Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 105.

First Edition (October 2001)

This edition applies to the beta version of WebSphere Studio Application Developer for use with the Windows NT and Windows 2000 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.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8  Building 662
P.O. Box 12195
Research Triangle Park, NC 27709-2195

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2001. All rights reserved.
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Preface ............................................. vii
The team that wrote this Redpaper ................................ vii
Special notice .................................... vii
IBM trademarks ................................... viii
Comments welcome ................................ viii

Chapter 1. Introduction ................................ 1
1.1 WebSphere Studio Site Developer ................... 2
1.2 WebSphere Studio Application Developer ........... 3
1.3 For more information ................................ 3

Chapter 2. Developing applications .................... 5
2.1 Workspace ...................................... 6
2.2 Projects ........................................ 6
  2.2.1 Creating a new project ....................... 7
  2.2.2 Project build options ....................... 7
2.3 Workbench window ............................... 8
2.4 Working with perspectives ......................... 10
  2.4.1 Customizing perspectives ...................... 12

Chapter 3. J2EE testing environment .................. 15
3.1 Creating a server project ......................... 16
3.2 Preparing to test ................................ 18
  3.2.1 Publishing your code to the server .......... 18
3.3 Working with the Server perspective ............ 19

Chapter 4. Working with Enterprise Archive projects .... 21

Chapter 5. Working with Web projects ................. 23
5.1 Working in the Web perspective .................. 24
5.2 Working with HTML files ....................... 25
5.3 Creating JavaServer Pages ....................... 30
5.4 Working with graphics .......................... 31
  5.4.1 Importing graphic files ...................... 32
  5.4.2 WebArt Designer ............................ 33
5.5 Testing your Web project ........................ 34

Chapter 6. Working with EJB projects .................. 37
6.1 Creating enterprise beans ....................... 37
6.2 Editing enterprise beans ........................................... 39
6.3 Mapping enterprise beans to database tables ............... 40
6.4 Creating Access beans ............................................. 42
6.5 Validating enterprise beans ................................... 42
6.6 Generating the deployment code ............................... 42
6.7 Testing enterprise beans ........................................ 43
   6.7.1 EJB Test Client ............................................ 43

Chapter 7. Working with Java projects .......................... 47
7.1 Creating a new Java project ..................................... 48
7.2 Creating Java elements ......................................... 49
7.3 Specifying alternate JREs ................................. 51
7.4 Testing Java projects ....................................... 52

Chapter 8. Working with Web services ......................... 55
8.1 Searching UDDI registries for Web services ............... 57
8.2 Creating a Web service ..................................... 58
8.3 Testing a Web service ....................................... 64
8.4 Adding Web services to a UDDI registry ................. 65

Chapter 9. Working with XML ..................................... 67
9.1 Creating XML files ........................................... 68
9.2 Creating DTD files .......................................... 70
9.3 Relating an XML file to a DTD file ....................... 71
9.4 Creating XML schema ....................................... 73
9.5 Relating an XML file to an XML schema file ........... 74
9.6 Using the XML catalog ..................................... 74
9.7 XML-to-XML mapping ..................................... 76
   9.7.1 Transforming the files from the old format to the new 77
9.8 RDB-to-XML mapping ..................................... 78
9.9 SQL Query Builder ........................................ 79

Chapter 10. Working with databases ......................... 81
10.1 Working with existing databases ......................... 81
   10.1.1 Connecting to a database .......................... 82
   10.1.2 Importing the database design .................... 83
10.2 Working database designs ................................. 84
   10.2.1 Building and executing SQL statements .......... 85

Chapter 11. Analyzing performance ......................... 87
11.1 IBM Agent Architecture .................................... 88
11.2 Getting started ........................................... 88
11.3 Using the Profiling perspective ............................ 89
   11.3.1 Profiling projects view ............................ 89
Preface

This Redpaper introduces IBM WebSphere Studio Application Developer, the follow-on technology for VisualAge for Java and WebSphere Studio. This is one of a new generation of application development tools based on a common workbench and integrated set of tools. It provides end-to-end support for J2EE application development, testing, and deployment, taking advantage of the latest technology and open standards.

This paper will give you a broad overview of the product, showing you the highlights of its features and giving you insight into the look and feel of the new IBM application development tools.

The team that wrote this Redpaper

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

Carla Sadtler is a Senior Software Engineer at the International Technical Support Organization, Raleigh Center. She writes extensively in the WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Anish Jacob is an IT specialist in IBM Global Services India, Bangalore. He has more than 5.5 years of total industry experience and more than 3.5 years of experience in the area of application development and Web technologies. He holds a degree in Mechanical Engineering from the University of Calicut, India.

Special notice

This publication is intended to give Java application developers an overview of the function available in IBM WebSphere Studio Application Developer. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM WebSphere Studio Application Developer. See the PUBLICATIONS section of the IBM Programming Announcement for IBM WebSphere Studio Application Developer for more information about what publications are considered to be product documentation.
IBM trademarks

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

- e (logo)®
- IBM ©
- CICS
- DB2
- DB2 Universal Database
- Redbooks
- Redbooks Logo
- SP
- VisualAge
- WebSphere

Comments welcome

Your comments are important to us!

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

- Use the online Contact us review redbook form found at:
  [ibm.com/redbooks](http://ibm.com/redbooks)
- Send your comments in an Internet note to:
  [redbook@us.ibm.com](mailto:redbook@us.ibm.com)
- Mail your comments to the address on page ii.
Chapter 1. Introduction

IBM is introducing a new generation of application development tools based on a common set of frameworks and services that provide a tool integration platform. These tools, designed for the J2EE standards of development and deployment, combine the best features from VisualAge for Java and WebSphere Studio and integrate them with the latest technology for application development.

The foundation for this new generation of tools is the IBM WebSphere Studio Workbench. WebSphere Studio Workbench is an open, portable universal tool building and integration platform based on the IBM's Eclipse project technology. It provides a common user interface and a base set of services upon which all new tools that integrate with the WebSphere software platform will be built on. New functionality will be added in the form of plug-ins.

The advantage of Workbench-based offerings is having a common, easy-to-use interface, or workbench. The core common services include such things as resource management, a team programming model, debugging, and workspace synchronization. It provides the same tool platform regardless of the type of development being done. Moreover, the Workbench-based product will have the same look and feel regardless of vendor.

The beta code for the first two Workbench-based application development tools was made available in July 2001:
WebSphere Studio Site Developer is the follow-on technology for WebSphere Studio. It consists of the IBM WebSphere Studio Workbench with a set of plug-ins targeted toward Web application developers.

WebSphere Studio Application Developer is for the end-to-end development of Java applications in an integrated environment. It is the follow-on technology for VisualAge for Java and WebSphere Studio. It consists of the IBM WebSphere Studio Workbench and a set of plug-ins targeted toward enterprise application developers. This includes the capabilities of WebSphere Studio Site Developer and adds enterprise integration capabilities.

1.1 WebSphere Studio Site Developer

WebSphere Studio Site Developer combines many of the best features of WebSphere Studio Pro 4.0 and VisualAge for Java along with new technology to provide a comprehensive development environment for Web and Java developers. It includes:

- Java development tooling with features such as content assist, integrated debugging features, automatic compile, search support, refactoring, scrapbook facility, syntax highlighting, JDK switching, and much more.
- Web module tooling to develop and deploy Web applications. This includes tools to create servlets based on Servlet 2.2 specifications, an advanced page designer to create JSPs based on JSP 1.1 and HTML pages, and graphics support.
- XML tooling that provides components for building DTDs, XML schemas, XML, XSLT, and mappings between XML and different back-end stores.
- Web services development support based on open, cross-platform standards such as UDDI4J, SOAP, and WSDL.
- SQL query tools to build and execute SQL statements against relational databases. Tooling to convert SQL to XML allows you to create XML, XSL, DTD, XSD, and HTML files for implementation in applications.
- A Test environment for Web projects. The server tooling includes a local copy of IBM WebSphere Application Server Advanced Single Server Edition for Multiplatforms (WebSphere Application Server) runtime environment and a Tomcat unit test environment. In addition it supports WebSphere Application Servers remote to the workbench and Apache Tomcat runtime running locally.
- Team development support using Concurrent Versions System (CVS). In future releases, other repositories will be available.
1.2 WebSphere Studio Application Developer

WebSphere Studio Application Developer provides the functionality included with WebSphere Studio Site Developer. In addition, it expands the support for some of those features and adds support for EJB and enterprise application development. In addition to the capabilities of the WebSphere Studio Site Developer, it provides:

- EJB tooling that provides a complete development and test environment for enterprise beans coded to the EJB 1.1 specification, including database mapping for entity beans and XML-based deployment descriptors.
- Creation of all required J2EE packaging artifacts including EJB modules, Web modules, Enterprise Application (EAR) modules.
- Tracing and performance analysis tools.

The Enterprise Connector Feature is an optional feature that can be added to the Application Developer. It includes:

- Bi-model EAB/SAP tools support for CCF and JCX Connectors
- Implementation of J2EE Connectors Architecture (JCX1.0)
- J2EE-enabled Connectors (SAP, CICS)
- New J2EE Connectors (PeopleSoft, Oracle Financials, JD Edwards)

1.3 For more information

For more information on Eclipse technology and the IBM WebSphere Studio Workbench, see:

- [http://eclipse.org](http://eclipse.org) or [http://eclipsecorner.org](http://eclipsecorner.org)
Chapter 2. Developing applications

The new tools have a common GUI interface called the workbench. The workbench is where all resource management, viewing, and editing takes place. The plug-in components run in the context of the workbench.

The workbench is designed to meet the needs of developers according to their particular roles, for example, J2EE developer, Web developer, Java developer, or Web services developer. In assuming that the developer is acting in one of these roles at a time, it is logical to assume that only a subset of the many available functions will be of interest to him.

