
      41                       or Temp-name = Spaces
      42                    Set Loop-done to true
      43                 Else
      44                    Call 'PrintApp' using Program-pass-:
      45                 End-if
      46             End-perform.
      47
      48             Goback.
```

Figure 14-42 Remote debug: set breakpoint

- Run the debugger until the breakpoint by clicking ▶ (the *Resume* icon).
- In the Variables view, change the `TEMP-NAME` variable by double-clicking and change `HEKKENBERG` to ‘`YOURNAME`’ (your own name), including quotes (Figure 14-43).



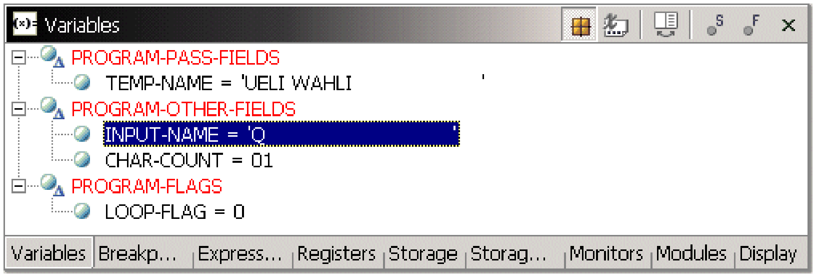
Variables

```
PROGRAM-PASS-FIELDS
    TEMP-NAME= 'UELI WAHLI'
PROGRAM-OTHER-FIELDS
    INPUT-NAME = 'HEKKENBERG                    '
    CHAR-COUNT = 01
PROGRAM-FLAGS
    LOOP-FLAG = 0
```

Figure 14-43 Remote debug: change variable

- Step into the `PrintApp` program by clicking (the *Step Into* icon).

  Notice that variable view is changing. Keep stepping through the code by clicking (the *Step Over* icon) until you are back in `StartApp` and continue until you see **Q** as the value in the `INPUT-NAME` variable (Figure 14-44).



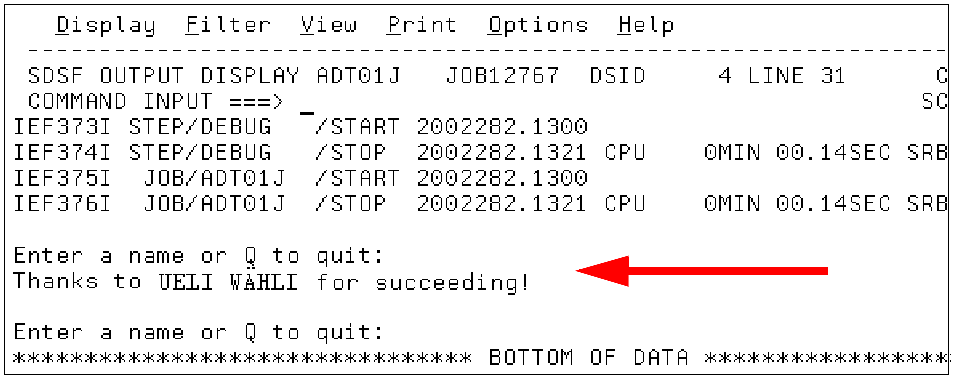
```
PROGRAM-PASS-FIELDS
    TEMP-NAME = 'UELI WAHLI                  '
PROGRAM-OTHER-FIELDS
    INPUT-NAME = 'Q                          '
    CHAR-COUNT = 01
PROGRAM-FLAGS
    LOOP-FLAG = 0
```

Figure 14-44 Remote debug: variables view before ending

Q indicates that loop should finish and the loop flag is raised. You can step through the program or click (the *Step Return* icon) to jump to the end.

Debug job output

One way of looking at the output of the debug job would be to go to SDSF on the mainframe and have a look (Figure 14-45). Notice `YOURNAME` in the output.

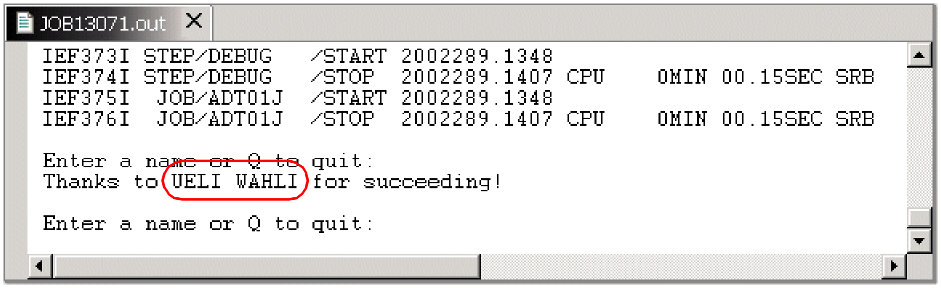


```
   Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------
 SDSF OUTPUT DISPLAY ADT01J   JOB12767  DSID     4 LINE 31       C
 COMMAND INPUT ===> _                                            SC
IEF373I STEP/DEBUG  /START 2002282.1300
IEF374I STEP/DEBUG  /STOP  2002282.1321 CPU    0MIN 00.14SEC SRB
IEF375I  JOB/ADT01J /START 2002282.1300
IEF376I  JOB/ADT01J /STOP  2002282.1321 CPU    0MIN 00.14SEC SRB

Enter a name or Q to quit:
Thanks to UELI WAHLI for succeeding!

Enter a name or Q to quit:
******************************** BOTTOM OF DATA ******************
```

Figure 14-45 Remote debug: job output in SDSF

However, we have shown in "Job and command interactions with z/OS" on page 386 that you do not have to leave Enterprise Developer to look at your output. Use the z/OS Job Monitor view and retrieve the output of the newest job:

- In the z/OS Perspective, refresh the job display in the job monitor by clicking (the *Refresh* icon).
- Double-click the top job, and `jobnumber`.out file is opened (Figure 14-46)



```
IEF373I STEP/DEBUG   /START 2002289.1348
IEF374I STEP/DEBUG   /STOP  2002289.1407 CPU    0MIN 00.15SEC SRB
IEF375I  JOB/ADT01J  /START 2002289.1348
IEF376I  JOB/ADT01J  /STOP  2002289.1407 CPU    0MIN 00.15SEC SRB

Enter a name or Q to quit:
Thanks to UELI WAHLI for succeeding!

Enter a name or Q to quit:
```

Figure 14-46 Remote debug: job output through job monitor

Code maintenance scenario

Let's assume that our application is now in production and a requirement is raised to change the text that is returned by the application.

If you try to define another project to work with the same data sets, you will fail. Remember that data sets assigned to a project are removed from the z/OS Systems view.

Restriction: A given folder (MVS data set) can exist in only one project.

Changing projects

To illustrate the code maintenance task, we will use a new project. To use the same MVS data sets in the new project, we have two choices:

- Remove the `ItsoCobolzOS` project from the Workbench by selecting *Remove Project* (context). This action deletes all the local workstation files of the project.
- Select all the `ENTDEV.*` folders in the `ItsoCobolzOS` project and *Subtract from MVS Project* (context). This action deletes the local folders from the project. The project itself remains and folders can be assigned again later.

Both actions leave the z/OS data sets untouched and return the folders to the z/OS Systems view. Now these folders can be assigned to another project.

Important: Do not use the *Delete Project And Content* action, unless you are absolutely sure that you want to delete the local project with all folders and also the data sets with all the members on z/OS.

For our purposes, we remove the folders from the ItsoCobolzOS project and return them to the z/OS Systems view.

Define a new MVS project

Define a new MVS project called ItsoCobolMaintzOS. Follow the steps illustrated in “Creating and configuring a remote project” on page 373.

- Be sure to set the COBOL compiler options and linkage editor options.
- Select the ENTDEV.* folders in the z/OS Systems view and *Add to Project*. (see “Map data sets for the remote project” on page 376).

Thus far, we have not discussed and illustrated the edit and syntax check capabilities within the z/OS perspective of Enterprise Developer. Let us do that now.

Editing

Editors are always up for debate. For our COBOL programs, we can choose between:

- JLPEX—Java LPEX editor with limited features, integrated into the Workbench
- LPEX—LPEX editor, full function, non-integrated, must refresh after save
- Text—simple text editor, integrated into the Workbench
- System—defaults to file associations, in this case the LPEX editor is used

JLPEX editor

Here we discuss some of the features of the LPEX editor that comes with Enterprise Developer. By no means do we discuss all functions and features. It will be up to the developers to decide how they will exploit its functionality.

Colors

The editor uses colors to distinguish the various language constructs.

Preferences

The preferences for the editor are set through *Window -> Preferences -> LPEX Editor*.

- On the LPEX Editor page, you can set the editor profile, that is, the personality. Instead of the default *lpex*, you could use *ispf* or *xedit* personality.
- On the Appearance page, fonts can be set.
- On the Controls page, you can set if a status, format, command line, or a prefix area is displayed.

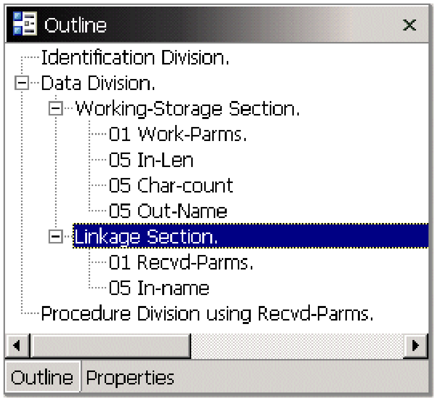
- On the User page, you can tailor the behavior even further by adding all sorts of user-defined actions and scripts.
- On the Parser Associations page, file types are associated to available parsers.
- On the Parsers page ,all the available parsers are displayed.

  These parsers enable the language sensitivity of the editor and provide the means to be able to show the Outline view in the perspective.

Outline view

The Outline view is a powerful instrument to navigate through the source code. The editor will reposition itself depending on what language construct is selected in the outline view.

Figure 14-47 is a representation of the outline view of our `PrintApp` program.



```
Identification Division.
Data Division.
  Working-Storage Section.
    01 Work-Parms.
    05 In-Len
    05 Char-count
    05 Out-Name
  Linkage Section.
    01 Recvd-Parms.
    05 In-name
Procedure Division using Recvd-Parms.
```

Figure 14-47   Outline view of JLPEX editor

Editor context menu

The editor context menu has a lot of features that are available to you depending on where you are positioned in the file and what you have selected.

Notes:

- While you are editing a file, the source code is in memory. It will be saved on disk only if FFS experiences problems. In that case, the source can be found on the local disk.
- Open editor sessions are reflected in the *Preferences* pages as well. `\\ffssash\` indicates that the source is rendered to the editor through FFS. A local source edit session would have the path of your workspace instead.

We have only scratched the surface here. You and your development support people can tailor the behavior to your projects and individual needs, thus optimizing the value-add of the editor within your environment.

Syntax check

Another feature that we have not discussed thus far is the capability of performing a syntax check on the source. Syntax checking can reduce to only one the number of compile jobs needed to get a clean compile.

When you submit a compile job after a clean syntax check, you are assured that the source is correct from a language syntax point of view.

For local projects, syntax check is done automatically when you save the file. For remote projects, no local file exists and a syntax check must be invoked explicitly.

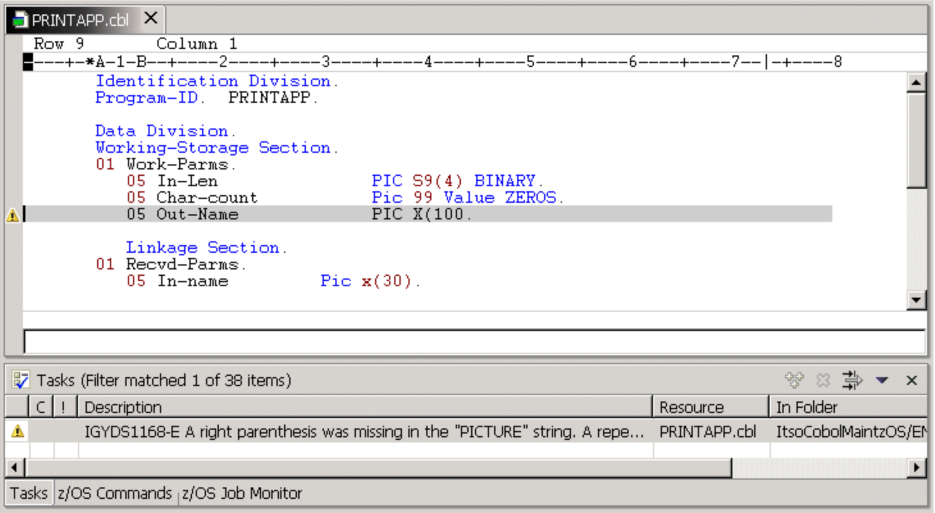
Let's see syntax checking at work:

- Open a JLPEX editor on `PRINTAPP.CBL`.
- Remove the right parenthesis on the PIC clause in line 9:

```
05 Out-Name             PIC X(100.
```

- Save the change.
- Select *SyntaxCheck* from the context menu with `PRINTAPP.cbl` selected.

  Notice that in the Tasks view a message appears and an icon is displayed at line 9 (Figure 14-48).



```
Row 9        Column 1
----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7--|-+----8
       Identification Division.
       Program-ID.  PRINTAPP.

       Data Division.
       Working-Storage Section.
       01 Work-Parms.
          05 In-Len                PIC S9(4) BINARY.
          05 Char-count            Pic 99 Value ZEROS.
          05 Out-Name              PIC X(100.

          Linkage Section.
       01 Recvd-Parms.
          05 In-name          Pic x(30).
```

Tasks (Filter matched 1 of 38 items)

	C	!	Description	Resource	In Folder
⚠			IGYDS1168-E A right parenthesis was missing in the "PICTURE" string. A repe...	PRINTAPP.cbl	ItsoCobolMaintzOS/EI

Figure 14-48 Error message from syntax check

Note: Under the covers, the local compiler is invoked to enable this. With the early availability level of the code, you can see this due to the fact that two command windows are displayed, one for retrieving the code through FSS, and another for invoking the local compiler.

If you receive a REXX version conflict window, see “Potential REXX conflict” on page 359 for a solution.

- Double-clicking the error in the Tasks view brings you to the line in the editor that contains the error. If the editor is not open, it is opened as well.

  This is a real productivity feature. Compare this with the steps you need to take in ISPF to accomplish the same.

Note: The syntax check action creates a new *Syntax folder* in the `zOSOutputProj`. It holds the output of the invocation of the local compiler needed for the actual syntax check.

- Correct the error by adding a right parenthesis.

Implement the requirement and test the application

Fulfill the application requirement by changing the strings:

```
"Thanks to " ==> "Hello "
" for succeeding" ==> " You did it!"
```

Build the application

Define the remote build server to the project. See “Define a distributed build server” on page 378 if you don’t remember.

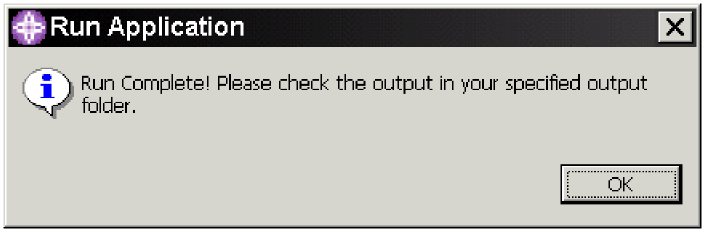
Tip: In a production environment it is probably better to have a standard build server defined in the preferences and there would be no need to define it now as a project property.

Perform *Rebuild Project* to trigger a complete build and check the output.

Run the application

Set up run-time options for the application (see “Debugging the remote executable” on page 389).

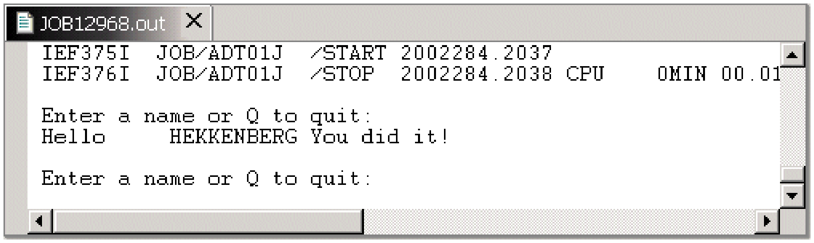
Select the `STARTAPP.EXE` file and run the application by selecting *Run Application* from the context menu. If successfully submitted, a message box will be displayed (Figure 14-49). Click *OK*.



Run Complete! Please check the output in your specified output folder.

Figure 14-49 Run Application message box

Refresh the z/OS Job Monitor view. Look at the bottom of the output of the newest job (Figure 14-50).



```
IEF375I  JOB/ADT01J  /START 2002284.2037
IEF376I  JOB/ADT01J  /STOP  2002284.2038 CPU    0MIN 00.01

Enter a name or Q to quit:
Hello     HEKKENBERG You did it!