The workbench organizes tasks into perspectives to provide an interface to the user that contains the information and wizards that he will most likely need. Perspectives consist of an initial set of views and action sets that can be customized to suit individual needs or preferences. Multiple perspectives can be open and the developer can switch between them depending on the task at hand.
2.1 Workspace

The workspace is a collection of projects and their resources arranged into a tree structure. Projects are at the root of the tree, with folders and files under the projects. The workspace actually maps to a physical location. There is a default location that can be used for all projects, or you can specify a separate location when creating a project. You can change the default workspace when you start Application Developer by giving it a parameter.

Using different workspaces for different projects is one way to cut down on the size of the workspace. The amount of elements in a workspace can have an impact on performance, especially if there are large or complex applications. One suggestion is to create multiple shortcuts to start Application Developer, each pointing to a different workspace.

2.2 Projects

Application elements are organized at the highest level into projects. A project is a grouping of the resources needed for specific tasks. It consists of a collection of folders and files related to a specific task.

There are unique project types for developing J2EE applications:

- EAR projects represent contain the resources needed for enterprise applications and can contain a combination of Web modules, EJB modules, JAR files, and application client modules. An EAR project is deployed in the form of a .ear file.
- Web projects: Web projects contain the resources needed for Web applications, including servlets, JSPs, Java files, static documents (for example, HTML pages or images), and any associated metadata. A Web project is deployed as a Web module (.war file).
- EJB projects contain the resources for EJB applications. An EJB project is deployed as an EJB module (.jar file).
- Application client projects contain the resources needed for application client modules. An application client project is deployed as a JAR file.
- Java projects contain Java code. They have an associated Java builder that can incrementally compile Java source files as they are changed. Java projects can be exported as JAR files or into a directory structure.
- Plug-in projects are used to create workbench plug-ins, plug-in components, and fragments.
Server projects describe the test environment for applications. Server projects contain the information necessary to deploy an application to an application server for testing.

Examples illustrating various types of projects and development techniques are shipped with Application Developer. You can load the supplied examples to get familiar with the development tools, or to use them as a starting point for your own application.

2.2.1 Creating a new project

Since projects are the root of any application development, you will always start a new application by creating one or more projects. The workbench provides wizards to create each specific type of project.

To begin, select File -> New -> Project. From here, you can select the appropriate project type, and the wizard for that type will open. All the wizards for creating projects are similar, with slight variations required for the project type. In general, when you create a new project you specify the following:

- Name of the project.
- The workspace location to use.
- File organization.
- The build output folder.
- Projects and libraries that need to be in the build classpath.

2.2.2 Project build options

 Builders associated with resources perform the necessary actions to ensure the resources are executable. For example, the Java builder, javac, builds class files for the Java source. You can choose to have builder run automatically as resources are changed, or you can choose to build projects manually. This choice is made once for all projects in the workbench preferences.

Automatic builds are convenient in that they keep all resources up to date and ensure that inconsistencies are found quickly. The builds are incremental, affecting only those resources changed since the last build.
On the other hand, manual builds can be done less frequently and can be full or incremental. They can be done for the entire workspace or for selected projects. In situations where you find that automatic builds are not providing much value to your programming efforts, you can use manual builds to reduce any performance impact you might be seeing with automatic builds. If you choose to do the builds manually, you will need to remember to do this before attempting to execute the code.

2.3 Workbench window

As an example, let’s take a look at the workbench window shown in Figure 2-1. This window has the J2EE perspective active. This perspective is built with the J2EE developer in mind. The toolbar will consist of the appropriate tools to create or work with J2EE components and the default views will be those considered most useful for J2EE development.

The workbench window will contain several elements:

- A shortcut bar on the left, allowing you to switch between perspectives.
  
  In Figure 2-1, there are several perspectives to choose from including the Resource perspective, the Java perspective, the Web perspective, the J2EE perspective (current), and the Help perspective.

- A toolbar at the top, with tools appropriate for the current editor.
  
  For example, in Figure 2-1, the toolbar includes icons to create J2EE resources such as enterprise applications, EJB projects, and Web projects.

- Multiple views, either single or stacked, determined by the perspective.
Most the data you see in the window are views. The views can be single views, or stacked in notebook fashion, switchable by selecting the appropriate notebook tab.

- The J2EE view (1 in Figure 2-1) shows the application resources organized as J2EE components (enterprise applications, Web modules, EJB modules, etc.). This organization is unique to J2EE. The same type of view for Java developers (the Java view) would show packages instead.

- The editor area in the upper-right corner (2) allows you to edit or browse the selected resource. The editor type will depend on the resource type. Multiple resources can be in the editor area, but only one will be active. You can switch between them by selecting the resource at the top of the area (3). The attributes that can be edited are selectable in the tabs below the editor area. For example, we have selected an enterprise application in the navigator view. The editor area will open the deployment descriptor file for the enterprise application, called application.xml. You can change the general
settings, change the security roles, or view the source by selecting the appropriate tab (4).

When you open a resource, the workbench first checks with its registered editors. If no registered editors are found for the resource type, it will check the operating system for registered editors for that resource type. If an external editor is located it will be launched. If no registered editor for the file type exists, a text editor is used.

The workbench supports OLE document editors, such as Microsoft Word. If you create a file with a .doc extension, workbench will open this document in Word at its editor area. These type of editors are called embedded editors.

- The Navigator view shows a tree view of the projects and resources that are in the workspace. Projects form the root entries. In Figure 2-1 the Navigator view is stacked on the J2EE view. In other words, it is not visible. To see it, you would need to click the Navigator tab at the bottom of the view (5). It would then appear in place of the J2EE view.

- The Properties view shows properties associated with selected resources, such as file name, last modified date, file system path, editable or not, and size etc. In Figure 2-1, the Properties view is not visible. It is stacked under the Tasks view. To see the Properties view, click the Properties tab at the bottom of the view (6).

Note that resources also have a properties window which you can access from the context menu for the resource. This window will display more detailed and complex information about the resources than seen in the Properties view.

- The Task view (7) displays all the tasks and problems in the workbench. Tasks can be created by the developer to be used as markers or notes. A task can be associated with a file or a line in a specific file, or it can be a generic task that is not associated with any specific files. If a task is associated with a file, double-clicking the task will open the file in the editor.

- The Outline view (8) displays an outline of a structured file that is currently open in the editor area, and lists such structural elements.

### 2.4 Working with perspectives

As mentioned previously, there are many available perspectives. We have just been talking about one, the J2EE perspective, and a small subset of the views available. Multiple perspectives have been defined, including:

- Data
- Debug
- Help
You can have multiple perspectives available in the window, or you can have multiple workbench windows open with different perspectives. For example, if you are developing a Java application, it is likely that you will eventually use several perspectives. You would probably start out with the Java perspective, but at various times, you might need the XML perspective or the Trace perspective.

One of the most common instances of having multiple perspectives open will be in the case of using the Help files. When you click the Help menu, the Help perspective will automatically open. Now, instead of the J2EE perspective you were viewing, you will see the Help perspective. Both perspectives are available and you can easily switch among them by clicking the desired perspective icon in the shortcut bar at the left of the window.
2.4.1 Customizing perspectives

As you can see, the interface used by the developer is highly customizable. He can specify what views, wizards, etc., he sees in a perspective. He can determine how views in the window are arranged and what action items appear. Views can be dragged, resized, and stacked within a window. Each view contains toolbars that are specific to the view, and filtering can be done to narrow down the contents of the view.

Customizing a perspective is easy. By selecting **Perspective -> Customize** from the workbench, you are given the opportunity to customize the various menus on the workbench. For example, you can shorten the list of options you initially see when clicking **File -> New** to the types of resources you plan to work with. By customizing this list, you do not eliminate any choices, but simply give yourself a shorter selection path to get to them. You can also do this for the list of perspectives and views you see when selecting **Perspective -> Open** and **Perspective -> Show View**.
You can also rearrange the views by dragging and dropping them to new locations. These settings can be saved for use next time.
Chapter 3. J2EE testing environment

Application Developer provides the ability to test JSPs, servlets, HTML, and enterprise beans. This test capability supports resources in Web projects, EJB projects, and EAR projects. There are several ways to test your J2EE applications on an application server:

- Using the one of the test environments supplied with Application Developer. Currently these are:
  - WebSphere Advanced Edition V4.0 test environment
  - Apache Tomcat V3.2 test environment
  - Apache Tomcat V4.0 test environment

  In our discussion we will refer to these as the test environment.

- Using an external server. Currently these are:
  - WebSphere Advanced Edition V4.0 remote server
  - Apache Tomcat V3.2 local server
  - Apache Tomcat V4.0 local server

  In our discussion we will refer to these as external servers.
A runtime environment called the TCP/IP Monitoring Server is also included with Application Developer. This is a simple server that forwards requests and responses, and monitors test activity. This runtime environment can only be run locally, and it only supports TCP/IP requests, such as HTTP requests between a J2EE application server and a Web browser. You cannot deploy projects to the TCP/IP Monitoring Server.

### 3.1 Creating a server project

Test environments and external servers are described in server projects. There are two elements that make up server projects: server instances and server configurations.

- A server configuration contains information about the server including the port the server is listening on, whether security is enabled or not, MIME types, and other basic information.
- A server instance identifies the test server. It points to a specific runtime environment and contains a reference to the server configuration it is currently using. If the server is external, the server instance defines the IP address of the machine, the WebSphere install directory, the deployment directory, and file transfer information needed to publish the application.

Wizards are provided to create server projects, configurations, and instances. Server projects and their elements are managed through the Server perspective.

A server project can contain multiple server instances and server configurations. Each server instance will point to one server configuration to use, but multiple server instances can point to the same server configuration. This allows you to use the same configuration to test an application in the test environment or on an external server.

In Figure 3-1, you can see a server configuration that has been created for a particular project. The configuration contains the environmental information needed to run the application, such as datasource definitions, and security settings.

**Note:** Apache Tomcat supports Web projects that contain servlets and JSPs.
Two server instances have been created and associated with this configuration:

- The first instance, called “PDK Test Server” is for the Application Developer WebSphere test environment.
- The second instance, called “PDK Prod Server” is for a production WebSphere. Note that although this WebSphere resides on the same machine as Application Developer (IP address is 127.0.0.1), it is external to the product and therefore considered remote.

Only one instance for a configuration can be active at a time. The active instance will have control of the ports defined in the server configuration.

When you select Run on a Server from the context menu of a resource, Server tooling will attempt to find a suitable server configuration and instance to run the resource.

Projects can be associated with a server configuration through menus in the Server perspective and project properties allow you to select the referred server instance for the project. If these preferences have not been selected, Application Developer will look for an appropriate environment to run the project. If no suitable server projects, configurations, or instances exist, Application Developer will create them automatically.
If multiple server instances have been defined for a configuration and no instance preference has been defined, the project will run on the server instance that is started for the configuration. If no instance is started, the default instance will be started and the application will run there.

### 3.2 Preparing to test

Before testing your project, you need to make sure your project is ready. Preparation for testing consists of the following:

1. If you are not using automatic builds, you need to rebuild your project. This will bring all the resources into sync and create the class files for Java programs.
2. Generate the deployment code for EJBs.
3. If you are not using automatic validation, you should validate your project.
4. If you are not using the workbench test environment, or if you want to set up multiple server instances, make sure the server project has been created and updated and that your project has been added to the appropriate configuration.
5. Publish the application to the server instance. By default, applications are automatically published when the server is started. The Servers view in the Server perspective will also indicate if you need to republish and give you the option to do so from the context menu of the server instance.

#### 3.2.1 Publishing your code to the server

To test a project, it must be published to the server. Publishing the project copies the files necessary for the project to the correct locations for the chosen server. You can choose to have projects published automatically when the server starts by clicking Window -> Preferences -> Server -> Automatically publish before starting the server, or you can publish manually using the Publish icon in the Server perspective.

When you are testing an application and make changes while the server is running, you may or may not need to republish. For example:

- JSP pages are automatically reloaded by all server types.
- Servlets are automatically reloaded when Web applications have automatic reloading on.

If in doubt, check the Server view. It will tell you if a server is synchronized or if you need to republished.
3.3 Working with the Server perspective

The Server perspective has views related to the test environment. The Server Configuration views shows the server configurations and server instances. From here you can create, delete, or alter test environment configurations.

![Server perspective](image.png)

Figure 3-2 Server perspective

The Console view acts as a console for the application server, showing messages as they occur.

At the top, in the editing area, you have a browser that the Web application elements of your application will run. You can also open application resources for editing in this area. If you find that you need to make code changes, you can switch back to the appropriate perspective to make the changes, or you can select the resource in the Server perspective navigator view and open it directly into the editing area there. Once the changes are made, you need to go back through the steps outlined earlier, making sure a build and a validate have been performed. Then stop the server instance, publish the project, and restart the server instance.

Server instances can be controlled from the Servers view. You can stop, start, publish, or start debug from here.
Application Developer Server tooling also provides a Web-based EJB test client to test your Enterprise JavaBeans (EJBs). Using this test client, you can call the home and remote interface methods of your enterprise beans. This will be discussed more in 6.7, “Testing enterprise beans” on page 43.
**Working with Enterprise Archive projects**

When developing a J2EE application, the first step is to create an Enterprise Archive (EAR) project. The EAR project, or module (.ear), effectively provides the glue that holds the various parts of the application together. The EAR project contains pointers to the components of the J2EE application, including Web modules, EJB modules, JAR files, and application client modules. It may also include libraries referenced by modules, help files and documentation to aid the deployer.

When you create any of these components, you will be asked for the EAR project they are associated with. Pointers to these elements will be created in the EAR file and a deployment descriptor for the application (application.xml) will be updated with the information.
In Figure 4-1 you can see from the application deployment descriptor that several modules belong to this EAR module. A Web module (.war file) and three EJB modules (.jar files) are in the module pointer list. This list will initially be empty, but as you create resources, for example Web modules, and specify this EAR project during their creation, this list is automatically updated.
Working with Web projects

Application Developer provides an integrated Web development environment for each different role in a Web development team, including programmers, Web masters, content authors, and graphic artists, allowing each to work on the same Web project and access the files they need. Application Developer provides for:

- Web project creation, using J2EE hierarchy.
- Creation, validation, editing, and debugging of XML, JSPs, and HTML files.
- Animation and graphic editing.
- Editing support for Cascading Style Sheets (CSS).
- HTTP/FTP import.
- FTP export (simple resource copy) to a server.
- WAR file import and export.
- Link viewing, parsing, and management.
- Servlet creation.
- Generation of Web applications using wizards that create Web resources from database queries and Java beans.

A Web application consists of HTML pages, JSPs, servlets, and other resources, and optionally, source files that can be treated as a single unit. Web applications are packaged as Web application archive (WAR) files. A WAR file is used to import a Web application into a Web server.
A WAR file has a specific hierarchical directory structure. The top-level directory is the document root of the application. The document root is where JSP pages, client-side classes and archives, and static Web resources are stored. The document root contains a subdirectory called WEB-INF, which contains the web.xml deployment descriptor and Tag library descriptor files, and classes and lib directories.

New Web projects can be created by selecting **File -> New -> Web Project.** During the definition process, the EAR project that this Web project belongs to must be specified. Once the project has been created, you will be switched automatically to the Web perspective.

### 5.1 Working in the Web perspective

Let's first take a look at the Web perspective (shown in Figure 5-1) to get an idea of how the Web project is laid out. In the Navigator view, you will see the basic structure of the Web project.

![Figure 5-1 Web perspective](image-url)
The Web project will use a J2EE directory structure with the following folders and files:

- The source folder contains the project source for beans and servlets. When resources are added to a Web project, they are automatically compiled and the generated files are added to the /classes directory. The contents of this directory are not packaged in WAR files unless an option is specified when a WAR file is created.

- The webApplication folder contains the contents of the WAR file that will be deployed to the server. It contains all the Web resources, including HTML, JSPs, and graphics needed for the application. Any files not under the webApplication folder are considered development resources (for example, .java files and .sql files), and will not be deployed when the project is unit tested or published.

- The webApplication/theme folder contains cascading style sheets and other style-related objects.

- The webApplication/WEB-INF folder contains the supporting Web resources for a Web application, including the web.xml deployment descriptor file and the classes and lib directories.

- The webApplication/WEB-INF/classes folder contains servlets, utility classes, and the Java compiler output directory. The classes in this directory are used by the application class loader to load the classes. Folders in this directory will map package and class names. The .class files are automatically placed in this directory when the Java compiler compiles the source files from the source directory. Any files placed directly in this directory will be deleted by the Java compiler when it runs.

- The webApplication/WEB-INF/lib folder is for supporting .jar files that your Web application references. Any classes contained in these .jar files will be available for your Web application.

### 5.2 Working with HTML files

The Application Developer provides an HTML editor that makes designing and creating HTML pages very easy. The interface is intuitive and can be used by people at every skill level. The designer can work with a WYSIWYG interface or directly with the source code, previewing changes as they are made. HTML files can be imported by clicking File -> Import -> File system, or they can be created using the Create an HTML File wizard (click File -> New -> HTML File).

The HTML file should be placed in the Web project webApplication folder or a folder under this. If the JSP or HTML file is not in this path, these resources will not be included in the WAR file.
You will be automatically put in the Web perspective and the new HTML file will be open in the editor area. The editor area consists of three views: the Design view, the Source view, and the Preview view. In addition there are other views to use for reference as you are creating your page.

The Preview view shows you how the page will look from a Web browser. This view is simply to let you see the actual effect of your editing changes. To edit the page, you use the Design or Source view.

The editor views are linked in that any changes you make in one are automatically reflected in the others. Any changes made in the Design view are automatically put in the source. Any updates in the source automatically show up in the Design view. For example, you could add a table from the Source view, then switch to the Design view to enter the table entries. This means you can easily move between these views, depending on which view you prefer for any given task.

Both the Design and Source views provide a wide range of helpful tools, making HTML page design very simple and intuitive.

Source view
A basic HTML structure is automatically built for you when the file is created. The Source view shows you the source that has been generated for the page. This can be manually updated. There are many tools for creating and formatting content available in the tool bar. You can also copy and paste from other sources.

A code assistance window, called content assist, is available in this view to help you finish a tag or line of code, or insert macros. The content assist will be specific to the current code, displaying only valid options for insertion at the cursor position. You can specifically request content assist (press Ctrl+space), or set preferences that cause it to pop up automatically when certain characters are typed.

Figure 5-2 shows the updated code for our example, index.html, in the Source view. The content assist window shows options for JavaScript since the cursor is within the JavaScript code.
Design view
The Design view (shown in Figure 5-3) allows you to enter the content of the page without having to provide the HTML tags necessary. You can type in content, add images, etc. It is a WYSIWYG type of interface. As you are entering the content, the source code is automatically being built.
Links view
In addition to the editor views, there are other useful views in the perspective to assist you. The Links view gives you a sketch of the resources referenced by the open file.

In Figure 5-4 you can see the elements of the index.html page in the Links view.
Figure 5-4  Links view

**Gallery view**
The Gallery view (shown in Figure 5-5) gives you a list of reusable files that you use in your design. The files are categorized and include such things as sound files, wallpaper, stylesheets, JavaScript, graphical images, and animated images.
Figure 5-5  Gallery view

The category is shown in the gallery, with the individual items in the Thumbnail view. You can drag an item from the Thumbnail view into the design.

5.3 Creating JavaServer Pages

Creating JSPs is similar to the process for creating HTML pages. Select File -> New -> JSP File. This process is the same as that used to create an HTML page. You specify a name for the JSP, choose the location (under the webApplication folder), and optionally, a stylesheet.