Enter a name or Q to quit:
```

Figure 14-50 Output of changed application

This is according to the specifications, so your maintenance project is finished now!

Copy remote project to local for offline work

It is possible to copy projects between local and remote state. This enables a developer to work offline for a while and afterwards copy the work back to z/OS.

Copy remote project to local

To copy a remote project to local:

- Start the local build server (see “Defining and starting a local build server” on page 355). A build is invoked after the files are copied.
- Select *New -> Other -> z/OS -> Copy z/OS Project*.
- Select the remote project (for example `ItsoCobolMaintzOS`), *Local* as target, and enter a local project name (for example, `ItsoCobolMaintLocal`).

- Click *Finish*.

> **Important:** Be patient; the operation takes a while to complete. You may encounter pop-up windows for the CCU Security Manager (see Figure 14-30 on page 380), which you have to close. Otherwise the operation does not complete.

As a result of the operation a local project is created. The folders in the local project have the names of the z/OS folders, as they are in the remote project.

Browse the `BuildResults.xml` file for error messages.

Figure 14-51 shows the resulting local project.

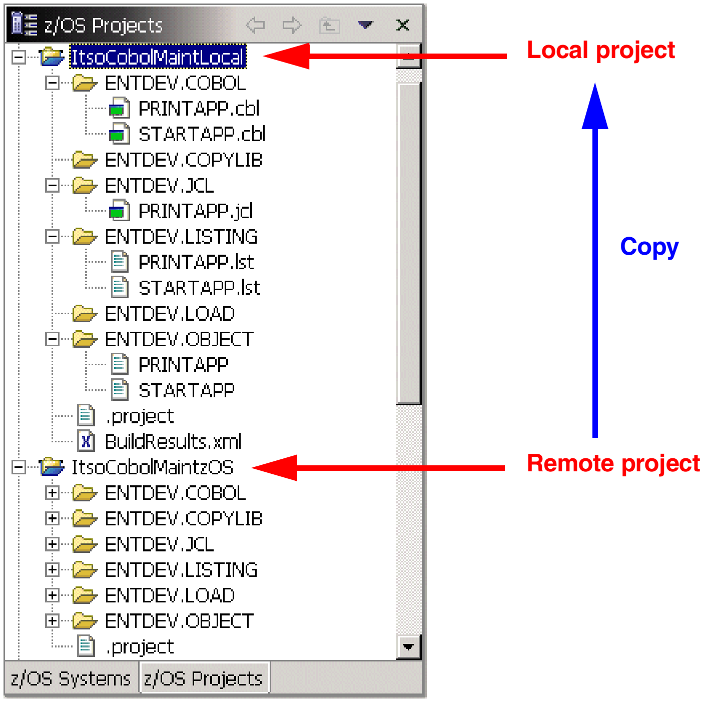


Figure 14-51 Local project as result of copy from remote

Work offline

For example, edit the `PRINTAPP.CBL` file to reply with *Hello again* (instead of *Hello*).

Copy local project to remote

To copy a local project to remote:

- Select *New -> Other -> z/OS -> Copy z/OS Project*.
- Select the local project (for example `ItsoCobolLocal`), *Remote* as target, and enter a remote project name (for example, `ItsoCobolMaintzOS`). Select the target system and directory (high-level qualifier).
- Click *Finish*.

> **Note:** The Tasks list shows errors for files that cannot be stored on the remote MVS system.

We get error messages for the `BuildResults.xml` file and for the object code, which has no extension in the local project.

Delete the file and the two object modules in `ENTDEV.OBJECT`. Rerun the copy.

> **Important:** This process produces error messages that the PDSs already exist on the z/OS system, and the result is that the files are not copied. **Updates are not supported in the current level of the product.**

Summary

In this chapter we provided a guided tour on the setup and exploitation of the functionality available within the z/OS perspectives of Enterprise Developer. We did that bearing in mind that this way of working with mainframe artifacts, combining the strength of two platforms, is quite different from using a mainframe development environment such as ISPF.

We also focused on the RECD scenario. In the last section we walked through a maintenance scenario. We hope you enjoyed this tour and have a big appetite to explore more on your own. There is plenty more to discover!

15

XML enablement for COBOL

In this chapter, we describe how IBM WebSphere Studio Enterprise Developer helps you in modernizing your Enterprise assets and adapting them to process and produce XML messages. The following topics are discussed:

- XML enablement architecture, its benefits, the run-time scenarios, and current limitations
- A complete example of modernizing an existing COBOL CICS application with instructions on how to modify the generated code

Introduction

Since the advent of the World Wide Web, mainframe applications, which used to rely only on binary interfaces for communication, have been trying to use XML as a new means of information exchange. This approach presents challenges to the programmers who are trying to efficiently adapt business applications in order to process and produce XML documents with minimal disruptions to the existing system infrastructure. There have been requests from IBM's enterprise customers to support the processing of XML messages from COBOL.

XML enablement for the enterprise lets you easily adapt existing COBOL-based business applications so that they can process and produce XML messages into native COBOL data, and to transform COBOL data into an XML output message. The converter programs use the new high-performance XML parsing capabilities of the IBM Enterprise COBOL compiler. The tool also generates a template COBOL program that shows how to invoke the converter along with the existing application.

By using XML enablement for the enterprise, you can use data produced by existing COBOL programs to communicate with systems that use XML for data interchange, including systems based on Web services. The XML parsing is done using a simple API for XML (SAX) parser.

Benefits of XML enablement

Existing COBOL applications can be modernized so they can communicate with J2EE applications or handle XML messages. XML enablement makes it a lot easier to provide an interface to existing COBOL applications to support XML messages.

The generation of converter and template driver programs in COBOL eliminates the time-consuming and error-prone part of using the XML support provided by the IBM Enterprise COBOL V3.1. Keeping things in COBOL leverages the customers' existing assets and skills. Overall, this contributes to measurable programmer productivity gains.

The XML parsing and conversions run on the z/OS system, leveraging the high-performance XML parser to realize performance gains.

Enabling XML for existing COBOL applications

Enterprise Developer has the possibility to create new COBOL programs that generate XML data, using existing COBOL programs as input. Figure 15-1 shows an overview of the XML enabling architecture.

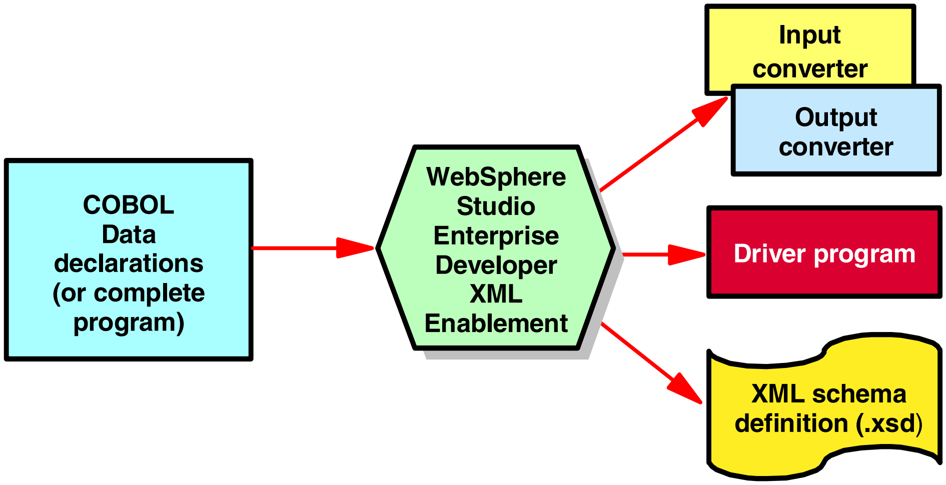

COBOL Data declarations (or complete program)
WebSphere Studio Enterprise Developer XML Enablement
Input converter
Output converter
Driver program
XML schema definition (.xsd)

Figure 15-1 XML enablement architecture

The input to the XML enablement tool is either the COBOL data structure, for example a COMMAREA, or the complete COBOL program.

The output consists of input and output converter programs, a driver program, and an XML schema. In the current version of the product (Early Availability), three output files are created after executing the XML transformation:

- The converter program, which contains both the inbound XML message processor that converts XML to native COBOL data and the outbound XML producer that converts native COBOL data to XML.
- The driver program, which is a template that has to be modified. It illustrates how the input and output converters are used in conjunction with the unchanged COBOL program. It also shows a way of doing error handling that can be adapted to suit the needs of the application.
- The document schema definition that describes the XML message that corresponds to the COBOL data structure we are using. This can be used to validate the input XML messages.

Note: The generated inbound and outbound programs will be in separate files when the Enterprise Developer is released (general availability).

z/OS prerequisites for XML enabling

The following software is required to develop and run the XML enabling feature of Enterprise Developer:

- IBM Enterprise COBOL for z/OS and OS/390 Version 3 Release 1 (program number 5648-A25) or later
- IBM Language Environment for OS/390 Version 2 Release 10 (program number 5647-A01) or later with PTF for APAR PQ65085 (available September 2002)
- OS/390 R8/R9/R10 and z/OS V1R1 support for Unicode is required for the XML converters generated by Enterprise Developer General Availability release

Using the generated code

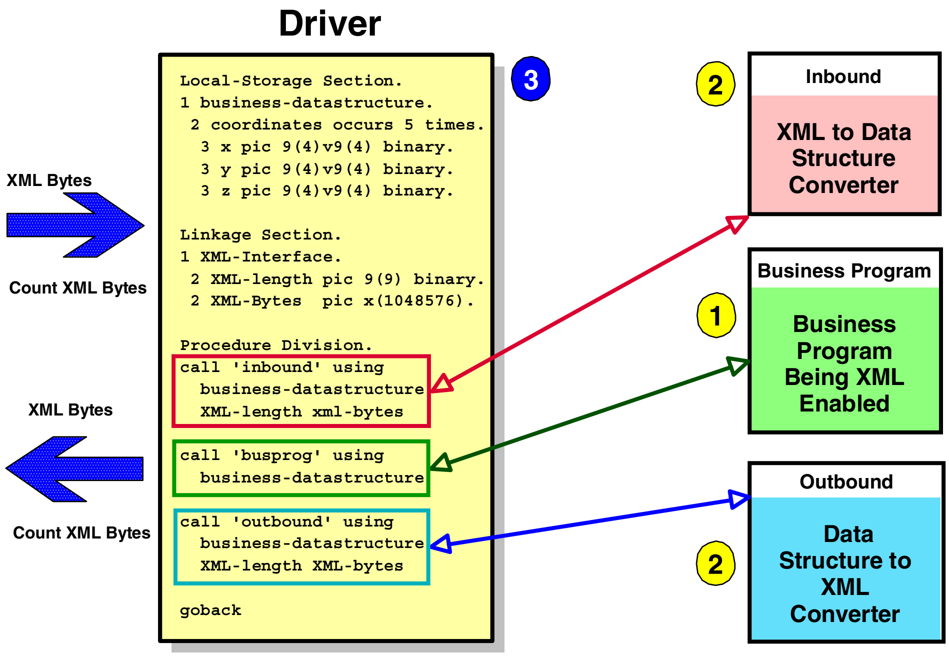
After code generation it is the user's responsibility to change the generated COBOL driver program to invoke the original business program, depending on the environment (batch, CICS) of that program.

The generated COBOL converters should not be modified. These converters are also known as inbound/outbound programs.

Figure 15-2 shows how the generated COBOL programs are used:

- The existing legacy business COBOL program (1) is the program being XML enabled. This program will be used by Enterprise Developer as input and no modifications are necessary on this program.
- The converter programs (2) are generated and will be called by the driver program (3) to perform the data transformation.
- The driver program (3) must be updated and modified to be able to call the existing business program (1).

All new systems that require the existing data in XML format will use the driver program instead of the existing business program.


Driver
3
XML Bytes
Count XML Bytes
XML Bytes
Count XML Bytes
2
Inbound
XML to Data Structure Converter
Business Program
1
Business Program Being XML Enabled
Outbound
2
Data Structure to XML Converter

```
Local-Storage Section.
1 business-datastructure.
 2 coordinates occurs 5 times.
  3 x pic 9(4)v9(4) binary.
  3 y pic 9(4)v9(4) binary.
  3 z pic 9(4)v9(4) binary.

Linkage Section.
1 XML-Interface.
 2 XML-length pic 9(9) binary.
 2 XML-Bytes  pic x(1048576).

Procedure Division.
call 'inbound' using
  business-datastructure
  XML-length xml-bytes

call 'busprog' using
  business-datastructure

call 'outbound' using
  business-datastructure
  XML-length XML-bytes

goback
```

Figure 15-2 Using the XML converters

XML enablement run-time scenarios

The same XML COBOL converters can be used (called) by CICS, IMS, and batch applications that take XML messages as input.

Figure 15-3 shows three scenarios where we could use the XML converters:

- **Scenario 1**—Shows a CICS or IMS transaction being invoked from a Web service through a JCA connector. This transaction will call the input XML converter to map the XML message into native COBOL data before passing that COBOL data to the existing COBOL application or transaction.
- **Scenario 2**—Is slightly different from scenario 1 in that an application server is not involved. The XML message may come from some source, say another CICS application or a WebSphere MQ application.
- **Scenario 3**—Shows a new batch COBOL application working with XML input and interfacing with the existing COBOL application.

Notice that a mid-tier application server is not a prerequisite to running the XML converters generated by the XML enablement tool.

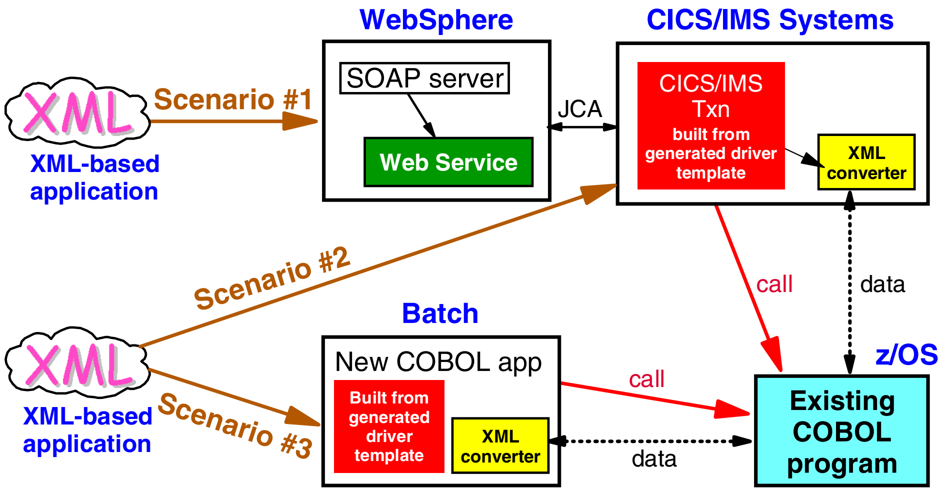


Figure 15-3 Run-time scenarios for XML enablement

General limitations

Some limitations apply when using the XML enablement tool with Enterprise Developer:

- The source and target, that is, the input and output to the XML enablement tool, must reside on the local workstation and not on a remote z/OS system.
- Copy books must be fully expanded, that is, they must be inline. This is a restriction imposed by the COBOL importer.
- The workstation COBOL compiler that comes with WebSphere Studio Enterprise Developer has not been upgraded to the Enterprise COBOL 3.1 level and thus has no XML support. Consequently, you cannot run the generated code on a workstation.
- COMP-X is a MicroFocus extension, not supported by the IBM host compilers. This is an issue only if porting a COBOL program written using MicroFocus to the z/OS platform.
- Mapping of XML element attributes are not supported.
- Only the original data items are handled; redefined data items are ignored. For example, if `A` is defined as `PIC X(30)` and `B` redefines `A` as `PIC X(20)`, data item `B` is ignored by the tool.

Early availability limitations

The version used in this book is the early availability version that has additional limitations:

- Online help is not available for the XML enablement tool. Refer to the *XML for the Enterprise* white paper for additional information on tool usage (see “Installing the CICS application sample in z/OS” on page 408).
- Inbound message processing Unicode UTF-16 is not supported.
- Outbound message generation limitations:
  - Simple *occurs-depending-on* (ODO) not supported.
  - Trailing/leading blanks in character content are not removed.
  - Trailing/leading zeroes in numeric content are not removed.
  - The characters <, >, ', ", & are not allowed in character content.

Sample application topology

The best way to understand how the XML transformation works is by using the sample COBOL programs that are shipped with the Enterprise Developer on the third CD in the directory `..\Samples\XMLEnterprise`. Figure 15-4 shows how these programs interact with each other.

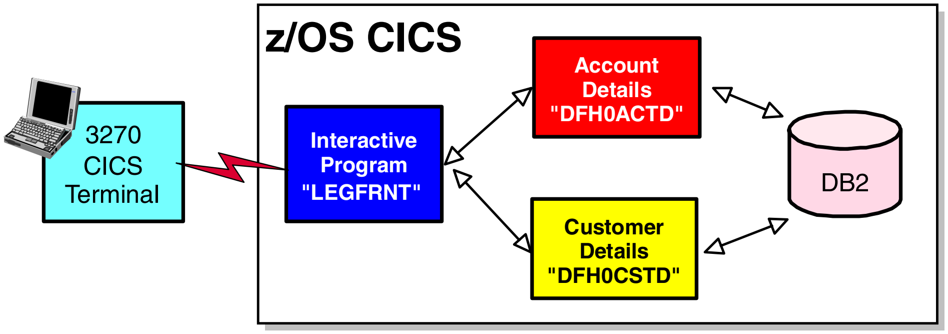

z/OS CICS
3270 CICS Terminal
Interactive Program "LEGFRNT"
Account Details "DFH0ACTD"
Customer Details "DFH0CSTD"
DB2

Figure 15-4   Sample CICS application distributed with Enterprise Developer

This CICS application consists of an interactive program, `LEGFRNT`, which calls two CICS programs, `DFH0ACTD` and `DFH0CSTD`. They, in turn, access DB2 tables to retrieve customer and account information. This information is exchanged in binary format in the CICS COMMAREA and the results are displayed on a 3270 terminal. Screens that illustrate the interaction are shown in “Running the existing 3270 CICS legacy application” on page 408.

Installing the CICS application sample in z/OS

To install and run this sample program in the z/OS system, you must follow the instructions detailed in *XML for the Enterprise - Providing an XML Interface to a CICS Application*, found in the root of the CD where you get the samples or at the Web site:

http://www-3.ibm.com/software/ad/studioenterprisedev/library/

Here are the required COBOL programs for the current application:

- DFH0ACTD (account details sample program)
- DFH0CSTD (customer details sample program)
- LEGFRNT (CICS front-end program for executing the business programs)
- LEGMAP (CICS BMS map for front-end program)
- DFH$EDB2 (creates the DB2 tables for the sample programs)
- DFH$ESQL (DB2 bind for the sample programs DFH0ACTD and DFH0CSTD)
- XML$CEDA (creates CICS table entries)

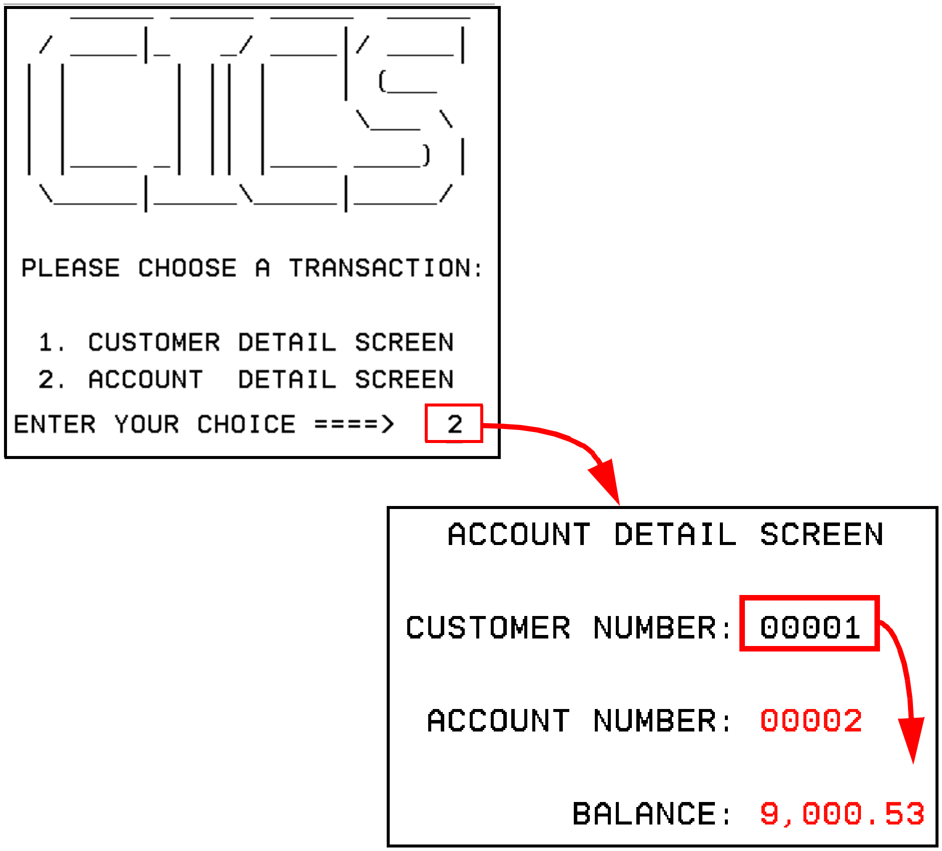
These COBOL programs are included in the Enterprise Developer samples. The following products are prerequisites for this example:

- IBM Database 2 Universal Database Server for OS/390 (DB2) Version 6 Release 1 (program number 5675-DB2) or later
- CICS Transaction Server for OS/390 Version 1 Release 3 (program number 5655-147) or later

Running the existing 3270 CICS legacy application

To start the application, bring up a CICS terminal and connect to the CICS system:

- Run the transaction LEGF.
- The first screen asks for entering the transaction type.
- Enter 2 to execute the program DFH0ACTD and press *Enter*.
- The second screen appears. Enter a valid customer number, for example, 00001 and press *Enter*. The account number and balance are displayed.
- Figure 15-5 shows a sample run.



```
PLEASE CHOOSE A TRANSACTION:

 1. CUSTOMER DETAIL SCREEN
 2. ACCOUNT  DETAIL SCREEN
ENTER YOUR CHOICE ====>  2
```

```
  ACCOUNT DETAIL SCREEN

CUSTOMER NUMBER: 00001

 ACCOUNT NUMBER: 00002

        BALANCE: 9,000.53
```

Figure 15-5 CICS 3270 sample run

Requirements for changing the existing application

Any of the three scenarios described in “XML enablement run-time scenarios” on page 405 could be the requirement to change the existing application. The entire transformation can be done without having to change any existing program (`DFH0ACTD` and `DFH0CSTD`).

To allow XML documents to flow through to the existing business programs, the source of those programs is passed through the XML enablement tool. The tool generates a set of COBOL programs called *XML converters* (inbound and outbound) based on the original binary interface. The tool also generates a template COBOL program called *converter driver* that illustrates how to invoke the converters.

Figure 15-6 shows the structure of the modernized application.

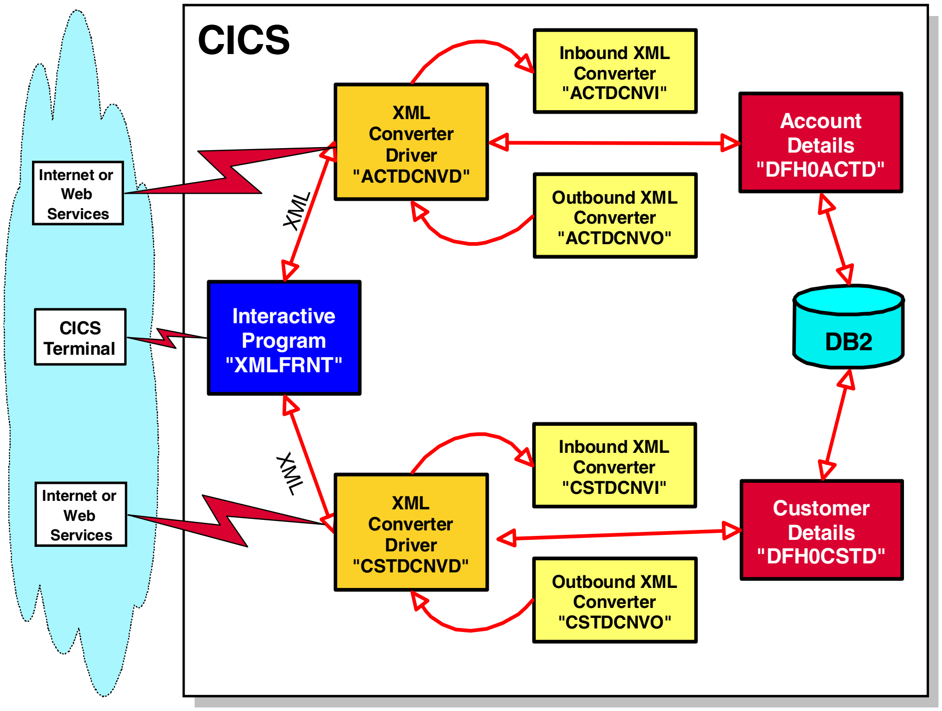

CICS
Internet or Web Services
CICS Terminal
Internet or Web Services
XML Converter Driver "ACTDCNVD"
Inbound XML Converter "ACTDCNVI"
Outbound XML Converter "ACTDCNVO"
Account Details "DFH0ACTD"
XML
Interactive Program "XMLFRNT"
DB2
XML
XML Converter Driver "CSTDCNVD"
Inbound XML Converter "CSTDCNVI"
Outbound XML Converter "CSTDCNVO"
Customer Details "DFH0CSTD"

Figure 15-6 Sample application enabled for XML

We show in “Modifying the converter driver programs” on page 418 how to extend the generated driver programs with EXEC CICS statements to call the existing business application in concert with calling the XML converters.

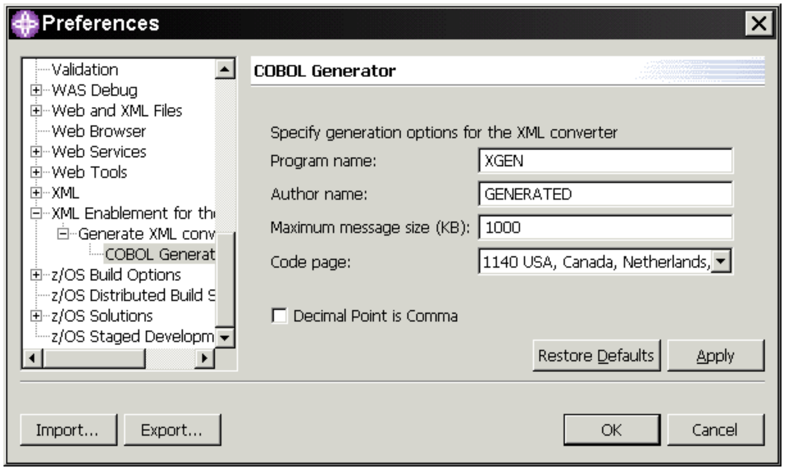
An interactive menu-driven 3270 front-end program (XMLFRNT) facilitates local testing of the new application. Note that this program is not generated by the Enterprise Developer and is used in our example as a test program.

XML enablement in Enterprise Developer

The XML enablement sample is provided with Enterprise Developer. In this section, we document how to generate the driver and converter programs from the sample COBOL code.

Preferences

Preferences for XML enablement can be set in Enterprise Developer by selecting *Window -> Preferences* (Figure 15-7).



COBOL Generator

Specify generation options for the XML converter

Program name: XGEN

Author name: GENERATED

Maximum message size (KB): 1000

Code page: 1140 USA, Canada, Netherlands,

□ Decimal Point is Comma

Figure 15-7 COBOL generator preferences for XML converter

Prepare a sample project

To load the XML sample, start the Enterprise Developer and use any perspective that has the Navigator view, for example the Resource perspective:

- Select *File -> New -> Project* and in the New Project pane select *Simple -> Project* and click *Next*.
- Enter `ItsoXMLConv` as the project name and click *Finish*.

Import the two legacy programs (`DFHOACTD.cbl` and `DFHOCSTD.cbl`) that will be called by the front-end interactive CICS programs:

- Select the `ItsoXMLConv` project and *Import* (context menu). Select *File system -> Next* and click *Browse* to locate the directory `Samples\XMLEnterprise` on the third distribution CD.
- Select the two programs `DFHOACTD.cbl` and `DFHOCSTD.cbl` and *Create selected folders only* and click Finish.

Figure 15-8 shows the result of the import.

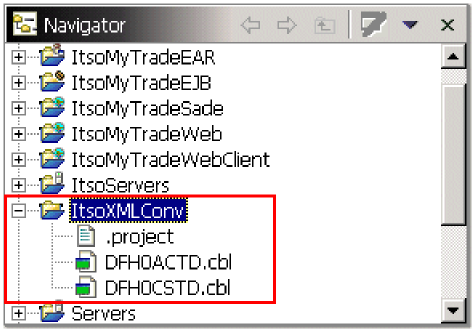
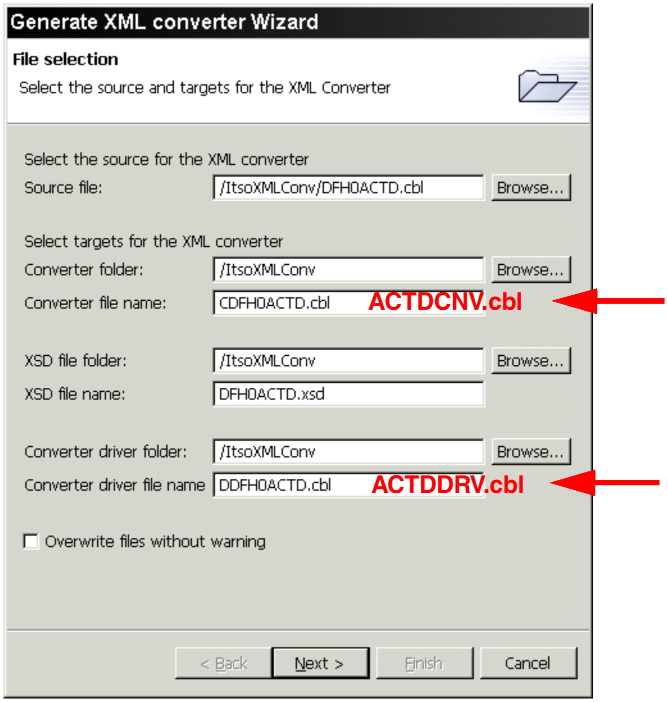


Figure 15-8 Imported COBOL source programs

Generating the XML converters and drivers

We use the XML converter wizard to generate the XML converters and drivers for the two sample programs:

- Select the `DFH0ACTD.cbl` program and *Enable XML -> Generate XML Converter* (context) to start the wizard (Figure 15-9).



Generate XML converter Wizard

File selection

Select the source and targets for the XML Converter

Select the source for the XML converter

Source file: /ItsoXMLConv/DFH0ACTD.cbl

Select targets for the XML converter

Converter folder: /ItsoXMLConv

Converter file name: CDFH0ACTD.cbl ACTDCNV.cbl

XSD file folder: /ItsoXMLConv

XSD file name: DFH0ACTD.xsd

Converter driver folder: /ItsoXMLConv

Converter driver file name DDFH0ACTD.cbl ACTDDRV.cbl

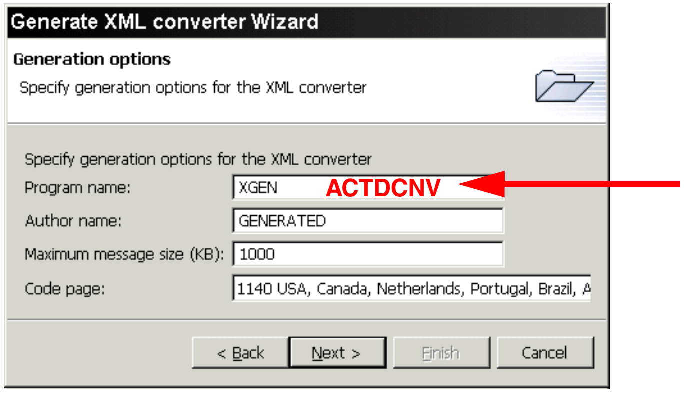
□ Overwrite files without warning

Figure 15-9 XML converter wizard file names

- This first page of the wizard is used to specify input and output file names. The default names for the converter and driver is the input file name with a C (converter) or D (driver) prefix.
- We change the file name for the converter programs to `ACTDCNV.cbl` and the file name for the driver program to `ACTDDRV.cbl`. We accept the default file name for the XML schema.

> **Important:** Because the file names could be greater than 8 characters long, you may run into problems when you try to move these into a z/OS partitioned data set (PDS). Therefore check the generated names!

- Click *Next*.
- For generation options (Figure 15-10) enter the program name `ACTDCNV` that will be used to create the PROGRAM-ID COBOL statements in the generated files. The actual program IDs will be formed with suffixes **I** an **O** for converters and **D** for the driver.



Generate XML converter Wizard

Generation options

Specify generation options for the XML converter

Specify generation options for the XML converter

Program name: XGEN ACTDCNV

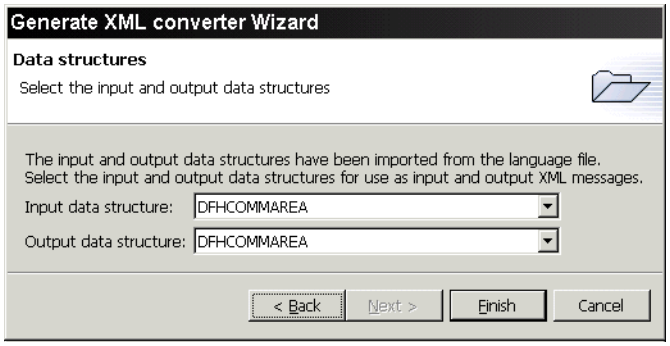
Author name: GENERATED

Maximum message size (KB): 1000

Code page: 1140 USA, Canada, Netherlands, Portugal, Brazil, A

Figure 15-10 XML converter wizard generation options

- In the data structures dialog, select `DFHCOMMAREA` as input and output data structure (Figure 15-11) because this is the structure to be converted.
- Click *Finish* to complete the XML generation.


Generate XML converter Wizard
Data structures
Select the input and output data structures

The input and output data structures have been imported from the language file.
Select the input and output data structures for use as input and output XML messages.

Input data structure: DFHCOMMAREA

Output data structure: DFHCOMMAREA

< Back
Next >
Finish
Cancel

Figure 15-11 XML converter wizard data structure

When the generation is finished, you will find the three generated files in the Navigator view.

Repeat this operation by starting the wizard for the `DFHOCSTD.cbl` program:

- Enter `CSTDCNV.cbl` and `CSTDDRV.cbl` as the file output names.
- Enter `CSTDCNV` as the program name.
- Select `DHFCOMMAREA` for the input and output data structures.

Understanding the generated code

Table 15-1 shows the XML converter input and generated output.

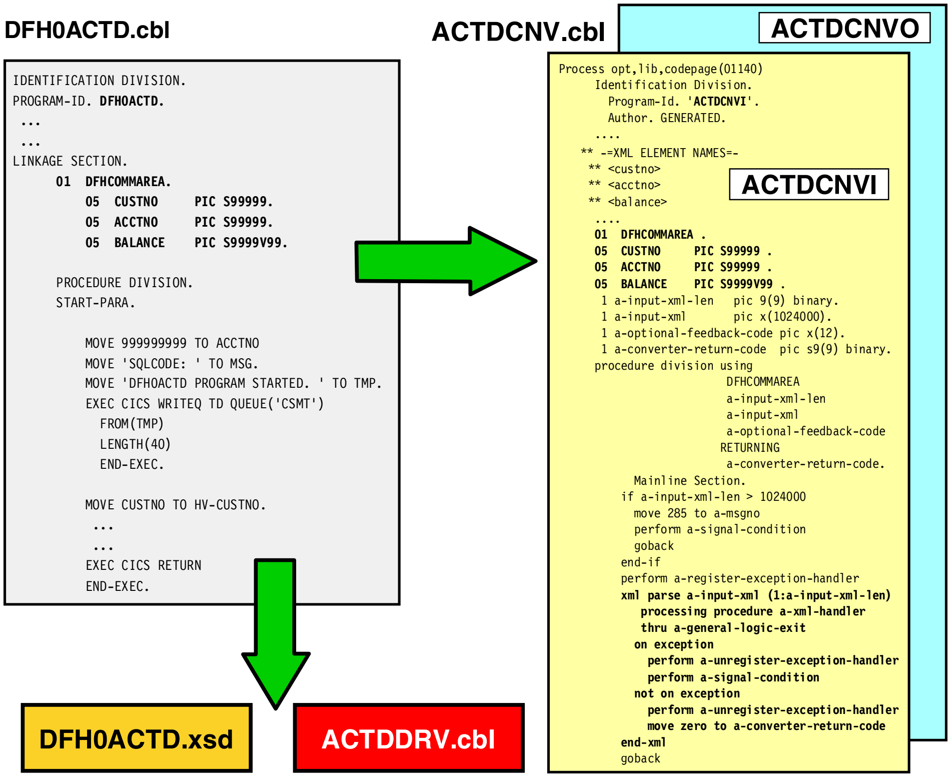
Table 15-1 XML enablement generated code

Input program	Generated converter - program IDs	Generated driver - program IDs	Generated XML schema
DFHOACTD.cbl	ACTDCNV.cbl - ACTDCNVI - ACTDCNVO - ACTDCNVN	ACTDDRV.cbl - ACTDCNVD - ACTDCNVH	DFHOACTD.xsd
DFHOCSTD.cbl	CSTDCNV.cbl - CSTDCNVI - CSTDCNVO - CSTDCNVN	CUSTDDRV.cbl - CSTDCNVD - CSTDCNVH	DFHOCSTD.xsd

Note that the converter files hold three programs, the inbound (I) and outbound (O) converters and a utility program (N). The driver files hold two programs, the driver (D) and an exception utility program (H). This may change in the final Enterprise Developer product.

Inbound converter

Figure 15-12 shows the input COBOL program and the generated inbound converter program used in the sample.



```
IDENTIFICATION DIVISION.
PROGRAM-ID. DFH0ACTD.
 ...
 ...
LINKAGE SECTION.
      01  DFHCOMMAREA.
          05  CUSTNO     PIC S99999.
          05  ACCTNO     PIC S99999.
          05  BALANCE    PIC S9999V99.

      PROCEDURE DIVISION.
      START-PARA.

          MOVE 999999999 TO ACCTNO
          MOVE 'SQLCODE: ' TO MSG.
          MOVE 'DFH0ACTD PROGRAM STARTED. ' TO TMP.
          EXEC CICS WRITEQ TD QUEUE('CSMT')
            FROM(TMP)
            LENGTH(40)
            END-EXEC.

          MOVE CUSTNO TO HV-CUSTNO.
           ...
           ...
          EXEC CICS RETURN
          END-EXEC.
```

```
Process opt,lib,codepage(01140)
     Identification Division.
       Program-Id. 'ACTDCNVI'.
       Author. GENERATED.
     ....
   ** -=XML ELEMENT NAMES=-
    ** <custno>
    ** <acctno>
    ** <balance>
     ....
     01  DFHCOMMAREA .
     05  CUSTNO     PIC S99999 .
     05  ACCTNO     PIC S99999 .
     05  BALANCE    PIC S9999V99 .
      1 a-input-xml-len   pic 9(9) binary.
      1 a-input-xml       pic x(1024000).
      1 a-optional-feedback-code pic x(12).
      1 a-converter-return-code  pic s9(9) binary.
     procedure division using
                         DFHCOMMAREA
                         a-input-xml-len
                         a-input-xml
                         a-optional-feedback-code
                        RETURNING
                         a-converter-return-code.
           Mainline Section.
         if a-input-xml-len > 1024000
           move 285 to a-msgno
           perform a-signal-condition
           goback
         end-if
         perform a-register-exception-handler
         xml parse a-input-xml (1:a-input-xml-len)
            processing procedure a-xml-handler
            thru a-general-logic-exit
           on exception
             perform a-unregister-exception-handler
             perform a-signal-condition
           not on exception
             perform a-unregister-exception-handler
             move zero to a-converter-return-code
         end-xml
         goback
```

Figure 15-12 Input COBOL and XML generated inbound

The generated `ACTDCNVI` COBOL program processes incoming XML data and converts the contents of its elements into a COBOL data structure that matches the existing application. The inbound converter uses high-performance XML parsing capabilities of the latest IBM Enterprise COBOL compiler and run-time library. The conversion and moving of data is based on proprietary algorithms that provide high efficiency in transforming character data from the XML document into appropriate COBOL data.

Note that a new COBOL instruction *parse* is used in the Inbound program. It validates the incoming XML and in case of errors an exception is thrown (see an example in Figure 15-20 on page 422).

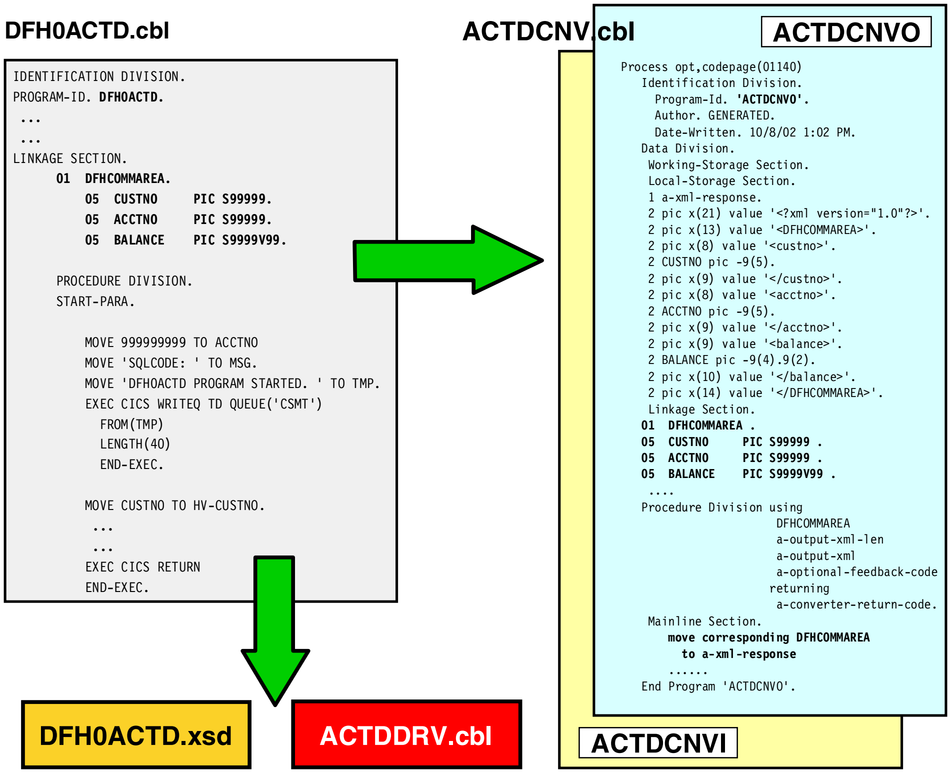
Example 15-1 shows the generated `parse` statement.

Example 15-1 New parse statement used in the converter

```
xml parse a-input-xml (1:a-input-xml-len)
              processing procedure a-xml-handler
              thru a-general-logic-exit
             on exception
               perform a-unregister-exception-handler
               perform a-signal-condition
             not on exception
               perform a-unregister-exception-handler
               move zero to a-converter-return-code
           end-xml
```

Outbound converter

Figure 15-13 shows the generated outbound converter program.



```
IDENTIFICATION DIVISION.
PROGRAM-ID. DFH0ACTD.
 ...
 ...
LINKAGE SECTION.
      01  DFHCOMMAREA.
          05  CUSTNO     PIC S99999.
          05  ACCTNO     PIC S99999.
          05  BALANCE    PIC S9999V99.

      PROCEDURE DIVISION.
      START-PARA.

          MOVE 999999999 TO ACCTNO
          MOVE 'SQLCODE: ' TO MSG.
          MOVE 'DFH0ACTD PROGRAM STARTED. ' TO TMP.
          EXEC CICS WRITEQ TD QUEUE('CSMT')
            FROM(TMP)
            LENGTH(40)
            END-EXEC.

          MOVE CUSTNO TO HV-CUSTNO.
           ...
           ...
          EXEC CICS RETURN
          END-EXEC.
```

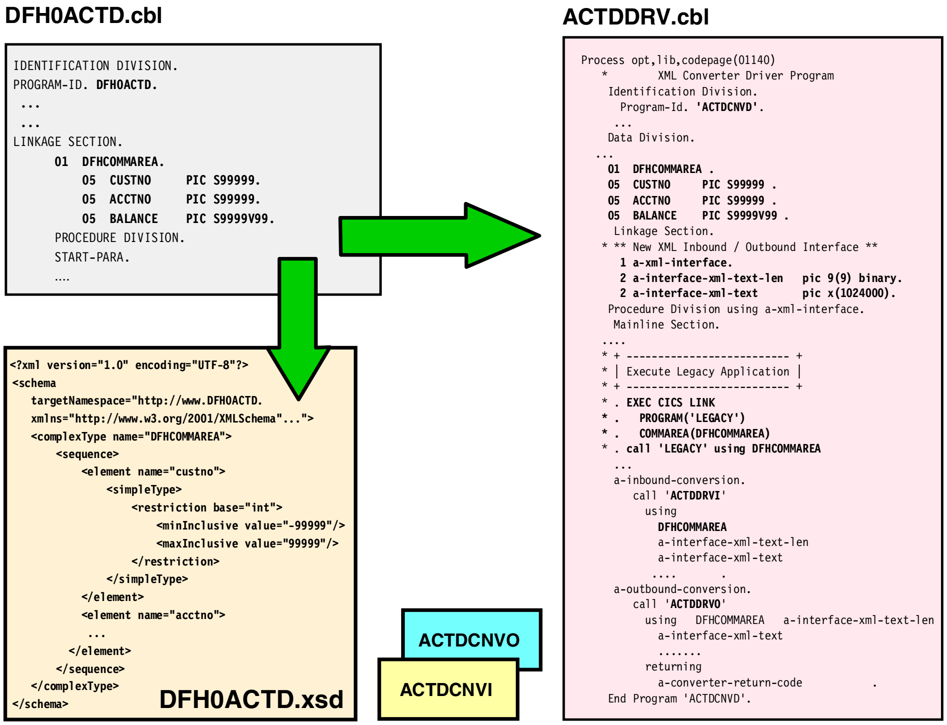
```
Process opt,codepage(01140)
   Identification Division.
     Program-Id. 'ACTDCNVO'.
     Author. GENERATED.
     Date-Written. 10/8/02 1:02 PM.
   Data Division.
    Working-Storage Section.
    Local-Storage Section.
    1 a-xml-response.
    2 pic x(21) value '<?xml version="1.0"?>'.
    2 pic x(13) value '<DFHCOMMAREA>'.
    2 pic x(8) value '<custno>'.
    2 CUSTNO pic -9(5).
    2 pic x(9) value '</custno>'.
    2 pic x(8) value '<acctno>'.
    2 ACCTNO pic -9(5).
    2 pic x(9) value '</acctno>'.
    2 pic x(9) value '<balance>'.
    2 BALANCE pic -9(4).9(2).
    2 pic x(10) value '</balance>'.
    2 pic x(14) value '</DFHCOMMAREA>'.
    Linkage Section.
   01  DFHCOMMAREA .
   05  CUSTNO     PIC S99999 .
   05  ACCTNO     PIC S99999 .
   05  BALANCE    PIC S9999V99 .
    ....
   Procedure Division using
                     DFHCOMMAREA
                     a-output-xml-len
                     a-output-xml
                     a-optional-feedback-code
                    returning
                     a-converter-return-code.
    Mainline Section.
       move corresponding DFHCOMMAREA
         to a-xml-response
       ......
   End Program 'ACTDCNVO'.
```

Figure 15-13 Input COBOL and XML generated outbound

The outbound converter takes the results of the original program and converts COBOL data into an XML message. This message is returned to the client. In case of an error during execution of the transaction, an XML-based error message is returned.

Converter driver and XML schema

Figure 15-14 shows the generated sample converter driver template and the XML schema. Note that the file name specified was ACTDDRV (Figure 15-10 on page 413), and the program ID is ACTDCNVD.



```
IDENTIFICATION DIVISION.
PROGRAM-ID. DFHOACTD.
 ...
 ...
LINKAGE SECTION.
      01  DFHCOMMAREA.
          05  CUSTNO     PIC S99999.
          05  ACCTNO     PIC S99999.
          05  BALANCE    PIC S9999V99.
      PROCEDURE DIVISION.
      START-PARA.
      ....
```

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   targetNamespace="http://www.DFHOACTD.
   xmlns="http://www.w3.org/2001/XMLSchema"...">
   <complexType name="DFHCOMMAREA">
       <sequence>
           <element name="custno">
               <simpleType>
                   <restriction base="int">
                       <minInclusive value="-99999"/>
                       <maxInclusive value="99999"/>
                   </restriction>
               </simpleType>
           </element>
           <element name="acctno">
            ...
         </element>
       </sequence>
   </complexType>
</schema>
```

```
Process opt,lib,codepage(01140)
   *        XML Converter Driver Program
    Identification Division.
      Program-Id. 'ACTDCNVD'.
     ...
    Data Division.
  ...
    01  DFHCOMMAREA .
    05  CUSTNO     PIC S99999 .
    05  ACCTNO     PIC S99999 .
    05  BALANCE    PIC S9999V99 .
     Linkage Section.
   * ** New XML Inbound / Outbound Interface **
      1 a-xml-interface.
      2 a-interface-xml-text-len   pic 9(9) binary.
      2 a-interface-xml-text       pic x(1024000).
    Procedure Division using a-xml-interface.
     Mainline Section.
   ....
   * + -------------------------- +
   * | Execute Legacy Application |
   * + -------------------------- +
   * . EXEC CICS LINK
   * .   PROGRAM('LEGACY')
   * .   COMMAREA(DFHCOMMAREA)
   * . call 'LEGACY' using DFHCOMMAREA
     ...
     a-inbound-conversion.
        call 'ACTDDRVI'
          using
            DFHCOMMAREA
            a-interface-xml-text-len
            a-interface-xml-text
           ....       .
     a-outbound-conversion.
        call 'ACTDDRVO'
          using   DFHCOMMAREA   a-interface-xml-text-len
            a-interface-xml-text
            .......
          returning
            a-converter-return-code           .
    End Program 'ACTDCNVD'.
```

Figure 15-14 Input COBOL and XML driver and XSD generated

The XML schema is automatically generated and contains the description of the names and types of XML elements. These elements can be present in an XML document that our program will be able to process and generate. For more information on XML schema, visit:

http://www.w3.org/XML/Schema

Modifying the converter driver programs

The XML converter driver (`ACTDDRV.cbl`) is a COBOL program that shows the invocation sequence for the inbound converter, the existing program, and the outbound converter.