As with HTML, you will have three editor views: Design, Source, and Preview. These work in the same way as for HTML.
Figure 5-6  JSP source editor

The content assist feature (press Ctrl+space) in the Source view helps you in choosing the appropriate tags for the cursor location. The assist window will give you a list of JSP or HTML tags to choose from, or if you are in a scriptlet, a list of appropriate Java choices. Double-click an item in the list to insert it in the code.

5.4 Working with graphics

Including existing graphics into your Web design is simply a matter of importing graphics files into the Web project. Application Developer also supplies a WebArt Designer tool for creating your own graphics.
5.4.1 Importing graphic files

If your code will include links to graphics, you will need to make these accessible to Application Developer. If the graphics already exist on your system, you can simply import them into the Web project.

The first step is to create a folder for the images. Click File -> New -> Folder. The name and placement of the folder in the project should correspond to what the application expects.

Next, import the images by selecting File -> Import -> File System from the context menu. The wizard allows you to browse for and select the images to import.

Deselect the Create complete folder structure checkbox if you want the files copied directly into the new folder. Otherwise, the original file structure will be created under the new folder. Click Finish to import the files. The graphic images are now available to the application.
5.4.2 WebArt Designer

The following file types are associated with the WebArt Designer editor by default:

- .bmp
- .gif
- .jpe
- .jpg
- .jpeg
- .mif
- .png

Double-clicking a file with one of these extensions in the Navigator window or creating a file (by clicking **File -> New -> Simple**) with one of these extensions will automatically open the file in a new window with the WebArt Designer (Figure 5-8).

![WebArt Designer](image.png)

Figure 5-8  WebArt Designer

The WebArt Designer provides the tools to create or modify graphical images and to save them. You can choose objects to add from the gallery objects, or from a file of your own. You can also create your own graphics by using the tools.

You can also access the AnimatedGif Designer from here (click **Tools -> AnimatedGif Designer**), which allows you to create or edit animated objects. It provides a wizard to help you get started and the tools to assist you in your design.
Once you have opened the WebArt Designer or AnimatedGif Designer, you have access to help files for that editor.

5.5 Testing your Web project

If you are not using automatic build, you need to perform a manual build from the project context menu.

If you are not using automatic validation, you should do a manual validate from the Web project context menu. The WAR validator will check the following:

- Deployment descriptor (web.xml)
- Servlets
- Security roles
- Servlet and servlet mappings
- All JSP files and HTML (for broken links)
- EJB references

Testing resources in a Web project can be done as individual elements, as a Web project, or as a part of the EAR project. When using WebSphere Application Server, the EAR project is the base element associated with the server. You can use the workbench WebSphere Application Server environment with no Server project setup, or you can define a test environment.

To test, select the file (HTML, jsp, or servlet), the Web project, or the EAR project and select Run on a Server from the context menu. The test environment will be launched, the project published to the server, and you will be switched to the Server perspective, shown in Figure 5-9. A Web browser window will open with the output. Messages from the application server are shown in the Console panel.
Welcome to Web sample. Please choose from the following menu:

- Go to the hog farm display.
- Go shopping for llama goods.
- Test drive a donkey.

Figure 5-9  Testing Web projects
Chapter 6. Working with EJB projects

With the J2EE specification, enterprise beans are packaged in EJB modules. Application Developer provides the means to edit and deploy new or imported enterprise beans. This includes:

- Adding methods to the home and remote interfaces.
- Adding custom finders where necessary.
- Adding, defining, and mapping additional required CMP fields.
- Selecting and enabling the code validators for your project resources.
- Setting the EJB deployment descriptor (if default properties are not sufficient).
- Generating deployment code for your enterprise beans.
- Creating EJB access beans and using them to create your client application.
- Testing the enterprise beans.

EJB development is done from the J2EE perspective, which provides structural support and icons to make working with EJB modules easy. It supports session beans, CMP entity beans, and BMP entity beans.

6.1 Creating enterprise beans

EJB projects contain one or more enterprise beans. Creating an EJB project is similar to creating any other type of project (File -> New -> EJBPProject). The EJB project must be associated with an EAR file at creation. If you are not already in the J2EE perspective, creating an EJB project will switch you there.
The new EJB project will be listed under the EJB modules in the navigator view. You will also see a new JAR file for the EJB project under the EAR project you selected.

You can create new enterprise beans using the Enterprise Java Bean wizard or you can import an existing enterprise bean into your project.

Using the wizard makes enterprise bean creation simple. In Figure 6-1 you can see the wizard window we used to create a session bean called Sample. The wizard automatically enters the bean class name for you based on this name. In our case, it is called SampleBean. The only other field we needed to enter was the bean class package. The wizard will automatically fill in the home interface package and remote interface package fields with the same package name.

![Create EJB Wizard](image)

**Figure 6-1  Create EJB wizard**

The navigator view shows you the enterprise bean and interfaces created by the wizard under the EJB project. The required methods are also automatically created.
6.2 Editing enterprise beans

There are two editors available for working with enterprise beans: the EJB Editor and the EJB Extensions Editor. When working with these editors, code verification automatically occurs any time a change is made to the source code or properties.

To begin working with enterprise beans in an EJB module, double-click the EJB module. The EJB Editor (shown in Figure 6-2) will open the ejb-xml.jar descriptor file. The editor has multiple tabs, each representing an aspect of the enterprise bean: general, bean, security, transaction, environment, EJB reference, EJB resource reference, and source.

![Figure 6-2 EJB Editor](image)
In the Bean tab, you have the enterprise beans in this EJB module on the left. On the right, you have information about the currently selected bean, including the bean and interface names. Editing the source for these can be done by clicking the appropriate Edit button. Persistence fields are listed in the Attributes box.

To add methods to the home interface, including custom finder helpers for CMP entity beans, you would click the Edit button next to the Home interface field.

**Note:** Both EJB and SQL query languages are supported for custom finder helpers, but EJB is the recommended language. The implementation can be done with deployment descriptor extension documents (recommended) or with a finder helper interface for existing EJB 1.0 JAR files.

You can add methods to the remote interface without updating it directly by creating the method in the bean class and then promoting it to the remote interface by selecting **Enterprise Bean -> Promote to Remote Interface** in the Outline view.

Opening an EJB module with the EJB Extension editor gives you more advanced options, including allowing you to:

- Define access intent, isolation level, and security identity settings for methods.
- Add, modify, and delete relationships (associations).
- Define finder-descriptors.
- Define bean cache and local transaction settings.
- Set JNDI name and datasource access permissions.

### 6.3 Mapping enterprise beans to database tables

When you create an enterprise bean, some CMP fields are automatically created. These fields appear on the Bean tab in the attributes box. You can add or delete fields from this list.

These fields need to be mapped to a database schema. Generally, there will be a separate database table for each entity bean. Persistence fields in the bean will map to a column in the table. How this is done depends on the approach you take:

- When the enterprise bean is built before the database, you can generate a schema and map from the enterprise bean, then export that schema to the database. This is called top-down mapping.
When the database tables already exist, you can import the schema from the database, then generate the enterprise beans and mappings. This is called bottom-up mapping.

When both the enterprise bean and the database are built independently, you can create the schema by importing the tables from the database. Once the enterprise beans are created, map fields to tables in the schema manually. This is called meet-in-the-middle mapping.

The EJB development environment provides the EJB-RDB mapping tool to help you map persistence fields to databases. To use this tool, select **Generate for enterprise bean(s) -> Create EJB to RDB Mapping** from the context menu of the EJB project. For top-down mapping, this tool creates a schema and mapping. For bottom-up mapping, it creates an enterprise bean and mapping. For meet-in-the-middle mapping you manually create the mappings using existing enterprise beans and database schema.

![EJB-RDB mapping](image)

**Figure 6-3   EJB-RDB mapping**

Figure 6-3 shows the EJB-RDB mapping tool. You can see the CMP fields in the left pane and the database tables in the right. To create a mapping, simply select the fields in each pane and select **Create Mapping** from the context menu.
6.4 Creating Access beans

The Create Access Bean wizard is available (click File -> New -> Access Bean) to create and customize access beans for your enterprise beans. The wizard supports the following four types of access beans:

- Java bean wrappers
- Copy helpers
- EJB factories
- Data classes

New applications should use EJB factories or EJB data classes access beans. The EJB development environment automatically verifies that access beans are constructed correctly and that they are consistent with their associated enterprise beans.

6.5 Validating enterprise beans

Validating your code is always a good check point before generating deployment code. There are two validators available for EJB projects. The EJB Validator verifies that enterprise beans contained in an EJB project comply with the Sun Enterprise JavaBeans Specification, V1.1, Final Release. The Map Validator checks all maps in the EJB project.

These validators are selected in the EJB project properties. You can choose to have the validators run automatically whenever a resource is changed by selecting the option for automatic rebuild after changes.

Whether you use automatic validation or not, you can also validate the code manually from the project context menu. First make sure you have run a build on the project.

6.6 Generating the deployment code

As you proceed through the development process, the deployment descriptors are automatically updated. Before generating the deployment code for an EJB project you will need to build the project.

Once the build for the EJB project has been done, EJB deployment code can be generated using the EJB Deploy Tool. The tool does the following:

- Imports code from the input JAR file
- Generates deployment code
- Runs RMIC
Chapter 6. Working with EJB projects

6.7 Testing enterprise beans

Now that the deployment code has been generated and the EAR project rebuilt, it is time to test your enterprise beans. Enterprise beans or EJB projects can be tested on local or remote application servers defined using server projects. Testing using the WebSphere Application Server Single Server test environment included with WSAD is a good starting point. It is simple to do. No server project setup is necessary. Simply select the enterprise bean or EJB project you want to test and select Run on Server from the context menu. The EAR project will be published to the server and the server will be started.