You can now modify this converter driver template and call the existing application using the EXEC CICS language. The driver template also provides error-handling mechanisms that can be modified to suit your needs.

Changes to the data division

Because we want to call the driver program as a CICS transaction, we make the XML data structure the COMMAREA and we change the legacy communication area into a business structure that is passed to the legacy program as `COMMAREA` in the call.

Figure 15-15 shows the changes to the `process` statement and the `Data Division`.

```
 Process opt,lib,codepage(01140),cics
* XML Converter Driver Program *
 Identification Division.
   Program-Id. 'ACTDCNVD'.
   ....
   Data Division.
   ....
* ** Legacy Application Inbound / Outbound Binary Interface **
* ************************************************************
 01  ~~DFHCOMMAREA~~ BUSINESS-DATASTRUCT .
 05  CUSTNO     PIC S99999 .
 05  ACCTNO     PIC S99999 .
 05  BALANCE    PIC S9999V99 .
  Linkage Section.
* ******************************************
* ** New XML Inbound / Outbound Interface **
* ******************************************
   1 ~~a-xml-interface~~ DFHCOMMAREA .
   2 a-interface-xml-text-len   pic 9(9) binary.
   2 a-interface-xml-text       pic x(1024000).
```

Figure 15-15 Data division changes

Changes to the procedure division

Figure 15-16 shows the modified `Procedure Division` where the legacy program is invoked. Also the `goback` statements must be changed to `exec cics return`.

```
        Procedure Division using ~~a-xml-interface~~ DFHCOMMAREA.
         Mainline Section.
         ,....
     * Send Failure Response Message If Error Was Detected
         if a-exception-occurred = 'Y'
             move a-message-buffer to a-error-description
...
             move a-failure-response
               to a-interface-xml-text(1:a-interface-xml-text-len)
             perform a-unregister-exception-handler
             ~~goback~~
             exec cis return
             end-exec
           end-if
      * + -------------------------- +
      * | Execute Legacy Application |
      * + -------------------------- +
      * . EXEC CICS LINK
      * .   PROGRAM('LEGACY')
      * .   COMMAREA(DFHCOMMAREA)
      * .   ...
      * . END-EXEC ...OR
      * .
      * . call 'LEGACY' using DFHCOMMAREA
          exec cics link
            program(‘DFHOACTD’)
            commarea(BUSINESS-DATASTRUCT)
          end-exec
      * + -------------------------------- +
      * | Execute Outbound XML Transformer |
      * + -------------------------------- +
           perform a-outbound-conversion
      * + ---------------------------- +
      * | Unregister Exception Handler |
      * + ---------------------------- +
           perform a-unregister-exception-handler
      * + -------- +
      * | Finished |
      * + -------- +
           ~~goback~~
             exec cis return
             end-exec
```

Figure 15-16 Procedure division changes

Changes to converter calls

Figure 15-17 shows the changes in the converter calls where the original communication area (now the business structure) must be passed.

```
  a-inbound-conversion.
     call 'ACTDCNVI'
       using
         ~~DFHCOMMAREA~~ BUSINESS-DATASTRUCT
         a-interface-xml-text-len
         a-interface-xml-text
         omitted
*        a-optional-feedback-code
       returning
         a-converter-return-code
     .
  a-outbound-conversion.
     call 'ACTDCNVO'
       using
         ~~DFHCOMMAREA~~ BUSINESS-DATASTRUCT
         a-interface-xml-text-len
         a-interface-xml-text
         omitted
*        a-optional-feedback-code
       returning
         a-converter-return-code
```

Figure 15-17 Converter invocation changes

Exception handling routine

The generated driver file also contains a program called `ACTDCNVH` that does the exception handling. The only change required in this program is the additional `process` statement (Figure 15-18).

```
*  *************************************************************
*                       Exception Handler
*  *************************************************************
 Process nocics
 Identification Division.
   Program-Id. 'ACTDCNVH'.
   Author. GENERATED.
   Date-Written. 10/8/02 5:03 PM
  Data Division.
  .....
```

Figure 15-18 Exception handling changes

Changes to CSTDRV.cbl

Repeat all the modifications for the `CSTDDRVD.cbl` program.

Install the programs in a CISC environment

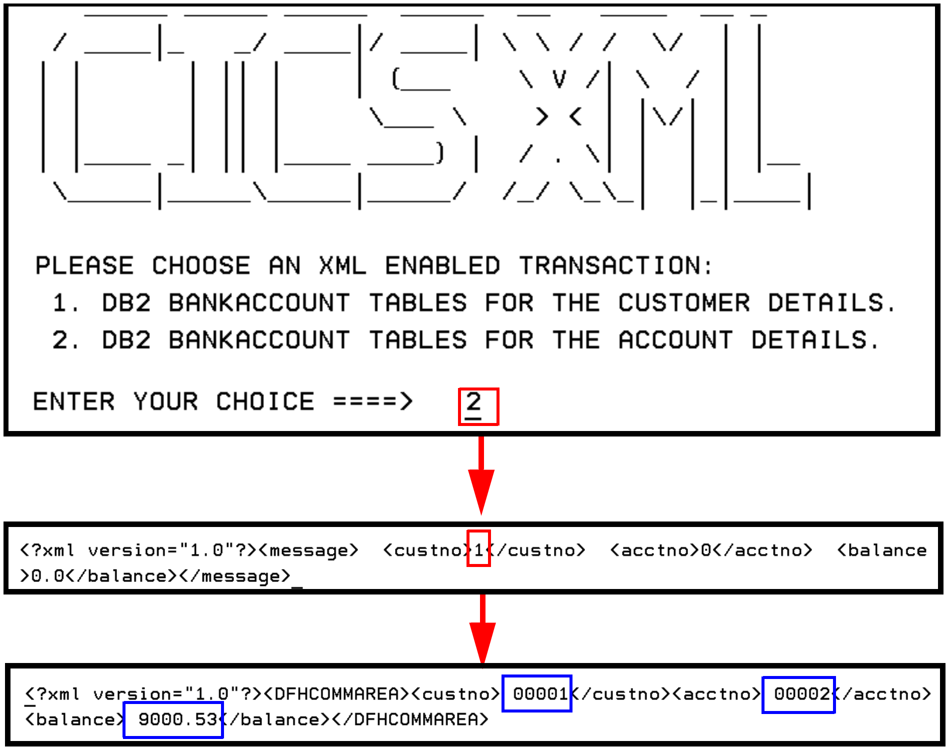
For the installation of those programs in the CICS environment, refer to the instructions of the paper *XML For the Enterprise* - *Providing An XML Interface To A CICS Application* that can be found in the root of the CD where you get the samples or at the site:

http://www-3.ibm.com/software/ad/studioenterprisedev/library/

Running the XML enabled application

In this section we run the XML enabled application. We assume that the XML front-end program (`XMLFRNT`) has been installed as a CICS transaction.

To start the application, bring up a CICS terminal and run transaction `XMLF`. A sequence of screens from a sample run is shown in Figure 15-19.



```
PLEASE CHOOSE AN XML ENABLED TRANSACTION:
 1. DB2 BANKACCOUNT TABLES FOR THE CUSTOMER DETAILS.
 2. DB2 BANKACCOUNT TABLES FOR THE ACCOUNT DETAILS.

ENTER YOUR CHOICE ====>   2
```

```
<?xml version="1.0"?><message>  <custno>1</custno>  <acctno>0</acctno>  <balance
>0.0</balance></message>_
```

```
<?xml version="1.0"?><DFHCOMMAREA><custno> 00001</custno><acctno> 00002</acctno>
<balance> 9000.53</balance></DFHCOMMAREA>
```

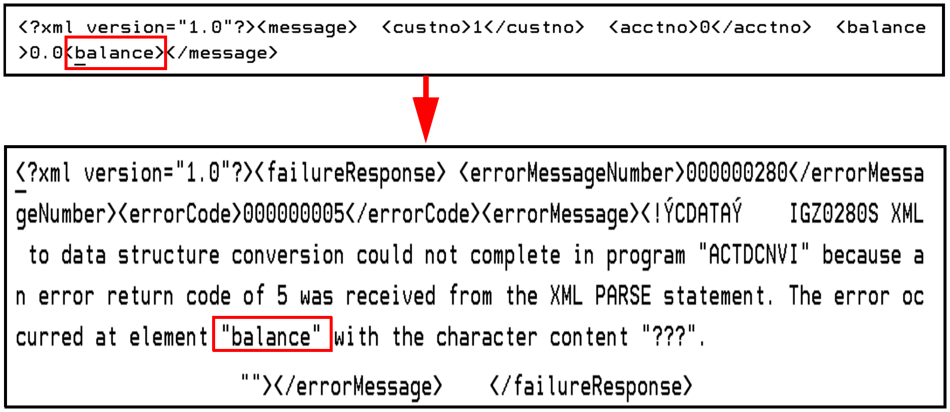
Figure 15-19 Testing the XML generated programs

- Enter 2 to run the account transaction.
- The XML input record is displayed. You can overtype the customer number and press *Enter*.
- The request is executed and the result XML is displayed, showing the account number and balance.

Errors messages parsing input XML data

The XML converters are able to report errors. You can verify that by performing tests with an invalid XML input structure or invalid input data.

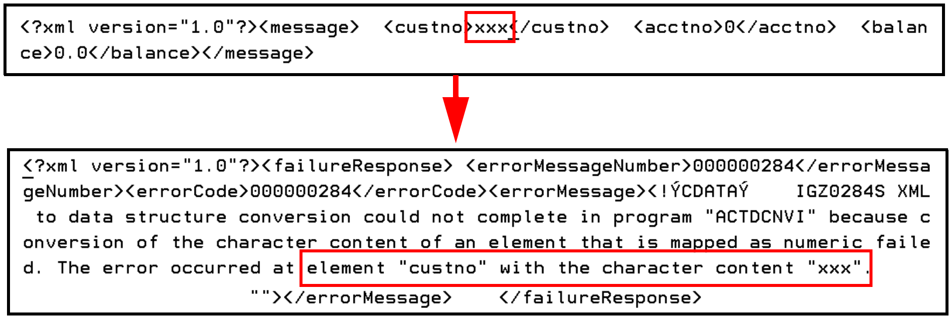
Figure 15-20 shows a run with an invalid XML structure where the `</balance>` end tag was modified, and Figure 15-21 shows a run where a non-numeric customer number was entered.



```
<?xml version="1.0"?><message>  <custno>1</custno>  <acctno>0</acctno>  <balance
>0.0<balance></message>
```

```
<?xml version="1.0"?><failureResponse> <errorMessageNumber>000000280</errorMessa
geNumber><errorCode>000000005</errorCode><errorMessage><!ÝCDATAÝ    IGZ0280S XML
 to data structure conversion could not complete in program "ACTDCNVI" because a
n error return code of 5 was received from the XML PARSE statement. The error oc
curred at element "balance" with the character content "???".
                   "")</errorMessage>    </failureResponse>
```

Figure 15-20   XML converter error message for invalid XML structure



```
<?xml version="1.0"?><message>  <custno>xxx</custno>  <acctno>0</acctno>  <balan
ce>0.0</balance></message>
```

```
<?xml version="1.0"?><failureResponse> <errorMessageNumber>000000284</errorMessa
geNumber><errorCode>000000284</errorCode><errorMessage><!ÝCDATAÝ    IGZ0284S XML
 to data structure conversion could not complete in program "ACTDCNVI" because c
onversion of the character content of an element that is mapped as numeric faile
d. The error occurred at element "custno" with the character content "xxx".
                    "")</errorMessage>    </failureResponse>
```

Figure 15-21   XML converter error message for invalid data

Modifying the XML converter interface

Both the inbound and outbound XML converters are invoked using a call statement. Arguments to the converters are a mixture of input and output parameters whose contents may be changed upon return from invocation. The call signature of the converters is displayed in Figure 15-22.

```
CALL ‘CONV’ USING
                  DATA-STRUCTURE (input)
                  XML-MESSAGE-LEN (input + output)
                  XML-MESSAGE-TEXT (input + output)
                  (FEEDBACK-CODE or OMITTED) (output)
            RETURNING
                  CONVERTER-RETURN-CODE (output)
```

Figure 15-22 Converter call interface

This COBOL code is an example of a call to a converter. Input and output properties for each argument are displayed in parenthesized italics. Because the structure of each argument is unique, it must not vary from the description here.

DATA-STRUCTURE is a piece of storage whose structure is identical to that of the data structure that was nominated as the inbound data structure when the converter was generated. During an inbound conversion, DATA-STRUCTURE will be populated with values from the input XML document provided in the arguments XML-MESSAGE-LEN and XML-MESSAGE-TEXT. In the case of an outbound conversion, DATA-STRUCTURE is used to populate an XML message, whose properties are placed in the XML-MESSAGE-LEN and XML-MESSAGE-TEXT arguments.

FEEDBACK-CODE is a 12-byte language environment condition token that can be omitted by using the OMITTED keyword on the call. Choosing to omit this argument will cause any error encountered by the converter to be signaled as a severe condition containing information about the error.

On the other hand not omitting this argument will cause the converter to simply place a condition token representing the error into FEEDBACK-CODE without signaling a condition. The structure of FEEDBACK-CODE is displayed in Figure 15-23.

More detailed information about the structure and use of this condition token can be found in the *Language Environment Programming Guide*.

```
1 FEEDBACK-CODE.
    2 CONDITION-TOKEN-VALUE.
     COPY CEEIGZCT.
       3 CASE-1-CONDITION-ID.
         4 SEVERITY PIC S9(4) BINARY.
         4 MSG-NO PIC S9(4) BINARY.
       3 CASE-2-CONDITION-ID
       REDEFINES CASE-1-CONDITION-ID.
         4 CLASS-CODE PIC S9(4) BINARY.
         4 CAUSE-CODE PIC S9(4) BINARY.
       3 CASE-SEV-CTL PIC X.
       3 FACILITY-ID PIC XXX.
     2 I-S-INFO PIC S9(9) BINARY.
```

Figure 15-23 Cobol program structure of FeedBack-Code

`CONVERTER-RETURN-CODE` is an output only argument, which will contain one of two classes of return codes upon completion of the call:

- If the converter encounters an error within its own facilities, that is, not an error from the XML `parse` statement, then the language environment message ID associated with the error will be placed in the argument.
- The second class of return codes is the codes returned from the XML `parse` statement. These will occur in the case where something was syntactically incorrect in the input XML document. Note that this second class of errors only occurs during an inbound conversion.

Summary

Although this feature can be greatly improved in the future, it is a good starting point for modernizing existing COBOL applications to support XML messages.

Developers can achieve significant productivity gains by utilizing the converter and driver template generators.

The application will benefit in regard to performance by running XML parsing and conversions on the z/OS systems.

The total cost of ownership (TCO) can be reduced by having one development environment.

Part 5

Appendixes

A

Team development

In this appendix we describe software configuration management support in Enterprise Developer Version 5.

The description provided here is in no way complete. We only scratch the surface of the team support capability of Enterprise Developer. Also we describe only how to work with Concurrent Versions System (CVS) and do not cover the support for ClearCase LT.

Team environment

For a good description of the team environment of the Application Developer Version 4, refer to Part 7 of the *WebSphere Studio Application Developer Programming Guide*, SG24-6585.

The concepts of team programming described in that redbook apply also to the Enterprise Developer, although there have been many changes in Version 5 of the products.

Concurrent Versions System

Concurrent Versions System (CVS) is an open-source network-transparent version control system. CVS is useful for everyone from individual developers to large, distributed teams.

CVS is not distributed with Enterprise Developer, but support for CVS is integrated in Enterprise Developer. CVS provides a team programming environment where team members do all of their work in their own Workbench, isolated from each other, and eventually share their work through a CVS repository.

CVS also can be used by a developer in stand-alone mode to keep versions of the code.

For general information about CVS and downloadable code for UNIX, go to:

`http://www.cvshome.org/`

For downloadable code for Windows, go to:

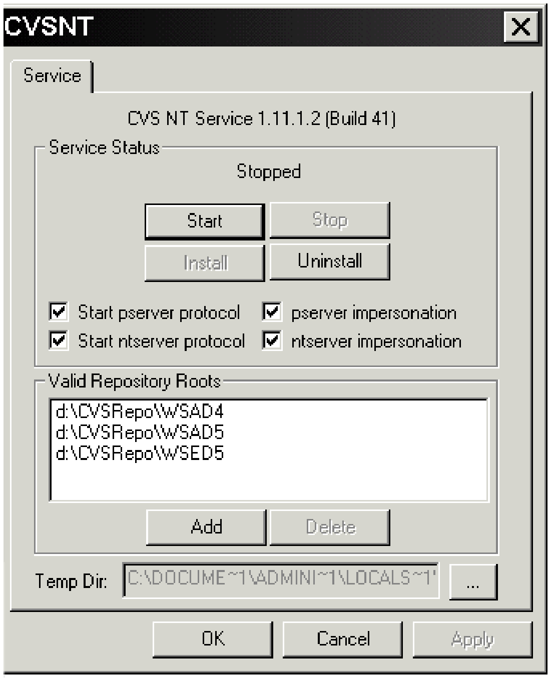
`http://www.cvsnt.org/`

CVS installation and configuration

We installed CVS for Window NT Version 1.11.1.2 for this redbook project.

After installation, the CVS server must be configured. Select P*rograms -> CVS for NT -> Configure server*. In the CVSNT dialog, select the protocols that you want to support, and add the repository locations. You can set up multiple repositories for different development efforts. Start the server when done.

Figure A-1 shows a sample CVSNT configuration dialog.



CVSNT

Service

CVS NT Service 1.11.1.2 (Build 41)

Service Status: Stopped

☑ Start pserver protocol ☑ pserver impersonation

☑ Start ntserver protocol ☑ ntserver impersonation

Valid Repository Roots

```
d:\CVSRepo\WSAD4
d:\CVSRepo\WSAD5
d:\CVSRepo\WSED5
```

Temp Dir: C:\DOCUME~1\ADMINI~1\LOCALS~1'

Figure A-1 CVS for NT configuration

What is new in Version 5?

The most important improvements in team support in Application Developer and Enterprise Developer Version 5 are:

- CVS Repository Exploring perspective to browse the content of repositories
- CVS Console view, shows messages returned from the CVS server
- Consistent CVS terms used throughout, for example, branch instead of stream
- File compression options for transferring files to the CVS repository
- New resources must be explicitly added to CVS control
- Text/binary support by identifying what file types are text or binary
- Synchronize outgoing changes optimizations (only outgoing changes are synchronized, which reduces network traffic)
- CVS decorators are visual indicators next to resources

This list does not show all the new features, but points to the major differences compared to previous versions of WebSphere Studio products.

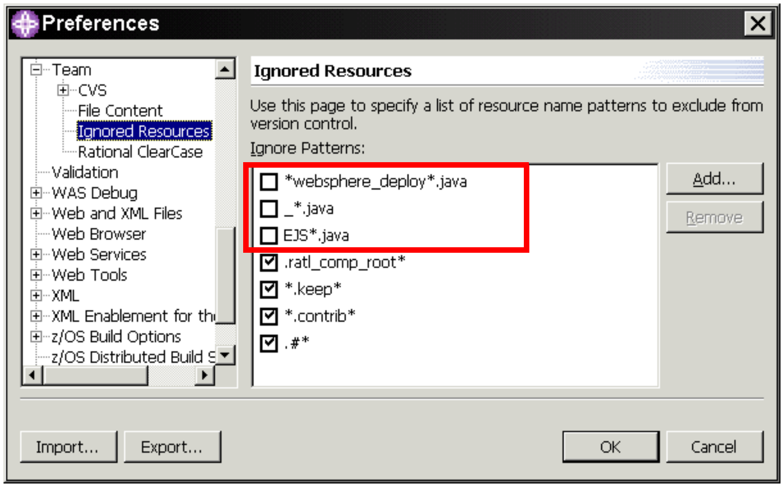
What changes could impact your work?

In Version 4, after adding an EJB project to CVS, you can delete the project from the workspace and when necessary import it again and all the components are reloaded.

In Version 5, this scenario will lead to many errors, such as missing classes, and you have to redeploy the EJBs. The reason is that some of the components, such as the deployed code, are not stored in the CVS repository in the default setup.

To store the deployed code in the CVS repository, you have to change the preferences of the Workbench:

- Select *Window* -> *Preferences* -> *Team* -> *Ignored Resources* (Figure A-2).
- Remove the first three check marks to store EJB deployed code in CVS.
- Click *OK*.



Ignored Resources

Use this page to specify a list of resource name patterns to exclude from version control.

Ignore Patterns:

☐ *websphere_deploy*.java
☐ _*.java
☐ EJS*.java
☑ .ratl_comp_root*
☑ *.keep*
☑ *.contrib*
☑ .#*

Figure A-2 CVS ignored resources preferences

More details on ignored resources preference

On the Ignored Resources page, you can specify file name patterns to be excluded from the version control management system.

Files are matched against the list of patterns, before they are considered as version control candidates. A file or directory that matches any one of the patterns will be ignored during update or commit operations. The patterns may contain the wildcard characters * (any sequence of zero or more characters) and ? (any one character).

To add a file type to the ignore list, click the *Add* button. In the dialog, enter a file type (for example, *.class). To remove a file type from the ignore list, select the file type in the ignore list and click *Remove*. You can temporarily disable ignoring the file pattern by de-selecting it from the list; you do not have to remove the specified file pattern from the list.

Ignoring resources from version control

When synchronizing resources, you may not want to commit all resources to the repository. There are two ignore facilities provided, allowing the user to specify which resources should be excluded from update and commit operations:

- The first is a global ignore facility, provided by the Workbench as shown Figure A-2.
- The second is the CVS ignore facility, which reads the contents of a special `.cvsignore` file to determine what to ignore.

CVS ignore facility

The Eclipse CVS client recognizes a file named `.cvsignore` in each directory of a project. This is a standard CVS facility and many existing CVS projects may contain such a file.

This text file consists of a list of files, directories, or patterns. In a similar way to the global ignore facility, the wildcard * and ? may be present in any entry in the `.cvsignore` file. Any file or subdirectory in the current directory that matches any one of the patterns is ignored.

It is important to note that the semantics of this file differs from that of the global ignore facility in that it applies only to files and directories in the same directory as the `.cvsignore` file itself. A project may contain one `.cvsignore` file in each directory. For more information, visit `http://www.cvshome.org`.

Resources that have not been added to CVS control can be ignored by selecting *Team > Add to .cvsignore* from the context menu of the resource in the Navigator view. This menu option is also available in the Synchronize view.

Development scenario for a single user

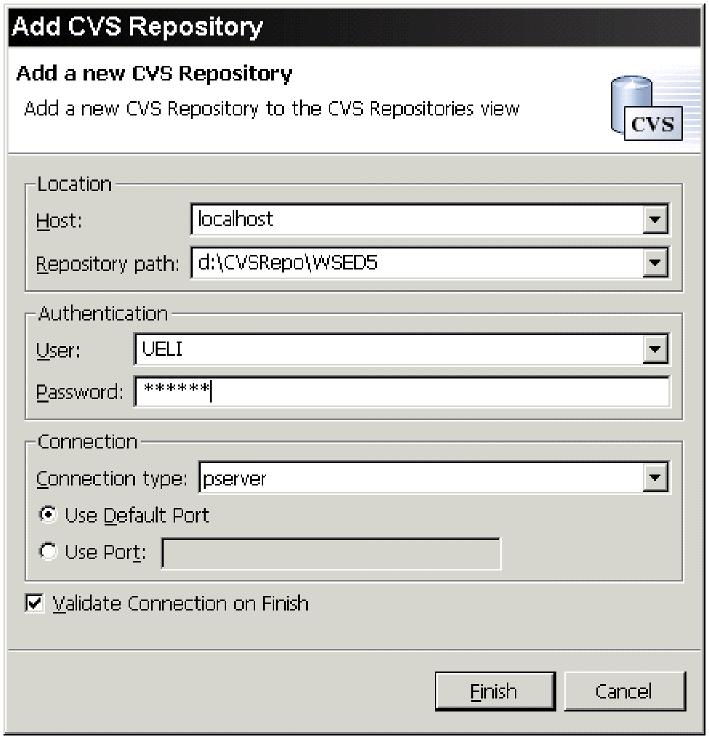
In this section, we provide a short scenario using the CVS support in Enterprise Developer. This scenario includes:

- Connecting to a CVS repository
- Adding a project to CVS
- Version a project
- Changing files and synchronizing a project

Connecting to a CVS repository

To connect the Workbench to a CVS repository:

- Open the CVS Repository Exploring perspective.
- Select *New -> Repository Location* from the context menu of the CMS Repositories view.
- Complete the dialog as shown in Figure A-3.



Add CVS Repository

Add a new CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host: localhost

Repository path: d:\CVSRepo\WSED5

Authentication

User: UELI

Password: ******

Connection

Connection type: pserver

● Use Default Port

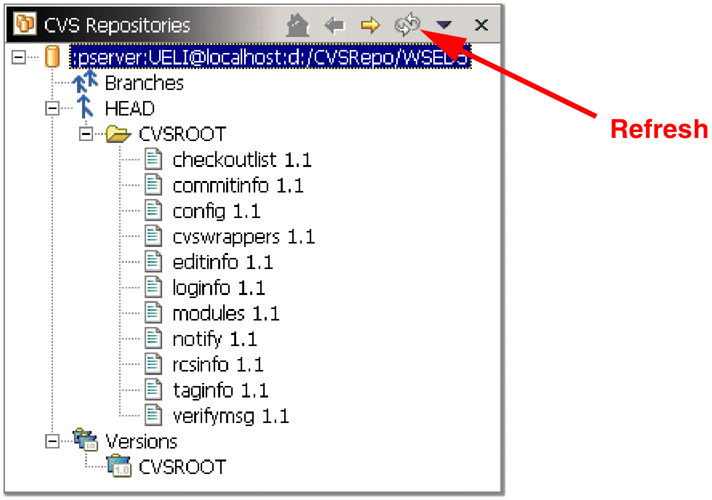
○ Use Port:

☑ Validate Connection on Finish

Figure A-3 New repository location

You must know the repository path on the target machine. The *pserver* connection type validates your user ID and password with the Windows system.

- The repository location is added to the CVS Repositories view (Figure A-4).



CVS Repositories

```
:pserver:UELI@localhost:d:/CVSRepo/WSED5
    Branches
    HEAD
        CVSROOT
            checkoutlist 1.1
            commitinfo 1.1
            config 1.1
            cvswrappers 1.1
            editinfo 1.1
            loginfo 1.1
            modules 1.1
            notify 1.1
            rcsinfo 1.1
            taginfo 1.1
            verifymsg 1.1
    Versions
        CVSROOT
```

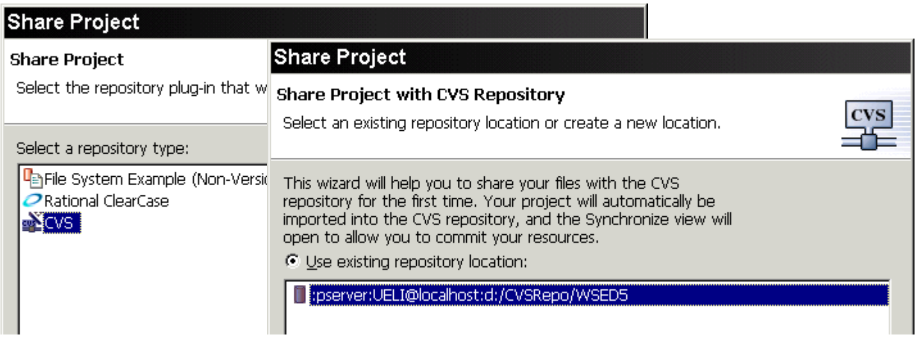
Refresh

Figure A-4 ICVS Repositories view after connecting

Adding a project to CVS control

The next step is to add projects to CVS control:

- Open the Web perspective, select the `ItsoMyTradeSade` project and *Team -> Share Project* (context).
- Select *CVS* in the dialog, click *Next*, select the repository, click *Next* to go through the rest of the dialog, and click *Finish* (Figure A-5).



Share Project

Share Project

Select the repository plug-in that w

Select a repository type:

File System Example (Non-Versi
Rational ClearCase
CVS

Share Project

Share Project with CVS Repository

Select an existing repository location or create a new location.

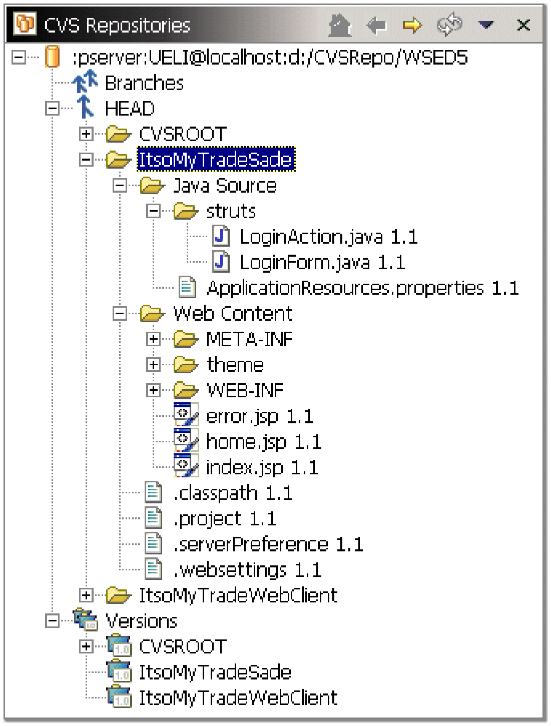
This wizard will help you to share your files with the CVS repository for the first time. Your project will automatically be imported into the CVS repository, and the Synchronize view will open to allow you to commit your resources.

● Use existing repository location:

:pserver:UELI@localhost:d:/CVSRepo/WSED5

Figure A-5 Share a project

- Refresh the CVS Repositories view to see the project name under HEAD and Versions (use the *Refresh* icon or the *Refresh View* context menu).
- Notice that no folders or files are visible; we have to add them to version control explicitly.
- In the Web perspective, select the folders of the `ItsoMyTradeSade` project you want to have under version control, for example, `Java Source` and `Web Content`, and *Team -> Add to Version Control* (context). Also select the project control files (`.project`, `.classpath`, ...) and *Team -> Add to Version Control* in a Navigator view.
- Now you can commit individual files, folders, or the whole project. Initially the easiest is to select the `ItsoMyTradeSade` project and *Team -> Commit*. You are prompted for a comment (enter `Initial`).
- Figure A-6 shows the CVS Repositories view after adding two projects to version control and committing the files. Notice that all files carry 1.1 as the revision number.



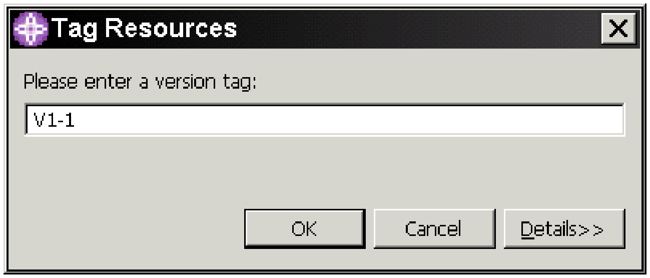
```
:pserver:UELI@localhost:d:/CVSRepo/WSED5
    Branches
    HEAD
        CVSROOT
        ItsoMyTradeSade
            Java Source
                struts
                    LoginAction.java 1.1
                    LoginForm.java 1.1
                ApplicationResources.properties 1.1
            Web Content
                META-INF
                theme
                WEB-INF
                error.jsp 1.1
                home.jsp 1.1
                index.jsp 1.1
            .classpath 1.1
            .project 1.1
            .serverPreference 1.1
            .websettings 1.1
        ItsoMyTradeWebClient
    Versions
        CVSROOT
        ItsoMyTradeSade
        ItsoMyTradeWebClient
```

Figure A-6 Projects added to CVS

Create a version

Now we can freeze the initial code:

- ► In the Web perspective, select the `ItsoMyTradeSade` project and *Team -> Tag as Version*.
- ► You are prompted for a tag (Figure A-7); enter `V1-1`, for example (periods are not allowed).

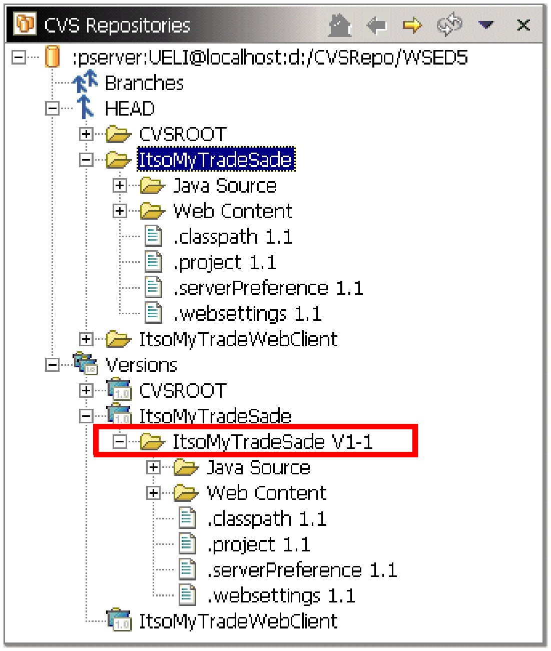

Tag Resources

Please enter a version tag:

V1-1

Figure A-7 Version tag

- ► Refresh the CVS Repositories view and the project version is visible (Figure A-8).


CVS Repositories

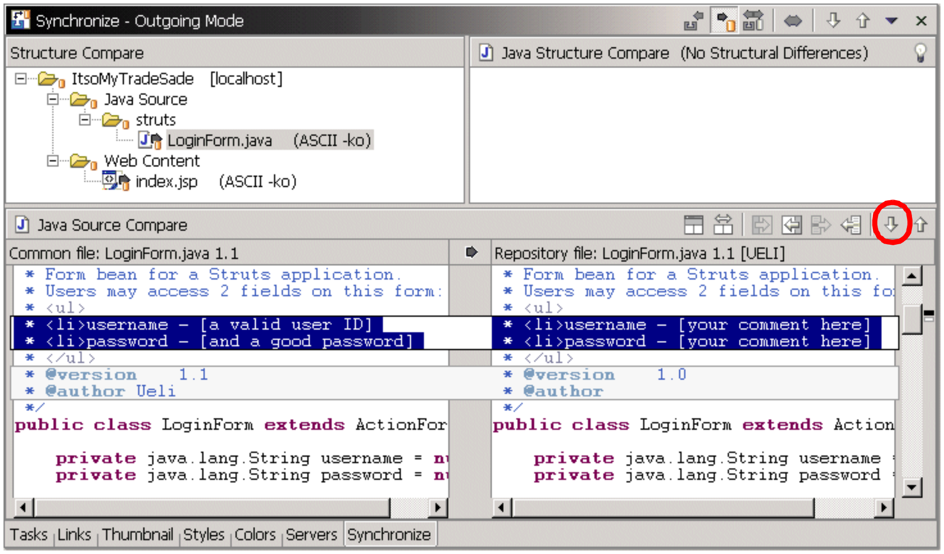
```
:pserver:UELI@localhost:d:/CVSRepo/WSED5
    Branches
    HEAD
        CVSROOT
        ItsoMyTradeSade
            Java Source
            Web Content
            .classpath 1.1
            .project 1.1
            .serverPreference 1.1
            .websettings 1.1
        ItsoMyTradeWebClient
    Versions
        CVSROOT
        ItsoMyTradeSade
            ItsoMyTradeSade V1-1
                Java Source
                Web Content
                .classpath 1.1
                .project 1.1
                .serverPreference 1.1
                .websettings 1.1
        ItsoMyTradeWebClient
```

Figure A-8 Project version

Making changes and synchronizing

The next step is to show how changes to the code are handled with CVS:

- In the `ItsoMyTradeSade` project change two files, for example:
  - Change the comments in the `LoginForm.java` source.
  - Add a line (`<h3>xxxxx</h3>`) after the form in the `index.jsp` file.
- After saving the files, select both files and *Team -> Synchronize with Repository* (context). The files are compared with the branch in the repository and the Synchronize view opens. Use the down arrow icon to display the first change (Figure A-9).


Synchronize - Outgoing Mode
Structure Compare
ItsoMyTradeSade [localhost]
Java Source
struts
LoginForm.java (ASCII -ko)
Web Content
index.jsp (ASCII -ko)
Java Structure Compare (No Structural Differences)
Java Source Compare
Common file: LoginForm.java 1.1
Repository file: LoginForm.java 1.1 [UELI]
Tasks
Links
Thumbnail
Styles
Colors
Servers
Synchronize

```
 * Form bean for a Struts application.
 * Users may access 2 fields on this form:
 * <ul>
 * <li>username - [a valid user ID]
 * <li>password - [and a good password]
 * </ul>
 * @version    1.1
 * @author Ueli
 */
public class LoginForm extends ActionFor

    private java.lang.String username = nu
    private java.lang.String password = nu
```

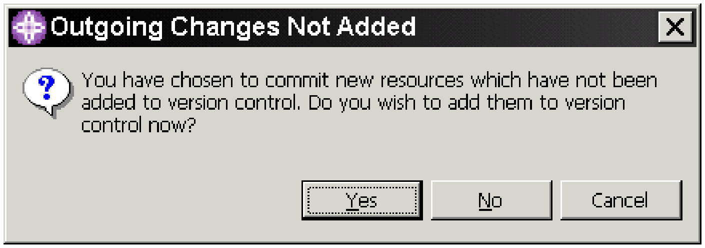
```
 * Form bean for a Struts application.
 * Users may access 2 fields on this fo
 * <ul>
 * <li>username - [your comment here]
 * <li>password - [your comment here]
 * </ul>
 * @version    1.0
 * @author
 */
public class LoginForm extends Action

    private java.lang.String username
    private java.lang.String password
```

Figure A-9 Synchronize view

- Move from change to change using the arrow icons and see the differences in the two files.
- Select the `ItsoMyTradeSade` project in the Structure Compare pane and *Commit* (context). This commits the changes to the repository. When prompted, enter `Change1` as comment.
- Refresh the CVS Repositories view and you can see that the two files carry 1.2 as the revision number.
- You could have committed the changes to the files by selecting *Team -> Commit* and the Synchronize view would have been bypassed.
- Select the project and *Tag as Version* and assign `V1-2` to the new version.

Synchronizing the project

When synchronizing a project, you may be prompted that the project includes files that have not been added to version control (Figure A-10). Click *Yes* to add the files to version control.

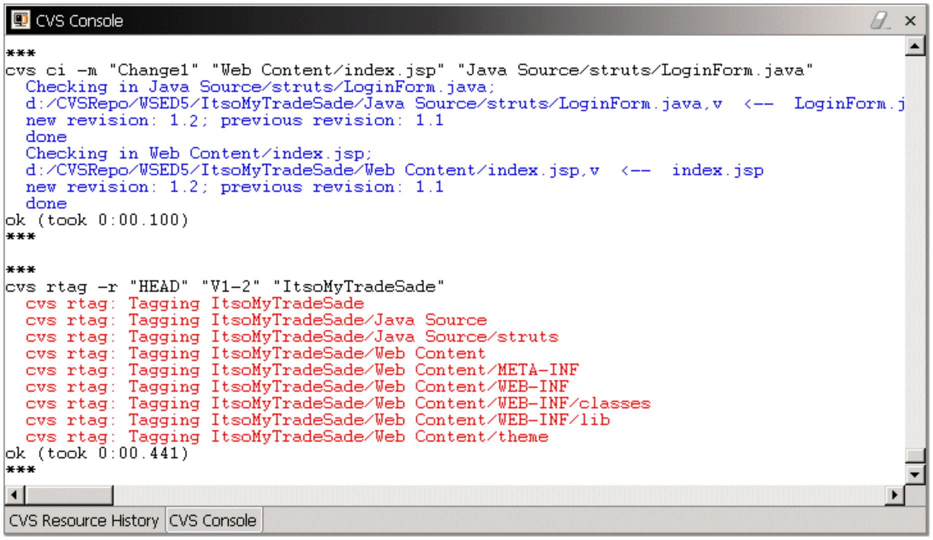


You have chosen to commit new resources which have not been added to version control. Do you wish to add them to version control now?

Figure A-10 Adding files to version control using synchronize

CVS console

In the CVS Console view, you can see all the interactions between Enterprise Developer and CVS. Select *Window -> Show View -> Other -> CVS -> CVS Console* to open the view (Figure A-11).

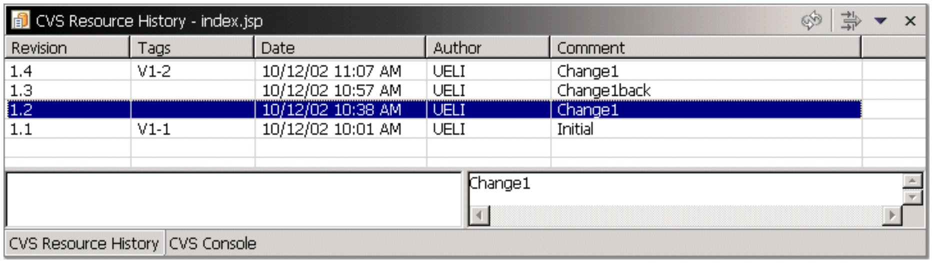


```
***
cvs ci -m "Change1" "Web Content/index.jsp" "Java Source/struts/LoginForm.java"
  Checking in Java Source/struts/LoginForm.java;
  d:/CVSRepo/WSED5/ItsoMyTradeSade/Java Source/struts/LoginForm.java,v  <--  LoginForm.j
  new revision: 1.2; previous revision: 1.1
  done
  Checking in Web Content/index.jsp;
  d:/CVSRepo/WSED5/ItsoMyTradeSade/Web Content/index.jsp,v  <--  index.jsp
  new revision: 1.2; previous revision: 1.1
  done
ok (took 0:00.100)
***

***
cvs rtag -r "HEAD" "V1-2" "ItsoMyTradeSade"
  cvs rtag: Tagging ItsoMyTradeSade
  cvs rtag: Tagging ItsoMyTradeSade/Java Source
  cvs rtag: Tagging ItsoMyTradeSade/Java Source/struts
  cvs rtag: Tagging ItsoMyTradeSade/Web Content
  cvs rtag: Tagging ItsoMyTradeSade/Web Content/META-INF
  cvs rtag: Tagging ItsoMyTradeSade/Web Content/WEB-INF
  cvs rtag: Tagging ItsoMyTradeSade/Web Content/WEB-INF/classes
  cvs rtag: Tagging ItsoMyTradeSade/Web Content/WEB-INF/lib
  cvs rtag: Tagging ItsoMyTradeSade/Web Content/theme
ok (took 0:00.441)
***
```

Figure A-11 CVS Console view

Resource history

The Resource History view shows all the changes that have been applied to a file. Select the `index.jsp` file in the CVS Repositories view and *Show in Resource History* (Figure A-12).



Revision	Tags	Date	Author	Comment
1.4	V1-2	10/12/02 11:07 AM	UELI	Change1
1.3		10/12/02 10:57 AM	UELI	Change1back
1.2		10/12/02 10:38 AM	UELI	Change1
1.1	V1-1	10/12/02 10:01 AM	UELI	Initial

Change1

Figure A-12 Resource history

File compare

There are a number of ways to compare two revisions of a file:

- Select two revisions in the Resource History view and *Compare* (context).
- Select a revision and a version in the CVS Repositories view and *Compare*.
- Select a file in any Navigator view and *Compare With*. In the dialog, select a repository version, repository revision, or a local history file for the compare operation (Figure A-13).

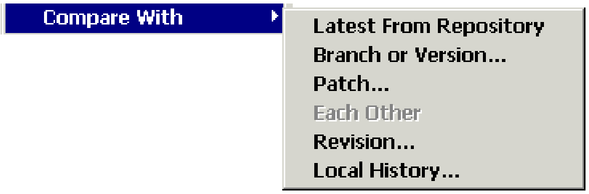
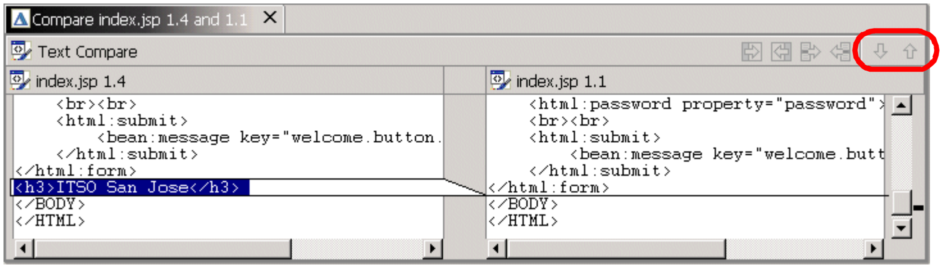


Figure A-13 Comparing files

- The file comparison opens and you can step from change to change using the arrow icons (Figure A-14).