6.7.1 EJB Test Client

Application Developer provides a Web-based EJB test client for testing enterprise beans. The test client provides the tools to find the EJB bean on the server, call the bean, view public fields of methods, filter the methods displayed, and change the property settings. When testing the enterprise bean, you can pass parameters to the methods and view the results.

**JNDI explorer**

The first step in testing an enterprise bean is to locate it on your server. The test client has a JNDI explorer (shown in Figure 6-4) to help you do this, either by typing in the JNDI name directly and using the lookup function or by selecting it from the namespace representation below the lookup function.
Figure 6-4  EJB Test Client: JNDI lookup

Once you have located the bean and selected it, the EJB home is loaded into the EJB page.

EJB page
The EJB page consists of three panes:

- References, where all EJB objects are listed in groups by their EJB home interface, non-EJB objects are displayed as object references, and loaded classes are displayed as class references. New classes can be loaded into the pane.
- Parameters, where the parameters for the selected method in the References pane are displayed.
- Results, where the results of the last method invocation are shown. If there are no results, but the method executed successfully, a message indicating a successful completion is shown. If there is a value in the Results pane, clicking Work with Object allows you to place the object into the References pane.
Selecting a method in the References pane will bring up the parameters for the method in the Parameters pane and a button to invoke the method.

Invoking the method will place the results in the Results pane. From there you can choose to work with any objects that have been created. Selecting the **Work with Object** button will put the object in the References pane. At this point you can select the new object (as you selected the method previously), pass values to it, invoke it, and review the results.
Working with Java projects

Java projects are used to create Java packages. These packages can be exported as JAR files and included in the build path for other project types, or in the classpath for application servers.

When you choose to create a new Java project with the Application Developer, the environment is set up for Java development. A Java builder is associated with the project so the Java source can be compiled as it is updated. The Java project contains information about the type hierarchy and Java elements, which kept current as changes are made.

When the project is created, the Workbench automatically opens to the Java perspective. The default views in this perspective include a Packages view, the editing area, a Hierarchy view, Outline view, Tasks view, Console, and Search.

Two other perspectives to note are:

- Type Hierarchy perspective: This perspective is used to explore the type hierarchy of Java elements.
- Debug perspective: This perspective is used for debugging Java programs. It has the following views: Process, Debug, Breakpoint, inspector, Variables, Display, Outline and Console.
7.1 Creating a new Java project

When you create a new Java project (by clicking File -> New -> Project -> Java -> Java project) you can specify the following:

- Name of the project.
- The workspace location to use.
- File organization.
  You can choose to have all the packages stored in the project folder (both class and Java source files) or you can choose to use separate folders under the project folder to organize the packages into groups.
- The build output folder.
- Projects and libraries that need to be in the build classpath.

When the Java project is created you are automatically switched to the Java perspective, shown in Figure 7-1.

![Java perspective](image)

Figure 7-1 Java perspective
7.2 Creating Java elements

The next step is to add the Java elements to the Java project. As you build your source code, a Java builder will automatically compile the source to create the class files. Depending on the file organization you chose when creating the project, the class files are either stored along with the Java files in the project folder, or if using source folders, the class files are stored in a separate output folder. The builder will also copy non-Java resources (for example, GIF files) into the output folder. The default output folder is `Java_project_name\bin`.

**Packages**
New packages can be created by selecting **New -> Package** from the context menu of the Java project.

**Classes**
Classes can be added to the package by selecting **New -> Class** from the context menu of the package.

The views in the Java perspective automatically change to accommodate the new class. The Java source is displayed in the editing area and the outline is updated to show the package and class.

**Methods**
Methods are entered directly into the editing area for the class. As you are typing you will get assistance from the editor in indentation, color changes for certain types of text (comments, strings), etc.
The Outline view will also be automatically updated. The new elements are automatically added. An icon to the left of each element indicates the type of element (method, class, interface, field, etc.). Filtering is available in the outline to limit what you see.

Execution arguments needed by applications using the method public static void main(String[] args) can be entered as class properties in the window shown in Figure 7-3. The class properties are accessed from the context menu for the class.
7.3 Specifying alternate JREs

One of the nice features of the Application Developer is the ability to select a different Java runtime. You can define multiple JREs to the Application Developer and specify one of them to use as a default. If you want to use a different JRE for a particular project, you then have the option of customizing the JRE properties of the Java project package to use a JRE other than the default.

Adding a JRE so it is available for use is done from the Java perspective by selecting Window -> Preferences -> Java -> Installed JREs. This will open the Installed Java Runtime Environments window, shown in Figure 7-4.

![Figure 7-4 Selecting the default JRE](image)

From here you add a new JRE by clicking Add. The JRE with the check mark will be the default used to build and run Java programs.

If you want to use a JRE other than the default to launch a Java project, you can go to the Java project JRE properties (using the context menu) and customize it to use the alternate JRE.
Note: This does not change the way the program is compiled. This is determined by the build classpath.

7.4 Testing Java projects

Java programs and projects can be tested by launching them from the Java perspective. They can be launched in either run or debug mode, depending on the launch icon selected. In run mode, the program executes. In debug mode, the execution can be suspended and resumed in order to see variables and how expressions are evaluated.

The first time you launch a program, you will be asked to specify a launcher for the project.

![Run dialog]

Once you make your selections, the application will be launched and you will be automatically put in the Debug perspective.
The Debug perspective gives you everything you need to examine how your program executes. Views in the Debug perspective allow you to work with and view aspects of the execution.

If the output is not as you expected, breakpoints provide a way of suspending execution at particular points in the code, allowing you to inspect the conditions and variables at that point. Breakpoints can be set by double-clicking in the editing area margin to the left of the code where you want to stop. Run the program by clicking **Debug** (instead of Run). The breakpoints will be shown in the Breakpoints view where you can delete, add, enable, disable, and otherwise work with them. When a breakpoint is reached you can use the Inspect option to have the expression evaluated and the results placed in the Inspector view. Click the **Resume** button (the left button at the top of the Debug view) to continue.
Chapter 8. Working with Web services

Web services is a set of related applications that can be invoked programmatically over the Internet. Service providers make Web services available by deploying them and then registering them to Web services registries. Clients can search for services in these registries and then invoke the service from the provider.

Web services provide self-contained business functions that can be used individually to provide a service. They could also be combined with other services to form a more complex business process. They should be designed in a way that makes them easily integrated with other processes. For example, a service could be something that provides information, such as weather reports or stock quotations. Or it could be a reservations system.

Web services relies heavily on several key architectural standards:

- Universal Description Discovery and Integration for Java (UDDI4J), defines a way to publish and discover information about Web services. Service providers that host a Web service register the service to a global UDDI registry, providing a Web Services Description Language (WSDL) descriptor to describe the service and its interface. Clients search for services in the UDDI registry. When they find a service, the descriptor tells them how to bind to the service to execute it.

- Simple Object Access Protocol (SOAP) is an XML-based standard for messaging over HTTP and other Internet protocols. It is a lightweight protocol using for the exchange of information in a decentralized, distributed
environment and it consists of three parts. SOAP is used to query the UDDI registry for Web services and enables the binding and usage of the Web services by defining a message path for routing messages.

- Web Services Description Language (WSDL) is an XML-based open specification that describes the interfaces to and instances of Web services. To make a Web service available, the service provider publishes a WSDL for the service to a service broker’s UDDI registry.

A service broker provides the UDDI registry. It registers and categorizes services and provide search services.

A Web service provider hosts one or more Web services. The service provider is responsible for:

- Creating, testing, and deploying the Web service
- Making the Web service available
- Publishing the Web service to a UDDI registry

The service clients use the service broker to find the needed service, then bind to and call the service provider.
Application Developer provides a full range of functions for developing, deploying, searching for, and registering Web services. We will take a look at some of them next.

8.1 Searching UDDI registries for Web services

The Application Developer provides a service requester client, giving you the ability to search for Web services using a UDDI registry. The client is in the IBM UDDI Explorer, which is included with the product. The IBM Test Registry is defined to the UDDI Explorer, allowing you to experiment with UDDI registries and to test your Web services. Once you have registered with other registries, you can add them to the UDDI Explorer.

One useful thing you can do with this is search for a service and import it into a Web project. This allows you to use the imported Web service as a skeleton or sample for creating your own.

Searching the registry and importing a service is easy. Select File -> Import, then select UDDI from the list of import source types, specify the Web project you want to import the service into, and click Finish. The IBM UDDI Explorer will open.

![IBM UDDI Explorer](image)

Figure 8-2 IBM UDDI Explorer
Using the search facilities, you can find services by specifying search criteria such as business or service name. When you find the one you want to import, click the import icon and the services will be imported to the Web project you specified.

**Note:** You can search the registry without a user ID and password, but in order to register a business or service type you will need to register and obtain a user ID and password.

### 8.2 Creating a Web service

Existing Java beans, URLs, and DADX files that reside in a Web project can be transformed into Web services by using the Web services wizard. The wizard (**File -> New -> Other -> Web services -> Web service**) is a series of windows that you use to specify the settings to use when creating the service. The defaults for the settings will be sufficient for most situations, making this a fairly painless task. The types of settings you can specify or change include:

- The Web project.
- SOAP descriptor elements:
  - Web service URN (Uniform Resource Name) used to route the request when it arrives.
  - The scope indicating the lifetime of the instantiation of the implementing class. Request indicates the object is removed after the request completes. Session indicates the object lasts for the current lifetime of the HTTP session. Application indicates the object lasts until the servlet that is servicing the request, is terminated.
  - Whether the methods are static or dynamic.
- The location of the descriptor files it will create (see Figure 8-3).
The methods to deploy and the encoding (SOAP or literal XML) for the input and output (see Figure 8-4).
Java to XML mapping customization (Figure 8-5). You have the option to display the mappings that will be used and customize them if necessary.
Java proxy settings (Figure 8-6). The generated Java bean proxy provides a remote procedure call interface to the Web service for use in testing.