Note: You can compare any two files with each other; we use revisions of one file as an example.



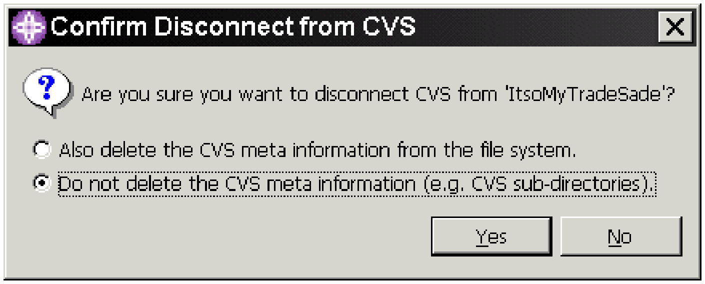
```
    <br><br>
    <html:submit>
        <bean:message key="welcome.button.
    </html:submit>
</html:form>
<h3>ITSO San Jose</h3>
</BODY>
</HTML>
```

```
    <html:password property="password" >
    <br><br>
    <html:submit>
        <bean:message key="welcome.butt
    </html:submit>
</html:form>
</BODY>
</HTML>
```

Figure A-14 File compare

Disconnecting a project

You can disconnect a project from the repository. Select the `ItsoMyTradeSade` project and *Team -> Disconnect*. You are prompted to confirm and also if you want to delete CVS control information (Figure A-15).



Are you sure you want to disconnect CVS from 'ItsoMyTradeSade'?

○ Also delete the CVS meta information from the file system.

● Do not delete the CVS meta information (e.g. CVS sub-directories).

Figure A-15 Disconnect confirmation

CVS adds special directories named CVS to the project and its folders. These directories can be deleted or kept on disconnect.

Reconnect

You can reconnect a project to the repository (*Team -> Share Project*). Reconnect is easier if the CVS folders are still in the project. If they were deleted, you are prompted to synchronize your code with the existing repository code.

Development scenario for a team

In a team environment, the CVS server is placed onto a server machine. All members of the team connect to the same CVS server. You can have multiple repositories managed by the same CVS server.

We will not explore real team operation in this redbook, and only outline a development scenario:

- All team members connect to the same repository.
- A team leader defines a new project.
- The project is added to CVS version control (*Team -> Share Project*).
- An initial version is created (*Team -> Tag as Version*).
- Team members load the project from CVS into their workspace. They select a project version in the CVS Repositories view and *Checkout as Project*.
- Team members add files and make changes to files. Periodically (frequently) they synchronize their work with the repository:
  - Commit own changes to the repository
  - Pick up changes from other team members
- If the same file has been changed by two team members, a conflict exists and must be resolved by merging the changes.
- Periodically, the team leader creates a new version of the project.
- For maintenance work and new development in parallel, a branch can be created. By default there is only one branch, called *HEAD*.

Where to be careful

The dangerous areas in such a team environment are files that are changed frequently, for example, deployment descriptors (`web.xml`, `ejb-jar.xml`):

- Every time a servlet is added, the deployment descriptor is updated.
- Every time an EJB is added, the deployment descriptor is updated.

The best way to handle this is to give such project updates to one team member, or to synchronize very frequently to minimize conflicts.

B

WebSphere Studio Asset Analyzer

In this appendix, we provide a description and illustrate some aspects of WebSphere Studio Asset Analyzer.

Asset Analyzer's prime functions are geared towards providing information on the current inventory of development artifacts on both mainframe and distributed environments. It provides developers with valuable insight on the structure of existing systems and relationships that exist between development artifacts. As such, it offers means to do impact analysis.

Furthermore, Asset Analyzer provides functionality in support of discovery and enablement of reusable code. The information is presented to the end user through a browser-based interface, thus allowing widespread usage of the information. This appendix can be seen as a enticement to encourage you to look at WebSphere Studio Asset Analyzer.

Inventory

In order for Asset Analyzer to provide information about your application, it has to gather metadata. Asset Analyzer is using a wealth of scanners and crawlers during the buildup of its inventory. Artifacts on the mainframe, which are held in partitioned data sets, can be scanned either from the mainframe through an ISPF dialog or from the workstations through the browser interface.

The scanning can be done on the basis of IBM SCLM or Serena Changeman projects, confining the scanning to the PDSs within a tools project. Rescanning can be triggered when, for instance, a system is moved into production.

After the initial scan, Asset Analyzer links the various artifacts found according to relationships defined in the information model.

On the distributed side, there is an Administrator GUI interface to initiate the crawling of artifacts held in the file system, ClearCase, or any WebDav enabled SCM server.

Figure B-1 provides an overview of WebSphere Studio Asset Analyzer.

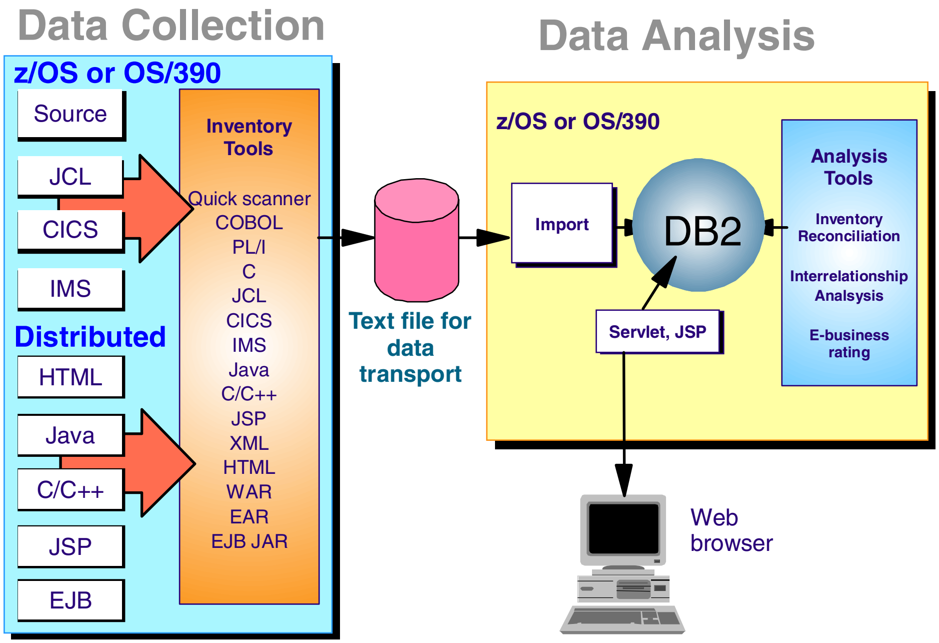


Figure B-1 Overview of WebSphere Studio Asset Analyzer

On the left-hand side of Figure B-1, you can get an impression of the artifacts supported. Note that several deployed artifacts are supported as well. All information is stored in a z/OS DB2 database. The functionality of Asset Analyzer can be exploited from a browser interface, which is served by servlets running on WebSphere. On the right side, you see three analysis functions mentioned.

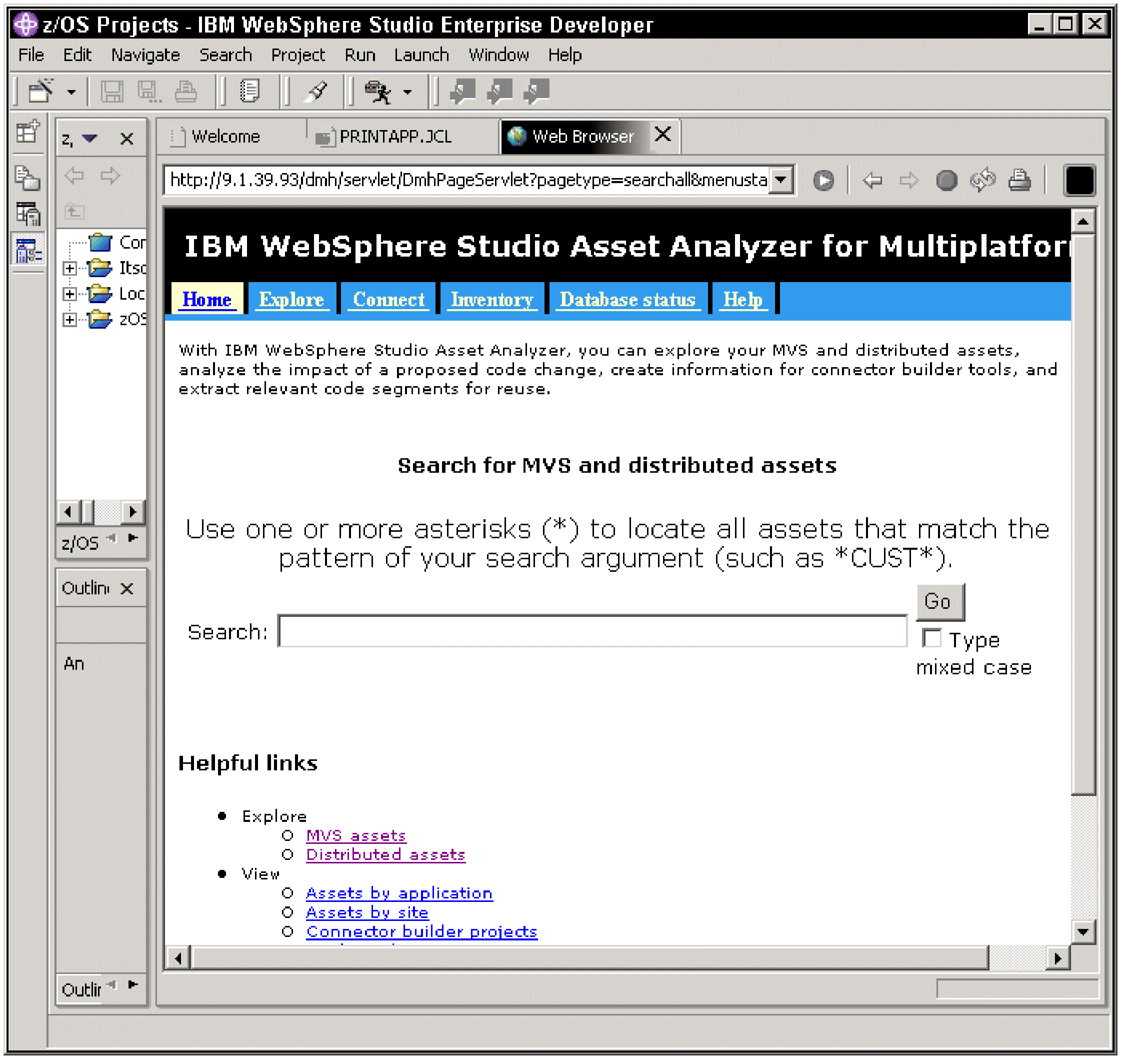
Impact analysis on the mainframe

To illustrate the impact analysis capabilities of Asset Analyzer for mainframe artifacts, we loaded the sample COBOL code used in Chapter 15, “XML enablement for COBOL” on page 401 into Asset Analyzer.

Note: Upon loading, Asset Analyzer indicated that some of the syntax was unknown, which resulted in flagging the analysis as not complete. The unknown syntax included statements, such as XML parse, which were only introduced in the latest level of COBOL. Our stand-alone version of Asset Analyzer uses a parser that does not recognize the latest syntax. In itself, it is a good thing that this was identified and put aside for future improvement.

There are several ways provided in the user interface of Asset Analyzer to get to the information. Let’s have a look at the home page of Asset Analyzer by selecting *Launch* -> *WSAA* -> *Impact Analysis* in the Enterprise Developer (Figure B-2).

Note: We created this chapter using a stand-alone version of Asset Analyzer. As a consequence, this is running on `localhost` instead of pointing to the URL on a mainframe. Some dialogs might differ slightly as well. For instance, the taking inventory dialogs will not have the mainframe options.


z/OS Projects - IBM WebSphere Studio Enterprise Developer
File Edit Navigate Search Project Run Launch Window Help
Welcome
PRINTAPP.JCL
Web Browser
http://9.1.39.93/dmh/servlet/DmhPageServlet?pagetype=searchall&menusta
Cor
Itsc
Loc
zOS
z/OS
Outlin
An
Outlir
Home
Explore
Connect
Inventory
Database status
Help
Go

IBM WebSphere Studio Asset Analyzer for Multiplatfor

With IBM WebSphere Studio Asset Analyzer, you can explore your MVS and distributed assets, analyze the impact of a proposed code change, create information for connector builder tools, and extract relevant code segments for reuse.

Search for MVS and distributed assets

Use one or more asterisks (*) to locate all assets that match the pattern of your search argument (such as *CUST*).

Search:

□ Type mixed case

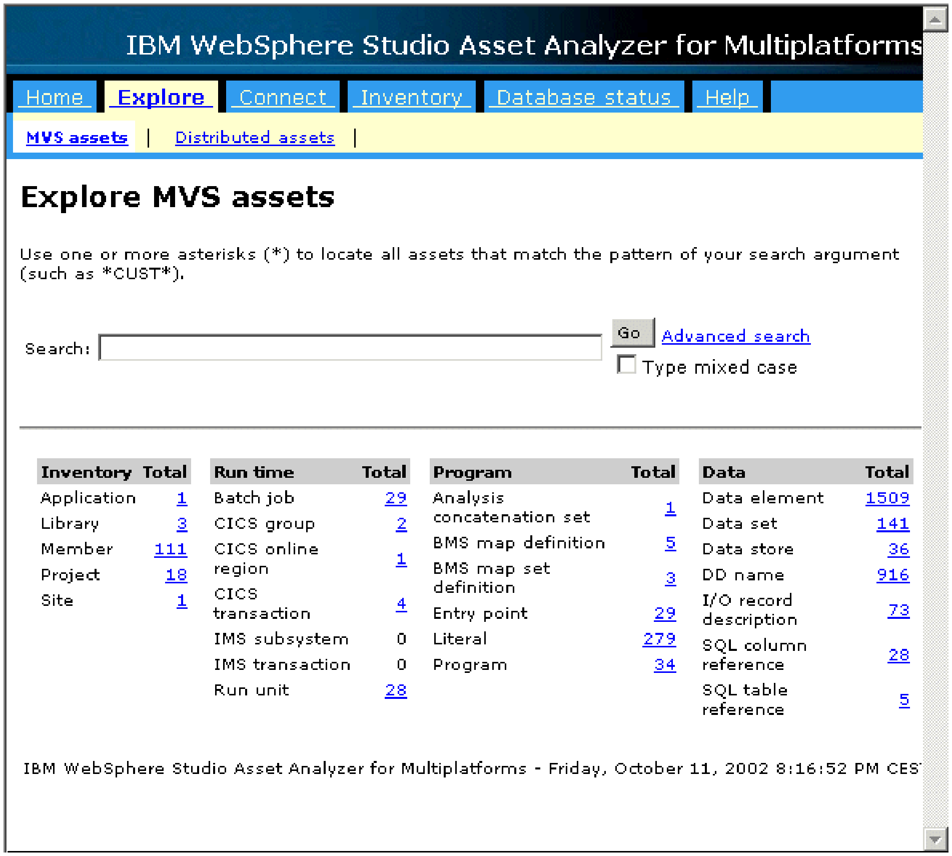
Helpful links

- Explore
  - MVS assets
  - Distributed assets
- View
  - Assets by application
  - Assets by site
  - Connector builder projects

Figure B-2 Asset Analyzer home in Enterprise Developer

In order to get to the information, we can either provide search criteria or click one of the links displayed at the bottom of the screen.

Let’s look at an overview of all MVS artifacts held in our database by selecting *MVS assets* (Figure B-3).



Explore MVS assets

Use one or more asterisks (*) to locate all assets that match the pattern of your search argument (such as *CUST*).

Search:

Inventory	Total	Run time	Total	Program	Total	Data	Total
Application	1	Batch job	29	Analysis concatenation set	1	Data element	1509
Library	3	CICS group	2	BMS map definition	5	Data set	141
Member	111	CICS online region	1	BMS map set definition	3	Data store	36
Project	18	CICS transaction	4	Entry point	29	DD name	916
Site	1	IMS subsystem	0	Literal	279	I/O record description	73
		IMS transaction	0	Program	34	SQL column reference	28
		Run unit	28			SQL table reference	5

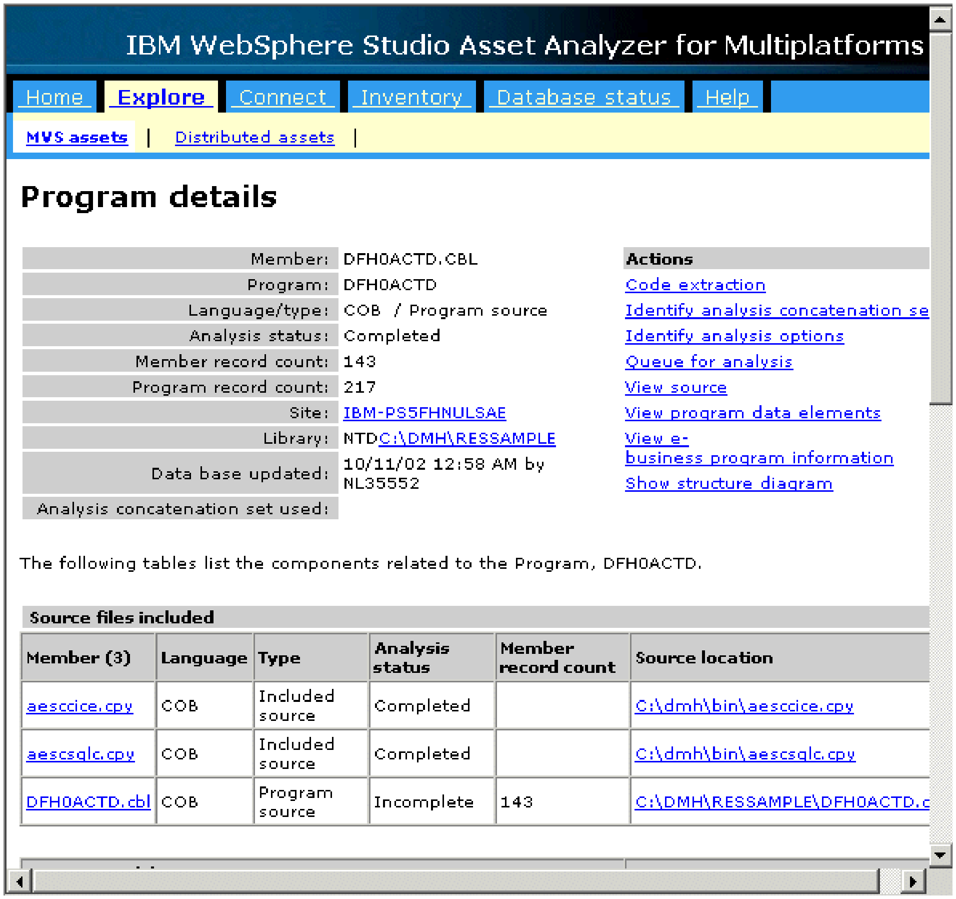
IBM WebSphere Studio Asset Analyzer for Multiplatforms - Friday, October 11, 2002 8:16:52 PM CES

Figure B-3 MVS assets overview

Grouped into four columns you can see:

- Inventory (we will only mention a few here):
  - Application—is a grouping you can define to hold a collection of artifacts
  - Project—activities in Asset Analyzer can be grouped into projects to hold all information gathered for a specific analysis or connector definition project
- Run time: these fields are quite clear
- Program: literals are strings
- Data: fields are clear

There now are multiple ways to get to the information for our DFHOACTD program. Either use search or click *Programs Total* and find it. Either way, you can get to the program detail screen (Figure B-4 and Figure B-5).



Program details

Member:	DFH0ACTD.CBL
Program:	DFH0ACTD
Language/type:	COB / Program source
Analysis status:	Completed
Member record count:	143
Program record count:	217
Site:	IBM-PS5FHNULSAE
Library:	NTDC:\DMH\RESSAMPLE
Data base updated:	10/11/02 12:58 AM by NL35552
Analysis concatenation set used:	

Actions

- Code extraction
- Identify analysis concatenation se
- Identify analysis options
- Queue for analysis
- View source
- View program data elements
- View e-business program information
- Show structure diagram

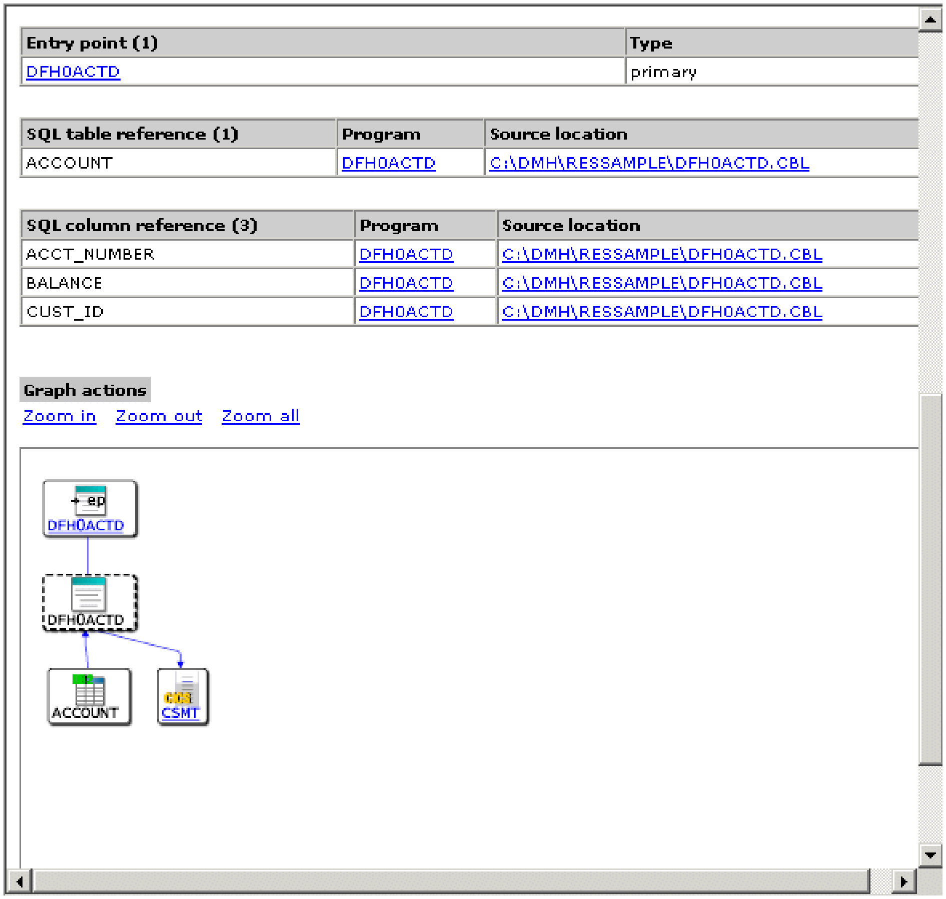
The following tables list the components related to the Program, DFH0ACTD.