XML to Java proxy mapping customization (Figure 8-7). As with the Java to XML mappings, you can choose to display and customize the XML to Java proxy mappings.
Sample Web application generation settings (Figure 8-8). The sample Web application demonstrates how to code the proxy file and is used to test the Web service.
The wizard takes the bean and wraps it as SOAP or HTTP GET/POST accessible services and builds the necessary descriptor files (Figure 8-9).

The wizard creates all the necessary files for the Web service in the Web project folders.
8.3 Testing a Web service

As you can see, when you create a Web service you are given the opportunity to create a sample application to test the Web service. You can choose to launch the Web service for testing in the wizard, automatically switching you to the Server perspective and launching the application.

If you did not go through the process of creating the Web service (for example, you imported an existing Web service) or if you did not opt to create the Java bean proxy and sample when you created a service, you can always do this later.

Application Developer makes it easy to test imported Web services by creating a Java bean proxy and sample application. You can do this when you create a Web service from an existing artifact by selecting the appropriate options, or you can do this later by using the Web service Client wizard. Application Developer uses the service WSDL document to create the required elements.
Figure 8-10  Web service Server test

The application is launched with the URL for the TestClient.jsp (in the samples folder).

8.4  Adding Web services to a UDDI registry

UDDI registries require that specific data be supplied to identify a service provider and its services. Application Developer provides you with the means to create these data structures within the registry.
Define your business to the registry
Before you can publish Web services to a registry, you will have to create an identity, or business entity, for your company in the registry. You will only need to do this once. After that, when publishing Web services, you will associate them with this entity. The business entity will contain information about your company.

Application Server provides a simple way to interface with the UDDI registry to create this entity. Select File -> Export, and select UDDI from the list of source types. The IBM UDDI Explorer will open. Since you are going to update the registry, you will need to select the registry and log in to it. Once you have done this, you can use the Publish Business Entity icon to open a wizard that allows you to enter the required information. This information will include the business name, phone number or other contact information, and the business category.

Publishing the Web service to the registry
After you have created a Web service and registered your company with a UDDI registry, the next step is to publish your Web service. This makes the information about your service available to clients searching the UDDI registry.

This is also done using export. Select File -> Export, and select UDDI from the list of source types. You will then be prompted to enter the WSDL service document you want to publish. The IBM UDDI Explorer will open. Since publishing requires a registry update, you will need to select the registry and log in to it.

An icon is available to publish the service. You will need to enter a description of the service and the categories it fits.
Chapter 9. Working with XML

Application Developer has different types of XML tools to support application development. The tools allow you to create, edit, and test XML files, create stylesheets to map one format to another and transform XML files using these stylesheets, generate Java beans from XML DTD or schema files, transfer data between DB2 and XML, and many other useful functions.

XML documents can stand alone or be associated with a DTD file or an XML schema. DTD / XML can be public, private, external, or included in the XML.

The relationship between an XML file and a DTD means that you can create a basic XML document from a DTD. The DTD provides enough information to build the XML structure, leaving you the work of filling in the attribute values.

Each type of XML document is unique, and as such has its own editor. Each editor is designed to give you the most useful features available for working with its particular file type, while recognizing and taking advantage of the relationship to the other file types.

Working with XML is done from the XML perspective. XML editors are visual tools for creating, editing and viewing XML files. Editor will help you create XML files from scratch, from an existing DTD, or from an existing XML schema. The XML editor helps to import existing XML files for structured viewing. You can associate your XML file with a DTD file or an XML schema from the editor.
9.1 Creating XML files

XML files are created using the Create XML File wizard (click File -> New -> Other -> XML -> XML File). When you create the XML file you can do it in one of three ways:

- From a DTD file
- From an XML schema file
- From scratch

The first two options are very convenient if you have an existing rules file that details how the new XML file should look. By letting the XML file be created based on these rules, you automatically get a coded skeleton for the file. The option to create XML files from these file types is also available from the context menu of DTD and XML schema files.

XML files are placed in an existing folder under any project type. If you plan to create Java beans from your DTD files then this should be a Java project.

Creating an XML file will automatically switch you to the XML perspective and open the file with the XML editor. There are two views in the XML editor: the Source view (shown in Figure 9-1) and the Design view (shown in Figure 9-2).

![Figure 9-1 XML editor Source view](image)

68 An Introduction to IBM WebSphere Studio Application Developer
The Source view of the XML editor displays the XML code as written. It has many text editing features useful for working with XML, including syntax highlighting, content assist, user-defined macros, code selection indicator, and unlimited undo and redo of changes.

If there is an associated DTD or schema, the content assist will use information from it to provide helpful suggestions for code at the cursor location.

![Figure 9-2 XML perspective with Design view](image)

The Design view represents the XML document both as a table and tree, making navigation and editing easier. Content and attribute values can be edited directly in the table cells, while pop-up menus on the tree nodes give alternatives that are valid for that location.

If there is a DTD associated with the XML file, you can validate the XML against it using the Validate icon. The content assist editing feature will also be aware of the DTD rules.
9.2 Creating DTD files

A Document Type Definition (DTD) contains declarations that define elements, attributes, notations and entities for a particular XML document. It establishes constraints for how these are used within that particular XML document. When creating a DTD, you specify the rules that control the structure of a particular XML document. These rules can be used to validate a particular XML document. XML parsers are there to validate the contents of the XML document against the DTD. These parsers will check lines in the XML document to ensure that it conforms to the rules set in the DTD for that particular XML document. If there is any deviation in the rule, the parser will generate the error and point to where the error occurs.

XML files are created using the Create DTD File wizard (File -> New -> Other -> XML -> DTD File). When you create the file you can do it in one of two ways:

- From an XML file
- From scratch

Creating a DTD file will automatically switch you to the XML perspective and open the file with the DTD editor. There are two views in the DTD editor: Design and Source (Figure 9-3).
The DTD editor has both the Design and Source views, similar to the XML editor allowing you to create and edit DTDs.

From the context menu of the DTD in the Navigator view you can generate the following from the DTD:

- An XML schema file
- Java beans
- An HTML form
- An XML file

### 9.3 Relating an XML file to a DTD file

If you create an XML file and DTD file independently, you can define a relationship between them by adding a DOCTYPE statement in the XML file that points to the DTD file directly or points to a DTD catalog entry.

Figure 9-4 shows a direct pointer to the DTD file. Any change in the DTD file location will require a change in any XML files using it. Using a catalog entry eliminates this problem. With catalog entries, you only have to change the catalog and all the XML files will still be valid.

![Figure 9-4 Relating the XML file to the DTD file](image-url)
Once the relation is made between the DTD and the XML document, the Reload Dependencies icon can be used to make sure the editor knows the constraints for the XML file. The Validate icon can be used to make sure the XML file follows the rules in the DTD file. From this point on, any input will be checked against the DTD to make sure it conforms. You can turn off this checking if desired.

The content assist menu in the XML editor is also now aware of the DTD rules and will show the valid elements for a cursor location based on the DTD rules.

To add a new element to the XML file, you will need to update the DTD file first so the element is considered valid. To properly add a new element:

- Open the DTD file in the editor.
- From the Design view, add the new element.
- Add the new element to the content model and save the DTD file.

In Figure 9-5 you can see the placement of the new element “leader” in the Outline view.

- Switch back to the XML file and reload the dependencies.

Now, you will be able to add the new element to the XML file. The content assist window will also be aware of the new element.
9.4 Creating XML schema

XML schemas are in the recommendation phase of W3C as a replacement for DTD. XML schemas are an XML language used to describe and constrain an XML file. An XML schema is functionally equivalent to a DTD, but written in XML. An XML schema provides extended functionality such as data typing, inheritance and presentation rules.

New XML schema files can be created from scratch using the Create XML schema wizard (File -> New -> Other -> XML -> XML schema). You can also generate XML schema from the context menu of a DTD file in the Navigator view. Creating an XML schema file will automatically switch you to the XML perspective and open the file with the XML schema editor.

![XML schema editor](image)

Figure 9-6  XML schema editor

The XML schema editor (shown in Figure 9-6) is much like the others we have seen for XML. There is a Design view and a Source view. The structure of the file in the outline will look similar to the DTD structure.
From the context menu of the XML schema file in the Navigator view you can generate the following:

- DDL
- A DTD file
- Java beans
- An XML file

### 9.5 Relating an XML file to an XML schema file

To create a relationship between an XML file and an XML schema (.xsd) file you will need to specify the file location or the public catalog ID in the XML file schemaLocation attribute.

```xml
<x:Redbooks xmlns:x=""
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
s:i:schemaLocation=" Redbook1.xsd">
```

### 9.6 Using the XML catalog

The XML Catalog contains entries for DTDs and XML schema files. Instead of pointing the XML file directly to its related DTD or XML schema file, you can do this indirectly by using the XML catalog instead.

Adding entries to the XML catalog is simple. From an XML file, you can open the XML Catalog wizard by selecting **Windows -> Preferences -> XML Catalog**.

There are only two things you need to tell the XML catalog: the public identifier for the DTD or XML schema file and the location of the file. If the file is an XML schema file, the ID must match the schemaLocation attribute of the root element in the XML file. If the file is a DTD file, the ID must match the value of the DOCTYPE statement in the XML file.
For example, in Figure 9-7 we show a catalog entry for a DTD file with the ID of RedbookCatalogEntry. For an XML file to use this DTD it would need the following statement:

```xml
<!DOCTYPE Redbooks PUBLIC "RedbookCatalogEntry" "RedbookAlternate.dtd" >
```

If, for some reason, the Redbook.dtd file isn't found, the XML file will use the DTD file called RedbookAlternate.dtd.
9.7 XML-to-XML mapping

XML-to-XML mapping allows you to take one or more source XML documents, either XML files or DTDs, and map the elements in them to a single target XML document, creating a new document with the results. This can be used, for example, when you have a new format for XML files and need to convert the existing files. Or if you have data in XML format that you want to display as HTML, the XML-to-XML mapping will do this for you.

The XML mapping editor is easy to use. To create the mappings you simply use the XML-to-XML mapping wizard to choose the source and target files. The files are placed side by side in the editor. You click a pair of elements to map, one in the source and one in the target, and click the Create New Mapping icon. The resulting mapping structure appears in the Composition view. The mappings are saved as a .xmx file.

![XML-to-XML mapping editor](image)

Figure 9-8  XML-to-XML mapping editor

The desired result of this mapping is to be able to produce a script that can take one XML document and transform it into a different structure.
When mapping two XML documents, you will often encounter the situation where the source and target elements are not a one-for-one match. A common occurrence of this situation is when the values specified in a source element may need to be split among multiple target elements. For example, you may have an element called Name in the source document, but the target has two elements called First Name and Last Name. The mapping between the two would need to take the text string in Name and divide it into two substrings delineated by the space between the names. The first substring would need to be mapped to First Name and the second substring to Last Name.

To accommodate this, the editor lets you define ways to manipulate the values in the source documents when the translation from original files to new files is performed. You do this by defining an XSL function for the target element. XSL functions define actions to be taken during translation to transform the original values to the values required for the new file. An XSL function can look at the value for an element in the source and determine what mapping actions to take based on that value. XSL functions are aware of different data types and can perform the appropriate functions depending on the type. For example, if you specify that the data is a text string, you can locate substrings, concatenate text, etc. If the data is numeric you can perform numeric operations on it. You could also use a Java bean to retrieve data, compute values, or perform some other function to transform the value for the target file.

9.7.1 Transforming the files from the old format to the new

Once the mapping is complete, the next step is to generate the XSLT (eXtensible Stylesheet Language: Transformations) script that will be used to transform XML files from the source structure to the target structure. This is done by selecting Mapping -> Generate XSLT Script. The resulting script will be saved with a .xsl file extension.

Once the stylesheet has been created, it can then be applied to the XML files that need to be transformed. To do this, select the source XML file and the XSLT script simultaneously in the Navigator (use the Ctrl key to select multiple files). Then select Apply XSL -> As XML or if transforming from XML to HTML, Apply XSL -> As HTML, from the context menu. The results are put in the XSL trace editor and can be saved as a new XML or HTML file from there.
From the XSL trace editor, you can step through the script, highlighting the rules as they are fired, viewing the results as they happen.

### 9.8 RDB-to-XML Mapping

The RDB (Relational Database)-to-XML mapping editor allows you to map relational database data to elements and attributes in an XML file. You can map between one or more relational tables and an XML document. The mapping information will be stored in a session file with an extension of .rmx. From this mapping information a Document Access Definition (DAD) script can be generated. The IBM DB2 XML Extender can use the DAD script to compose an XML document from existing DB2 data or to decompose an XML document into DB2 data.

To use the RDB-to-XML editor and to deploy DAD file, you must have installed IBM DB2 Universal Database and DB2 XML Extender Version 7.1 or higher.
9.9 SQL Query Builder

The SQL Query Builder allows you to construct SQL statements, execute them, and display the results. From the SQL query, you can generate an XML document, a DTD, and an XSL stylesheet. You can also deploy the query to WebSphere so you can dynamically generate XML.
Chapter 10. Working with databases

Application Developer provides you with the means to work with relational databases. Using the Data perspective, you can create connections to existing databases, and import databases, schemas and tables to the workbench. Or you can create and work with new databases. Once you have the database connection, you can create, edit and execute SQL queries.

The Data perspective offers two views specific to working with data:

- **DBA Explorer view** - Allows you to create a JDBC connection to existing databases. From this view, you can view the database design and import it to the local workbench.
- **Data view** - Allows you to select, update, insert, or delete data to database tables.

In order to work with a database design or schema locally, you need to create a Data project to hold the design or schema information.

### 10.1 Working with existing databases

First, let’s take a look at how you would work with an existing database. The Data Explorer view in the Data perspective provides the capability to connect to existing databases.
10.1.1 Connecting to a database

To work with a relational database, you first need to establish a JDBC connection. New connections are made in the Data Explorer view by selecting **New -> Connection** from the context menu.

![Create connection wizard](image)

The wizard uses the information you provide to create a connection to the database. The required information includes the database server hostname, database name, and database access user ID and password.

The WebSphere Studio development machine must have the JDBC drivers needed to connect to the database. You can enter the driver information or choose from one of the two pre-defined JDBC drivers:

- IBM DB2 App Driver (COM.ibm.db2.jdbc.app.DB2Driver) for connection to local databases or remote databases defined locally with the Client Configuration Assistant.
- IBM DB2 Net Driver (COM.ibm.db2.jdbc.net.DB2Driver) for connection to remote DB2 databases.

You can use the **Filters** button to specify tables you want to filter out of your view of the project. Your choices can be changed later using the Filter wizard.
When you click **Finish**, the wizard will establish a connection to the database and you will see the database design in the DB Explorer view.

![DB Explorer view-new connection](image)

This connection information will be saved so you can easily reconnect to the database.

### 10.1.2 Importing the database design

Once you are connected to the database, you can use the DB Explorer to browse the design of the database and to import the database design. Design elements (database, schema, or table) of the database can be imported selectively by choosing **Import to Local** from their context menu and identifying the target of the import. Databases are imported into project folders, schemas into databases, and tables into schemas.
You can work with the newly imported design by switching to the Data view. At this point, whether you are working with newly created or newly imported designs, the procedure is the same.

### 10.2 Working database designs

Once you have imported the design, switch to the Data view to work with it. If you are starting with a new design, start by creating a Database project and add a new database to it.
Whether you have a new or imported design, you can add or delete schema from the database context menu, tables from the schema context menu, or columns from the table context menu.

Any database element can be opened by double-clicking it. For example, opening a table will give you access to the column design. Columns can be added, deleted, or altered and primary and foreign keys can be chosen.

### 10.2.1 Building and executing SQL statements

Application Developer provides the SQL wizard and the SQL Query Builder tool to develop SQL statements designed to view or alter information in relational databases. The wizard can be opened by selecting **File -> New -> Other -> Data -> SQL Statement**. The Query Builder tool can be opened by selecting the type of SQL statement you want to create from the context menu of the Statements folder within a Database project.

Both tools provide a graphical interface to assist in building the statements. The statement type, database tables, columns and conditions are all selectable from the panels, making it easy to build SQL without having any knowledge of how it works.
Once the statement has been generated and is in the query builder, you can alter the statement by using drag and drop techniques and the tabs and options provided.

The statement can be executed by clicking the **SQL Run** button (circled in Figure 10-5). The results of select queries are displayed in table format. Otherwise a message will be displayed indicating the success or failure of the statement.

SQL DDL scripts can also be created and applied to databases.
The Performance Analyzer profiling tool analyzes runtime data to provide comprehensive information about the behavior of Java programs. It consists of views that present the data in a way that makes it easy to spot performance problems. Performance data can be gathered for programs executing locally, remotely, or distributed on different machines. This profiling tool can be used to analyze the performance of stand-alone Java applications, but more commonly, it is used to analyze Java code running in an application server.

The views in the Performance Analyzer are designed to help you:

- Identify time-consuming objects and methods
- Identify memory-intensive classes
- Find garbage collection problems
- Gauge the concurrency of a program
- Locate and determine the cause of memory leaks
- View an object or method
- Identify which threads are active, and when
- Identify frequently called methods
- Identify different phases of program execution
- View different invocations of a method
- View the caller of a method or object
11.1 IBM Agent Architecture

The profile data used for debugging and performance analysis is gathered and externalized using the IBM Agent Architecture. This architecture is implemented using the IBM Agent Controller, the IBM Java Profiling agent and the Default Logging Agent.

The Performance Analyzer acts as a client of the Agent Controller, using it to launch host processes. The Agent Controller resides on the same machine as the application process. For this reason, you will need to install and start the IBM Agent Controller (shipped with Application Developer) on each deployment server. The client (Performance Analyzer) can be local or remote from the Agent Controller.

When Performance Analyzer uses the Agent Controller to launch an application, a profiling agent is started to gather information about it. The agent sends the data back to the attached client through the Agent Controller.

11.2 Getting started

To begin the process you need to do a few things first:

1. From the Server perspective, open the server instance you will be testing on. Under the General tab, enable the profile server process. Under the System properties tab, add a property called java.compiler and give it the value of NONE.

2. Start the application server instance from the Servers view.

3. Look in the Console view to the first line to see the process ID of the WebSphere process.

4. Using the Profile icon at the top of the perspective, choose Attach to Java Process. In the attach to the application server process, select the agent under the process ID for the WebSphere server instance.

5. Navigate to the filters and make sure the classes for your application will show up. For example, if you are using the samples that come with Application Developer, create an “INCLUDE” filter for com.ibm.samples.* and move the filter to the top. Click Finish.

6. You will be switched to the Profiling perspective. Switch back to the Server perspective and navigate through your application, stopping just before the point you want to analyze.

7. Start a monitor by right clicking on the agent and selecting Start monitor.
8. Switch back to the Server view and continue running your application. When you want to see the performance data, switch to the Profile perspective. The data is gathered constantly, so you will need to use the Refresh icon to make sure you have current data.

11.3 Using the Profiling perspective

Performance analysis is done from the Profiling perspective. To begin a performance analysis, you launch an application using the Profile icon. From the icon you can choose to launch a local or remote Java application and the class you would like to analyze, or you can attach to a running agent. Launching an application will start the host process and the profiling agent on the chosen server.

Once the process is running, you can detach from it and leave it running. You can re-attach as long as it is still running, or another client could attach to it.