Source files included

Member (3)	Language	Type	Analysis status	Member record count	Source location
aescice.cpy	COB	Included source	Completed		C:\dmh\bin\aescice.cpy
aescsqlc.cpy	COB	Included source	Completed		C:\dmh\bin\aescsqlc.cpy
DFH0ACTD.cbl	COB	Program source	Incomplete	143	C:\DMH\RESSAMPLE\DFH0ACTD.c

Figure B-4 Program details: top half

Here you see information on the program and various actions you can perform.



Entry point (1)	Type
DFH0ACTD	primary

SQL table reference (1)	Program	Source location
ACCOUNT	DFH0ACTD	C:\DMH\RESSAMPLE\DFH0ACTD.CBL

SQL column reference (3)	Program	Source location
ACCT_NUMBER	DFH0ACTD	C:\DMH\RESSAMPLE\DFH0ACTD.CBL
BALANCE	DFH0ACTD	C:\DMH\RESSAMPLE\DFH0ACTD.CBL
CUST_ID	DFH0ACTD	C:\DMH\RESSAMPLE\DFH0ACTD.CBL

Figure B-5   Program details: bottom half

Note that in the bottom part you get a graphical representation of the artifacts that are related to this program:

- Entry point (top)
- Account table, which is used by the program
- CICS table

Also note that you can click the artifacts in the picture and drill down to their information, as well as click all information in blue to drill down to the details behind.

If we click *Show structure diagram* from the action list at the top, we get a detailed program structure (Figure B-6).

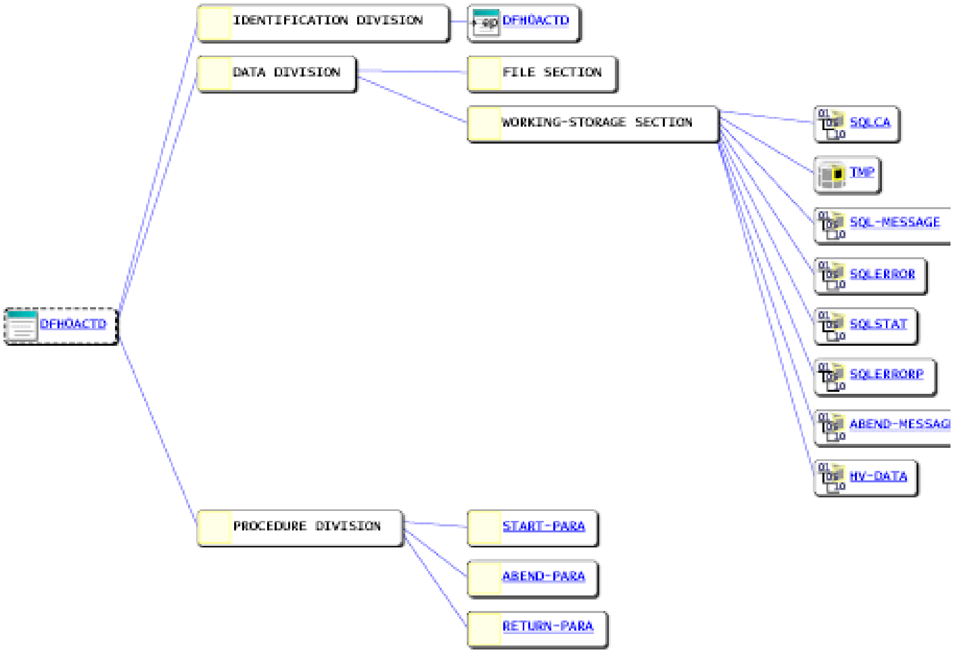
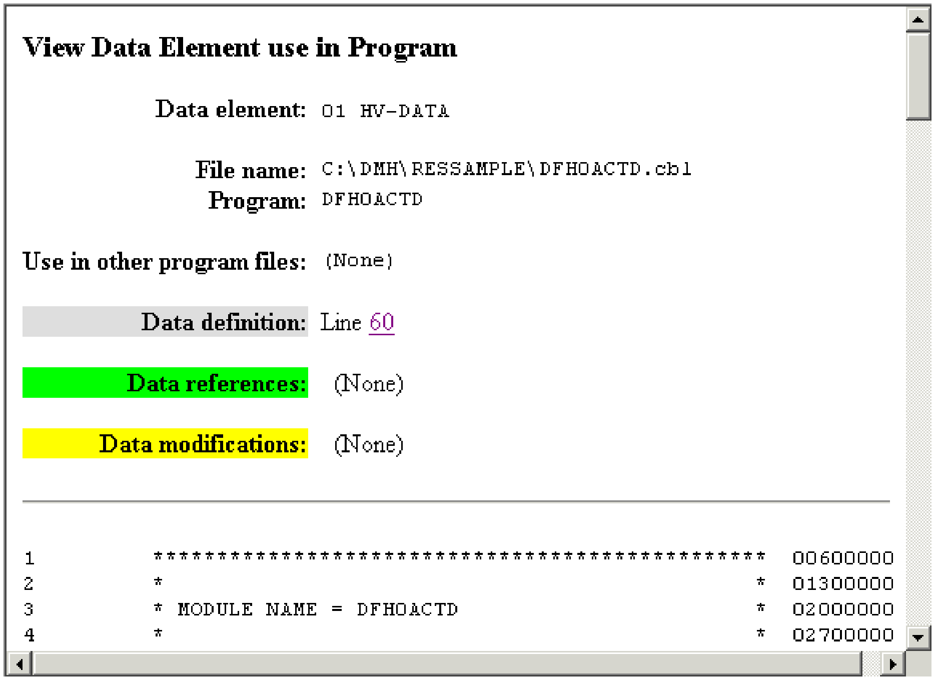


Figure B-6 Program structure diagram

Once more you can directly get to additional information by clicking an artifact. Note that there is a facility to zoom in and out as well. When we click *HV-DATA*, the details are displayed (Figure B-7).



View Data Element use in Program

Data element: 01 HV-DATA

File name: C:\DMH\RESSAMPLE\DFHOACTD.cbl
Program: DFHOACTD

Use in other program files: (None)

Data definition: Line 60

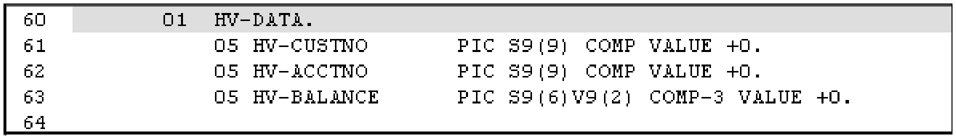
Data references: (None)

Data modifications: (None)

```
1         *************************************************  00600000
2         *                                               *  01300000
3         * MODULE NAME = DFHOACTD                        *  02000000
4         *                                               *  02700000
```

Figure B-7   Data element use

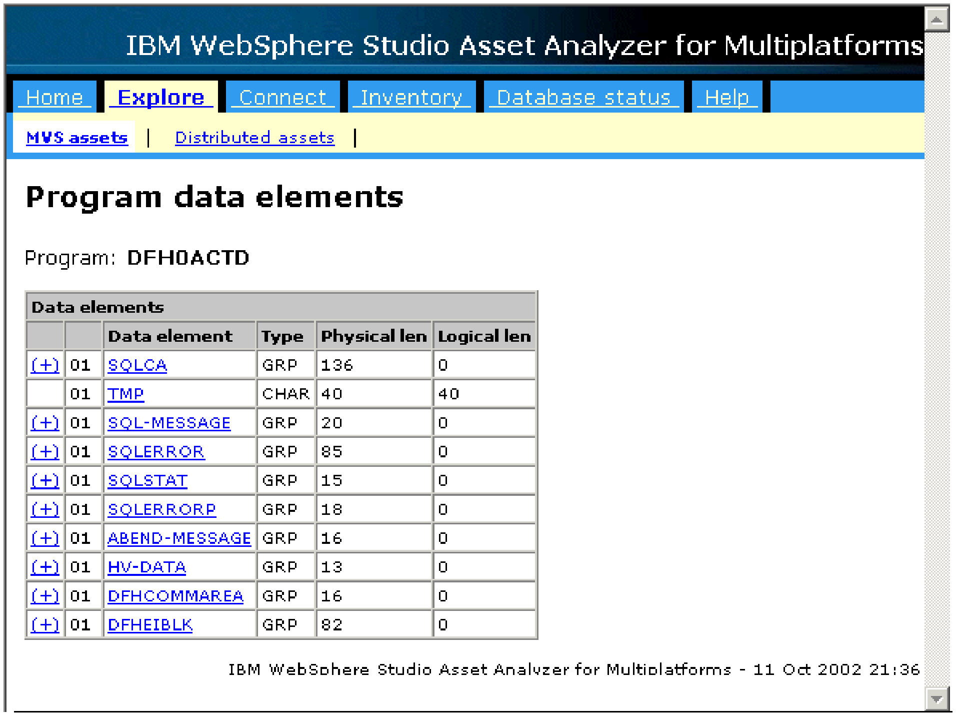
When you click *Data Definition*, the declaration statements in the source code are displayed (Figure B-8).



```
60          01   HV-DATA.
61               05  HV-CUSTNO        PIC S9(9)  COMP VALUE +0.
62               05  HV-ACCTNO        PIC S9(9)  COMP VALUE +0.
63               05  HV-BALANCE       PIC S9(6)V9(2)  COMP-3 VALUE +0.
64
```

Figure B-8   Structure details

Let’s assume we need to expand the `HV-CUSTNO` definition to accommodate our growing customer base. From the program details screen (Figure B-4 on page 446), select the *View program data elements action* (Figure B-9).



IBM WebSphere Studio Asset Analyzer for Multiplatforms

Program data elements

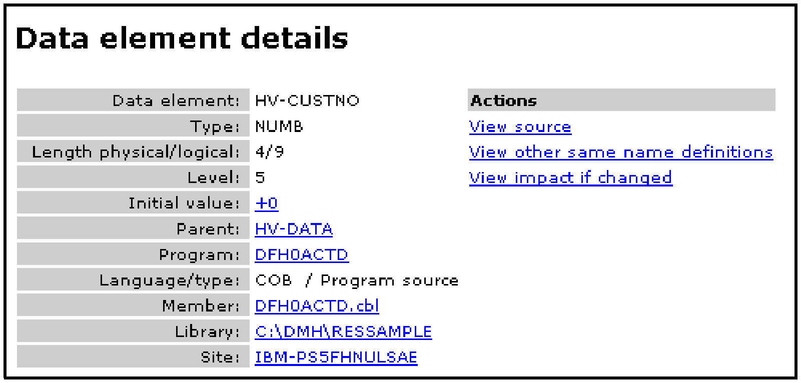
Program: **DFH0ACTD**

Data elements					
		Data element	**Type**	**Physical len**	**Logical len**
(+)	01	SQLCA	GRP	136	0
	01	TMP	CHAR	40	40
(+)	01	SQL-MESSAGE	GRP	20	0
(+)	01	SQLERROR	GRP	85	0
(+)	01	SQLSTAT	GRP	15	0
(+)	01	SQLERRORP	GRP	18	0
(+)	01	ABEND-MESSAGE	GRP	16	0
(+)	01	HV-DATA	GRP	13	0
(+)	01	DFHCOMMAREA	GRP	16	0
(+)	01	DFHEIBLK	GRP	82	0

IBM WebSphere Studio Asset Analyzer for Multiplatforms - 11 Oct 2002 21:36

Figure B-9 Program data structures and elements

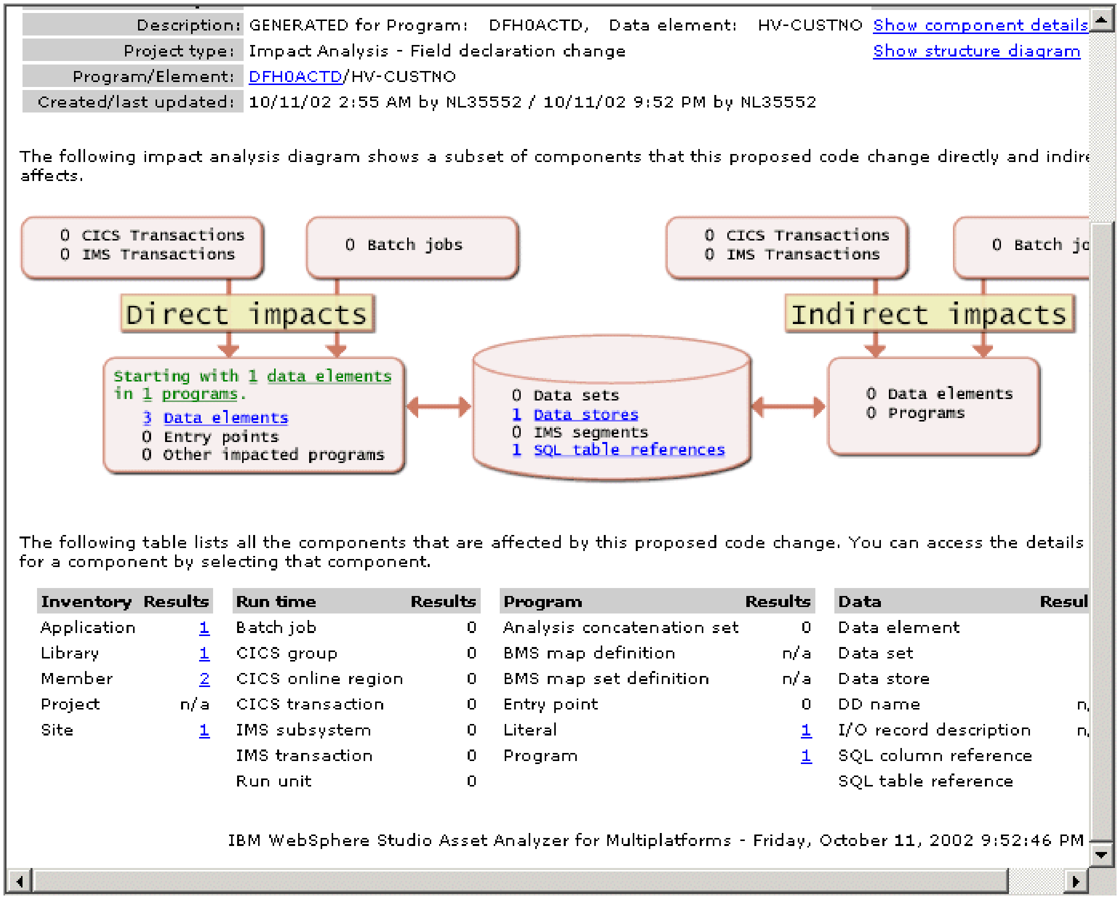
After drilling down to the details of CUSTNO (Figure B-10), you can select *View impact if changed* , which will take you to the panel shown in Figure B-11.



Data element details

		Actions
Data element:	HV-CUSTNO	View source
Type:	NUMB	View other same name definitions
Length physical/logical:	4/9	View impact if changed
Level:	5	
Initial value:	+0	
Parent:	HV-DATA	
Program:	DFH0ACTD	
Language/type:	COB / Program source	
Member:	DFH0ACTD.cbl	
Library:	C:\DMH\RESSAMPLE	
Site:	IBM-PS5FHNULSAE	

Figure B-10 Data element details (extract)



Description:	GENERATED for Program: DFHOACTD, Data element: HV-CUSTNO	Show component details
Project type:	Impact Analysis - Field declaration change	Show structure diagram
Program/Element:	DFHOACTD/HV-CUSTNO	
Created/last updated:	10/11/02 2:55 AM by NL35552 / 10/11/02 9:52 PM by NL35552	

The following impact analysis diagram shows a subset of components that this proposed code change directly and indire affects.

The following table lists all the components that are affected by this proposed code change. You can access the details for a component by selecting that component.

Inventory	Results	Run time	Results	Program	Results	Data	Resul
Application	1	Batch job	0	Analysis concatenation set	0	Data element	
Library	1	CICS group	0	BMS map definition	n/a	Data set	
Member	2	CICS online region	0	BMS map set definition	n/a	Data store	
Project	n/a	CICS transaction	0	Entry point	0	DD name	n,
Site	1	IMS subsystem	0	Literal	1	I/O record description	n,
		IMS transaction	0	Program	1	SQL column reference	
		Run unit	0			SQL table reference	

IBM WebSphere Studio Asset Analyzer for Multiplatforms - Friday, October 11, 2002 9:52:46 PM

Figure B-11 Impact analysis overview screen

Figure B-11 provides an overview of the potential affected artifacts on changing `custno`. It not only shows the direct impact, in this case three data elements in one program affecting a data store that is referenced by SQL, but also shows the indirect impact if, for instance, the data store is used in other programs.

Once more, further analysis can be done by just clicking your way through to assess what really needs to be changed. After analysis, the identified work can be done using Enterprise Developer.

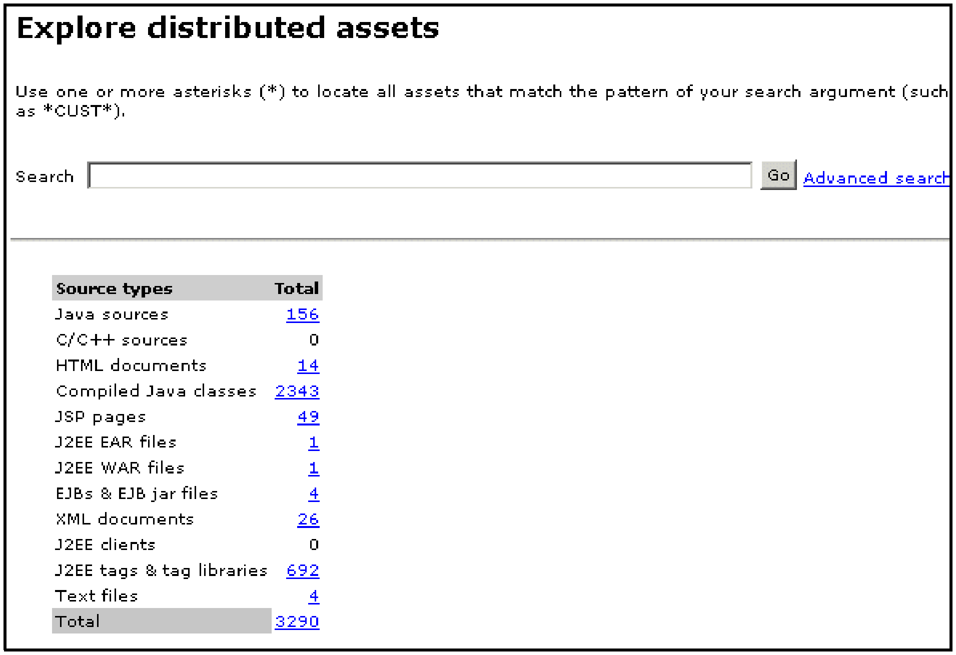
> **Tip:** From the HELP information within the product, you can go to WSAA's Web site. On this Web site, after selecting the library entry, there are a number of scenarios available that take you in detail through the functionality of the product.

Impact analysis distributed

The goal is the same as for impact analysis on the mainframe. We want to assess the impact that a change would have. The approach for distributed artifacts is different due to the fact that the information model for distributed artifacts is different from the one used for mainframe artifacts; therefore the capabilities of Asset Analyzer are different.

> **Note:** We loaded our sample `ItsoMyTrade` into Asset Analyzer including all EGL-generated Java classes, which can be reflected in amazingly complex figures, proving that EGL exploits many small discrete functional pieces of code when generating. It is unlikely that you would want to do this in a production environment, because you should not have an interest in changing the generated Java code. Another consequence of using our sample is that Asset Analyzer did detect some constructs from WebSphere Version 5 that it does not support yet, with the result that not all the information was loaded, which in itself is a good thing. The other loaded sample is the well-known mini-bank sample.

Lets have a look at the distributed artifacts within our database by selecting the distributed artifacts (Figure B-12).



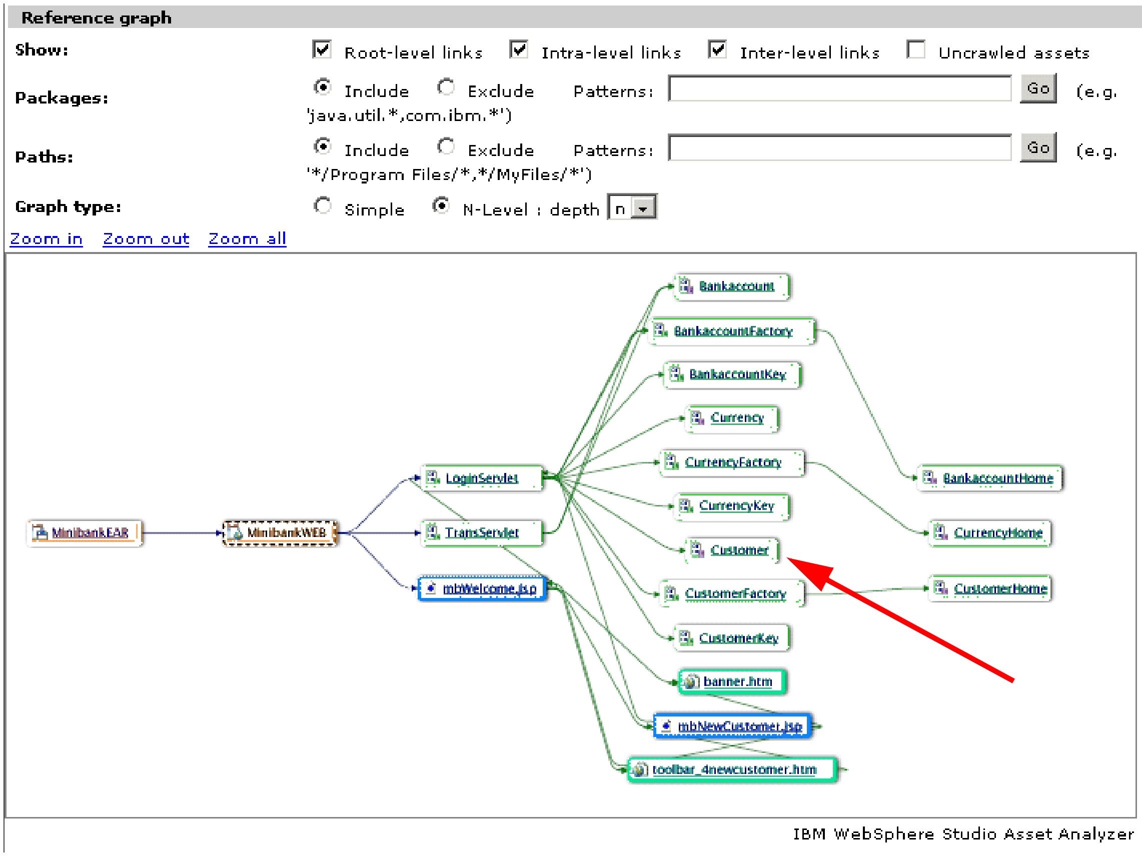
Explore distributed assets

Use one or more asterisks (*) to locate all assets that match the pattern of your search argument (such as *CUST*).

Source types	Total
Java sources	156
C/C++ sources	0
HTML documents	14
Compiled Java classes	2343
JSP pages	49
J2EE EAR files	1
J2EE WAR files	1
EJBs & EJB jar files	4
XML documents	26
J2EE clients	0
J2EE tags & tag libraries	692
Text files	4
Total	3290

Figure B-12 Distributed artifacts (extract)

Let’s have a look at the single `.war` file we have (Figure B-13).



Reference graph

Show: ☑ Root-level links ☑ Intra-level links ☑ Inter-level links □ Uncrawled assets

Packages: ● Include ○ Exclude Patterns: [ ] Go (e.g. 'java.util.*,com.ibm.*')

Paths: ● Include ○ Exclude Patterns: [ ] Go (e.g. '*/Program Files/*,*/MyFiles/*')

Graph type: ○ Simple ● N-Level : depth n

Zoom in Zoom out Zoom all

Figure B-13 Mini-bank WAR file reference graph

Note that the representation differs from what we have seen thus far. Functions are made available here to hide the things you are not interested in.

Click *Customer* to go deeper into the diagram (Figure B-14).

Through the graphical capabilities, you get insight into the structure of your applications.

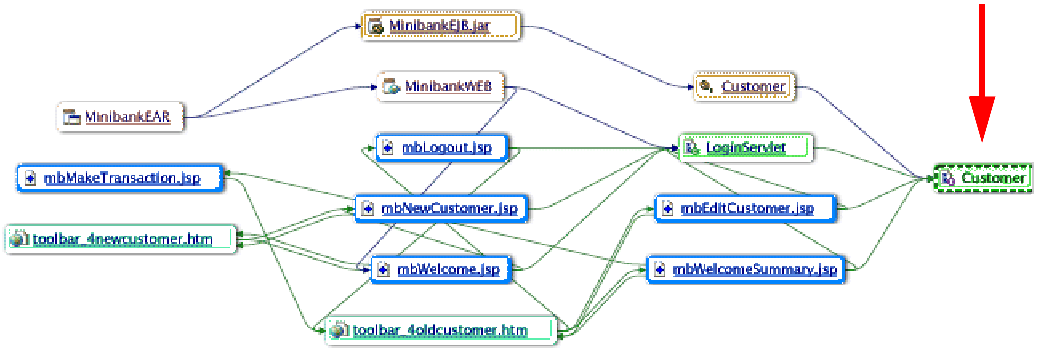
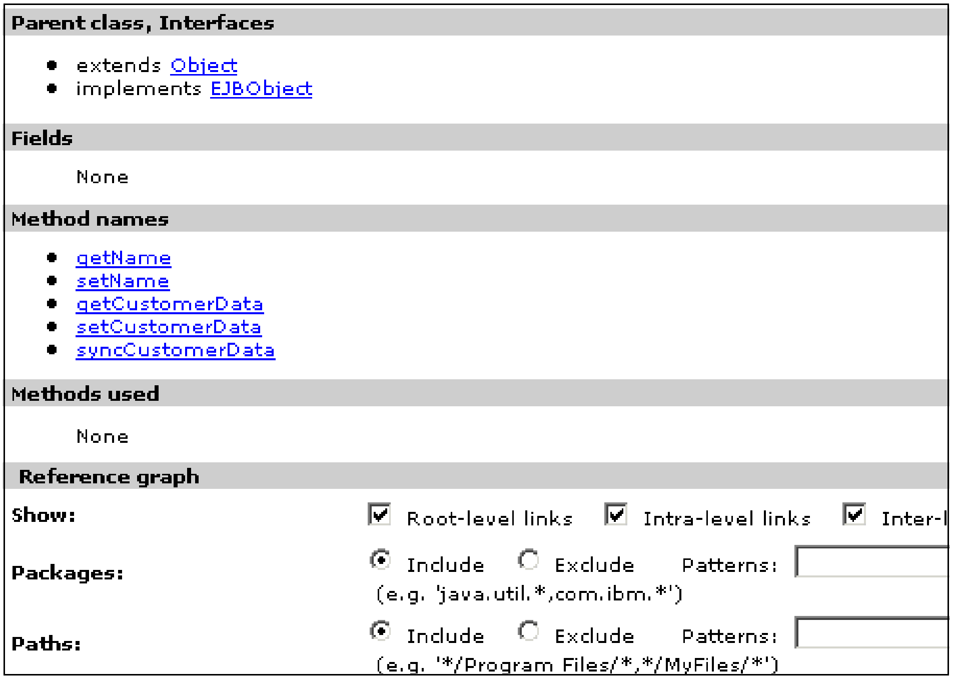


Figure B-14 Customer reference graph

On the top of the pane displaying the diagram in Figure B-14, you can find which methods are defined for the `Customer` class (Figure B-15).



Parent class, Interfaces

- extends Object
- implements EJBObject

Fields

None

Method names

- getName
- setName
- getCustomerData
- setCustomerData
- syncCustomerData

Methods used

None

Reference graph

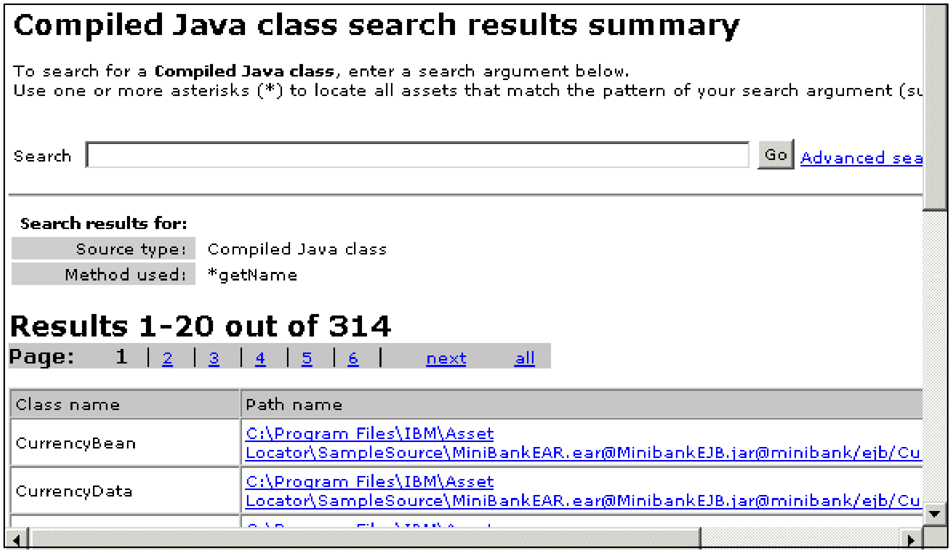
Show: ☑ Root-level links ☑ Intra-level links ☑ Inter-l

Packages: ● Include ○ Exclude Patterns:
(e.g. 'java.util.*,com.ibm.*')

Paths: ● Include ○ Exclude Patterns:
(e.g. '*/Program Files/*,*/MyFiles/*')

Figure B-15 Method names used by customer

If you click a method name, for instance `getName`, you will get a list of classes that use this method. So in effect you have done impact analysis to find out the potential impact if the `getName` method is changed (Figure B-16).



Compiled Java class search results summary

To search for a **Compiled Java class**, enter a search argument below.
Use one or more asterisks (*) to locate all assets that match the pattern of your search argument (su

Search [ ] Go Advanced sea

Search results for:

Source type:	Compiled Java class
Method used:	*getName

Results 1-20 out of 314

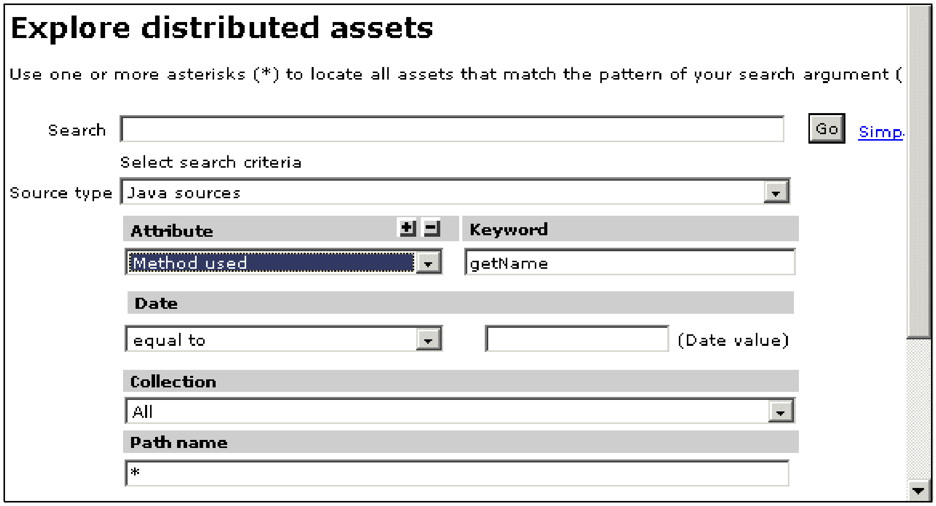
Page: 1 | 2 | 3 | 4 | 5 | 6 | next all

Class name	Path name
CurrencyBean	C:\Program Files\IBM\Asset Locator\SampleSource\MiniBankEAR.ear@MinibankEJB.jar@minibank/ejb/Cu
CurrencyData	C:\Program Files\IBM\Asset Locator\SampleSource\MiniBankEAR.ear@MinibankEJB.jar@minibank/ejb/Cu

Figure B-16 References to the getName method

Another way of accomplishing this is to exploit the Advanced Search capabilities that Asset Analyzer offers.

When you click *Advances Search* on the Explore distributed assets page (Figure B-12 on page 452) you get the Advanced Search dialog (Figure B-17).



Explore distributed assets

Use one or more asterisks (*) to locate all assets that match the pattern of your search argument (

Search [ ] Go Simp

Select search criteria

Source type: Java sources

Attribute: Method used

Keyword: getName

Date: equal to [ ] (Date value)

Collection: All

Path name: *

Figure B-17 Advanced search

If you click *Go*, you would see the same result as previously. With the pull-downs you can create complex queries on all source types and attributes of those source types. You can set query arguments for multiple attributes. You can confine the search to a specific collection, which can be all artifacts defined in the database due to a specific load operation.

So although the interface and the means vary somewhat, impact analysis for distributed artifacts is well supported.

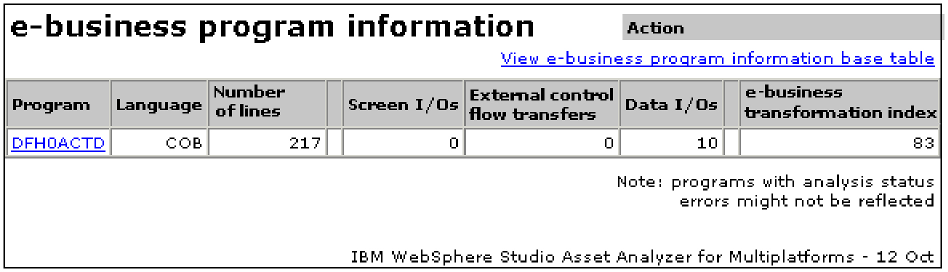
Reuse of existing code

All the functionality covered so far helps you to get insight into your existing systems and can make you more productive while maintaining your systems.

Another challenge within system development is how to leverage the existing implementation of business rules buried in your systems. Typically various business rules are implemented within a single transaction. It would be nice if one could get some guidance on where to look for potential reusable code.

Asset Analyzer provides some help here. Based upon various attributes of system artifacts whose weight can be changed, Asset Analyzer provides you with an e-business rating of your existing code artifacts.

If we go back to our mainframe sample, we can have a look at this capability (Figure B-18).



e-business program information

View e-business program information base table

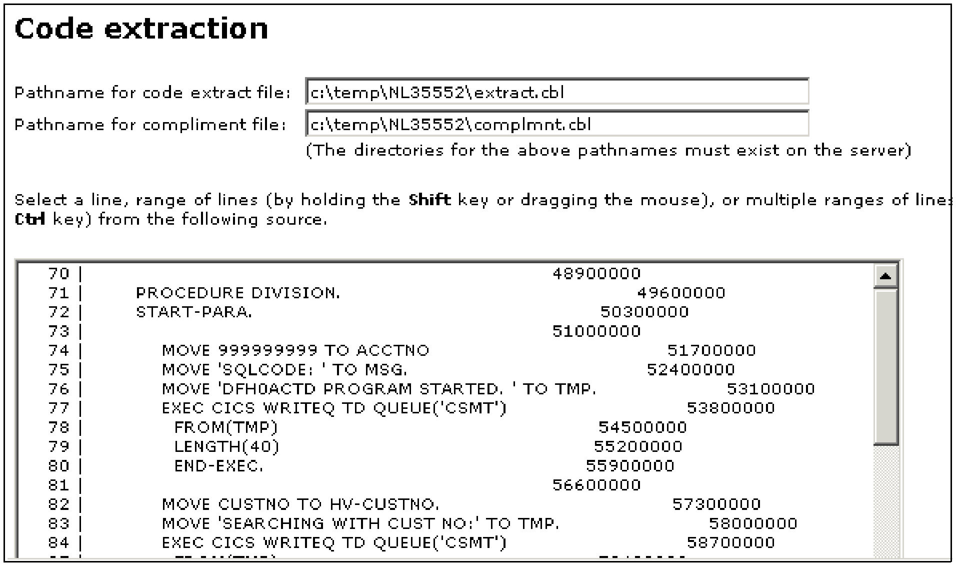
Program	Language	Number of lines		Screen I/Os	External control flow transfers	Data I/Os		e-business transformation index
DFH0ACTD	COB	217		0	0	10		83

Note: programs with analysis status errors might not be reflected

IBM WebSphere Studio Asset Analyzer for Multiplatforms - 12 Oct

Figure B-18 e-business transformation index

Let's assume that you decide to have a look at the program to actually see if there is some code with reuse potential. For this, you can invoke the code extraction function (Figure B-19).


Code extraction

Pathname for code extract file: c:\temp\NL35552\extract.cbl

Pathname for compliment file: c:\temp\NL35552\complmnt.cbl

(The directories for the above pathnames must exist on the server)

Select a line, range of lines (by holding the **Shift** key or dragging the mouse), or multiple ranges of line
Ctrl key) from the following source.

```
                                                  48900000
     PROCEDURE DIVISION.                                   49600000
     START-PARA.                                        50300000
                                                  51000000
        MOVE 999999999 TO ACCTNO                             51700000
        MOVE 'SQLCODE: ' TO MSG.                           52400000
        MOVE 'DFH0ACTD PROGRAM STARTED. ' TO TMP.                  53100000
        EXEC CICS WRITEQ TD QUEUE('CSMT')                     53800000
          FROM(TMP)                                     54500000
          LENGTH(40)                                    55200000
          END-EXEC.                                    55900000
                                                  56600000
        MOVE CUSTNO TO HV-CUSTNO.                            57300000
        MOVE 'SEARCHING WITH CUST NO:' TO TMP.                  58000000
        EXEC CICS WRITEQ TD QUEUE('CSMT')                     58700000
```

Figure B-19 Code extraction

Another function that can be helpful is the connector builder assistant function. Based upon the existing COBOL interface information, Asset Analyzer will guide you through a process to identify the parameters that should be used when exploiting this transaction from an e-business perspective. Based upon that, it will generate the I/O information in XML format, which can be fed into Enterprise Developer to create the appropriate connector.

Summary

We have rushed through WebSphere Studio Asset Analyzer with a focus on its impact analysis capabilities. We hope you enjoyed the ride and have obtained a feel for its capabilities.

Functionality like this is critical to make your developers more productive and take steps on the path leading to the reuse of existing code. As such, WebSphere Studio Asset Analyzer is an important asset in meeting the challenge of supporting the IBM enterprise modernization strategy.

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

```
ftp://www.redbooks.ibm.com/redbooks/SG246806
```

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 redbook form number, SG24-6806.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg246806code.zip	Sample code for following the trade sample though the book

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space	3 GB
Operating System	Windows 2000 or Windows NT
Processor	700 MHz or better
Memory	512 MB, recommended 784 MB

How to use the Web material

Unzip the contents of the Web material **sg246806code.zip** file onto your hard drive. This creates a folder structure **c:\SG246806\sampcode\xxxx**, where *xxxxx* refers to a chapter in the book:

createstruts	Creating a Struts-based Web application
jspaction	Adding JSPs and actions to the application
sade	Struts application diagram editor
implegl	Implementing EGL actions
implejb	Implementing EJB actions
implzos	Generating COBOL for z/OS
implcics	Implementing CICS actions
webserv	Implementing and using Web services
zoside	Developing for z/OS
zosxml	XML enablement

DB2 installation

The examples in this book assume that you have **DB2 Version 7.2 with Fixpack 6 or 7** installed on your machine.

Also it is required that you run the command file `java12\usejdbc2.bat` to enable the JDBC Version 2 support for data sources.

Abbreviations and acronyms

AAT	application assembly tool
API	application programming interface
BMS	basic mapping support
CCF	Common Connector Framework
CCI	common client interface
CICS	Customer Information Control System
CTG	CICS Transaction Gateway
DBMS	database management system
DOM	document object model
EAR	enterprise application archive
ECI	external call interface
EGL	enterprise generation language
EIS	enterprise information system
EJB	Enterprise JavaBeans
EJS	Enterprise Java Server
EPI	external presentation interface
EXCI	external CICS interface
FFS	foreign file system
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IIOP	Internet Inter-ORB Protocol
IMS	Information Management System
ITSO	International Technical Support Organization
J2CA	J2EE connector architecture
J2C	J2EE connector architecture
J2EE	Java 2 Enterprise Edition
JAR	Java archive
JCL	job control language
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JNDI	Java Naming and Directory Interface
JSP	JavaServer Pages
MVC	model-view-controller
RAD	rapid application development
RAR	resource adapter archive
RDBMS	relational database management system
RECD	remote edit compile debug
RMI	Remote Method Invocation
SCCI	source control control interface
SCM	software configuration management
SCMS	source code management systems
SDK	Software Development Kit
SOAP	Simple Object Access Protocol (also called Service Oriented Architecture Protocol)
SQL	structured query language
TCP/IP	Transmission Control Protocol/Internet Protocol
TSO	TIme Sharing Option

UDDI	Univeral Description, Discovery, and Integration
UOW	unit of work
URL	uniform resource locator
UTC	univeral test client
WAR	Web application archive
WSDL	Web Service Description Language
WWW	World Wide Web
XMI	XML metadata interchange
XML	eXtensible Markup Language
XSD	XML schema definition

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 464.

- *WebSphere Studio Application Developer Programming Guide*, SG24-6585
- *Web Services Wizardry with WebSphere Studio Application Developer,* SG24-6292
- *Self-Study Guide: WebSphere Studio Application Developer and Web Services,* SG24-6407
- *IBM WebSphere V4.0 Advanced Edition Handbook,* SG24-6176
- *WebSphere Version 4 Application Development Handbook,* SG24-6134
- *Programming J2EE APIs with WebSphere Advanced,* SG24-6124
- *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2,* SG24-6284
- *EJB Development with VisualAge for Java for WebSphere Application Server,* SG24-6144
- *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server,* SG24-5754

Other resources

These publications are also relevant as further information sources:

- *The Rational Unified Process, An Introduction* (Second Edition) by Philippe Kruchten, published by Addison-Wesley, ISBN 0-201-70710-1
- *XML for the Enterprise - Providing an XML Interface to a CICS Application,* found on the samples CD or at the Web site:

  `http://www-3.ibm.com/software/ad/studioenterprisedev/library/`

Referenced Web sites

These Web sites are also relevant as further information sources:

- *IBM software for application development and Enterprise Developer*
  `http://www.ibm.com/software/ad/`
  `http://www.ibm.com/software/ad/studioedm/`
- *WebSphere Developer Domain*
  `http://www7b.software.ibm.com/wsdd/`
- *IBM Patterns for e-business*
  `http://www.ibm.com/developerworks/patterns`
- *Eclipse*
  `http://www.eclipse.org`
- *Struts*
  `http://jakarta.apache.org/struts`
- *Sun JavaServer Pages*
  `http://java.sun.com/products/jsp`
- *XML schemas*
  `http://www.w3.org/XML/Schema`
- *Common Versions System*
  `http://www.cvshome.org`
  `http://www.cvsnt.org`
- *Rational*
  `http://www.rational.com`

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

IBM

Redbooks

Legacy Modernization with WebSphere Studio Enterprise Developer

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Legacy Modernization with WebSphere Studio Enterprise Developer

Creating enterprise applications with Struts

Introducing enterprise generation language

Developing for z/OS

The ability to connect components is the first step in modernizing your application portfolio. In this IBM Redbook, we look at a real-world example of creating and connecting a Web application to enterprise business logic using the Struts-based model-view-controller (MVC) framework and associated tooling within WebSphere Studio Enterprise Developer that makes this a snap.

To address the needs of large enterprises, a model-based paradigm for building applications in a Struts-based MVC framework is being delivered in the WebSphere Studio Suite. It provides a visual construction and assembly-based environment supporting the implementation of enterprise-level applications and including support for the multiple developer roles and technologies required by those applications. Examples of the technologies supported include HTML, Java, servlet, EJB, COBOL, EGL, PL/I, and connectors.

EGL is a high-level language that supports the development of applications in either WebSphere (Java) or traditional transactional environments (CICS). EGL's focus is to allow developers of various backgrounds to be able to write mission-critical business processes for the Internet, which can be leveraged from Struts-based Web applications.

This redbook introduces a sample application that encompasses Enterprise Developer concepts and best practices.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6806-00 ISBN 0738425710