The Profiling perspective uses the following views to present the data in a form that can be easily interpreted:

- Class Statistics view
- Method Statistics view
- Heap view
- Execution Flow view
- Method Invocation view
- Method Execution view
- Object Reference view

These views offer features to assist you, such as the ability to sort columns, to include or exclude columns, filtering capabilities, and coordination among views.

11.3.1 Profiling projects view

The first time you begin the process of profiling a process, a Profiling project is created with the following structure:

- A monitor for the application to contain the profiling information collected by its agents.
- A node representing the server running the application
- A process for each process currently running.
- Agents attached to the process. An agent can be running, running and collecting data (monitoring), or stopped. You can start or stop monitoring running and detached agents. A profiling agent maps to the lifetime of the
process. When the application ends, the profiling agent stops. If you relaunch the process, a new instance of the agent is created.

11.4 Class and method statistics

Two views are used to present information on the basic statistics of class and method execution.

The Class Statistics view, shown in Figure 11-1, includes information about a class in terms of the package to which a class belongs, the number of instances of the class in the application, the number of instances of the class for which garbage collection occurred, the base time spent in that class, the cumulative time for the class, the size of the class, the number of calls made to the class, the parent of the class, the interfaces of the class, and the Java file that contains the class.

Figure 11-1   Class Statistics view

The Method Statistics view, shown in Figure 11-2, includes the package (and thereby class) to which the method belongs, the number of calls made to the method, the base time spent in the method, and the cumulative time spent in that method.
These views are a logical place to start with a performance analysis. Both views show the base and cumulative time that the objects executed. These values can be sorted to give you a clearer picture of execution times, indicating which classes and objects warrant further analysis. The Size column of the Class Statistics view is useful for identifying classes that use a large amount of memory. Memory leak problems can be detected by monitoring the garbage collection numbers shown in the Collected column of the Class Statistics view.

11.5 Execution flow

Studying the execution flow of a program helps you identify methods with large execution times, frequently called methods, and gives you some insight into garbage collection. Looking at active thread information can help you identify when and where most of the program resources are spent. Three related views are provided to give you a comprehensive picture of the flow of execution for a program.
The Execution Flow view, shown in Figure 11-3, is a good place to start analyzing the program flow. It gives you information on the execution flow for an entire program. The execution flow is presented as a time scale, showing you for each thread (including garbage collection) the time each method is called, the time spent executing that method, when the method returned, the name of the method, its arguments, and its return types. To see portions of the view better, you can zoom in on sections.

![Figure 11-3 Execution Flow view](image)

The Method Invocation view, shown in Figure 11-4, is similar to the Execution Flow view, but focuses on one method at a time, giving you more specific details about the execution.
Figure 11-4  Method Invocation view

The Method Execution view summarizes the information in the Method Invocation view, showing you the interactions that underlie an execution pattern.

Figure 11-5  Method Execution view
These three views are coordinated so that a selected object in one view will be the selected object in another view if you switch.

11.6 Execution statistics

The Heap view, shown in Figure 11-6, shows instances (objects mode) or methods (methods mode) grouped by class, and indicates their level of activity in the form of a histogram. Class names appear in the left frame with the objects or methods in the right. The histogram uses color to represent numerical values, making it easy to spot problems visually.

![Figure 11-6 Heap view](image)

The Heap view is related to the execution and method flow views and can be used alone or in coordination with these views. You can switch to the Method Invocation view or Method Execution view from the context menu of a method in the Heap view. If you have something selected in the Heap view (a method, a class object, or a class instance), the Execution Flow view will have the same object selected.
Information available for an object includes the reported-at time, base time, cumulative time, number of times it is called, and number of bytes it occupies. For a method, the details include the method’s parameters, its base time, its cumulative time, and the number of times it is called. The Heap view shows only one information category at a time, depending on your selection.

You select what criteria is displayed and how it is displayed by customizing the color scheme for representation of objects, methods, or classes. The first step of the customization is to select the criteria you want the colorization to represent. You can have the objects or methods in the view “colored by” one of the following criteria:

- Base time. Useful for identifying time-consuming tasks.
- Cumulative time. Useful for identifying time-consuming tasks.
- Number of calls.
- Number of threads. This is useful to identify methods or objects that are executed on a single or multiple concurrent threads. A program that has objects executing concurrently is generally more efficient.
- Memory size active. This can be used to identify memory-intensive classes.
- Memory size total. Also used to identify memory intensive classes.

The next step in the customization is to associate different colors using a sliding scale with numerical values. A legend of the chosen colors is displayed at the top of the view.

The net result of this customization is that the histogram will have colored symbols that indicate the numerical value for the selected criteria for each object or method.

In objects mode:

- Rectangles denote either instances of the class, or the amounts of memory consumed by instances of the class, depending on the current choice in the “color by” customization. For an array (which is a pseudo-class with the name [type], each rectangle represents an array of type.

- Diamonds denote the class objects for a given class. Objects for which garbage collection has occurred appear as rectangular, or diamond-shaped outlines; those for which it has not occurred appear as color-filled shapes. A large number of color-filled rectangles could indicate a memory leak.

In methods mode, each rectangle represents a method of the corresponding class displayed on the left.
11.6.1 Object reference information

The Object Reference view is useful for finding elusive memory leaks by showing references to or from an object. References to an object could prevent garbage collection for that object.
A repository is a persistent store that coordinates multi-user access to projects and team streams. Communication between a workbench client and the repository server is done using TCP/IP. The supported repository for the beta release of Application Developer is CVS (Concurrent Version System). In future releases of Application Developer, other repository plug-ins will be added.

CVS uses a parallel development model for version control allowing more than one developer to work on the same files at the same time. You can find more information on CVS at http://www.cvshome.com.

### 12.1 Working with CVS

Before you do any team development, you must first install CVS. The latest CVS version is available at http://www.cvsnt.com. Read the documentation that comes with the product to install and configure the CVS server. You must also decide what kind of protocol to use. Two types of protocols are supported: pserver and ssh (secure shell). The pserver protocol is a password-based approach.
Streams
Teams can share and integrate their ongoing work in streams. A stream is a shared work area that can be updated by team members. A shared stream allows each member to work on files individually, while accessing changes others are making. The stream effectively represents the current shared state of the project.

Synchronizing
Two distinct processes are involved in synchronizing resources: “catch up” and “release”. When you make changes in the workbench, resources are saved locally. For others to access a resource you have to release it to the stream. At the same time, others may have released changes to the stream. You can catch up with these changes in the workbench.

Conflicts arise when you have modified a resource for which a more recent version is available in the stream. In this situation, you can do one of the three things: catch up the resource from the stream, release your version of the resource to the stream, or merge your work and the stream resource.

Versioning resources
A version of a resource is a non-modifiable copy of the resource that has been released to the repository. The repository can hold multiple versions of the resource, while a stream contains only the currently released version. A base version of a resource is released to the stream at the time of synchronization. The base version denotes what the resource looked like before you started modifying it.

12.1.1 Working with team using CVS
To illustrate CVS, let’s take an example. We will assume that you have installed CVS and created a repository for the team environment. To use resources from the CVS repository with Application Developer, you begin by defining the repository location to the workbench.

1. Switch to the Team perspective. In the Repositories view, select New -> Repository Location from the context menu.
2. In the wizard:
   
a. Select the authentication protocol of the CVS server from the connection type drop-down menu (for example, pserver).
   
b. In the User Name field, type the user ID under which you want to connect to the repository.
   
c. In the Host Name field, type the host address of the machine.
   
d. In the Repository Path field, type the path to the repository on the host.
   
e. The CVS Location field is automatically updated with the connection information from the other fields in the window. You do not need to enter anything in this field.
   
f. Select **Validate location on finish** to test the connection to the server. Deselect this option if you want to authenticate later.
   
g. Click **Finish**. If you selected the option to validate the location, you will be prompted for a password. Enter the password and click **OK**.

Next, you will need to synchronize the resources in the workbench with the resources in the stream. To do this:

1. Select a resource (or resources) that you want to synchronize with the stream from the Navigator view.
2. From the context menu of resource, select **Team -> Synchronize with Stream**.

3. In the window that pops up, select the repository location and stream you want to use to synchronize your resources. You can create separate streams for your team.

![Set Project Sharing window](image)

**Figure 12-2  Set project sharing window**

4. Click **OK** and you will be automatically switched to the Team perspective where you will see the Synchronize view. There are three modes in which resources are synchronized:

   - Catch up mode: Shows incoming changes only (resources in the steam that differ from what is in the workbench).
   - Release mode: Shows outgoing changes only (resources that have been modified in the workbench only).
   - Catch up/Release mode: Shows both incoming and outgoing changes.

You can switch to any of these three modes by using the view’s toolbar buttons. To show only conflicts (resources edited in both workbench and stream), click the **Show only conflicts** button on the view’s toolbar from any of these three modes.
In Synchronize view, you can see the Structure Compare and Text Compare area. The resources displayed in the Structure Compare area depends on which mode you are using in the Synchronize view.

In the Text Compare area, the workbench resource is on the left side and the stream resource is on the right side.

![Figure 12-3 Synchronize view](image)

### 12.1.2 Catching up

You can catch up with the work that has been done by other team members and released to the stream since you last synchronized. To catch up workbench resources:

1. Select a resource in the Navigator view whose stream you want to catch up.
2. From the resource’s context menu, select **Team -> Synchronize with Stream**.
3. In the Synchronize view, select the Catch Up mode button on the view’s toolbar. In the Text Compare area, your local workbench data is represented
on the left and repository stream data is represented on the right. Click the
Show only conflicts icon to see the differences.

4. If there are no conflicts (indicated by a red arrow), you can catch up all the
resources in the workbench without losing any changes you have made in the
workbench.

If there is a conflict, use the Text Compare area to merge resources with
conflicts. You can copy changes from the repository version of the resource to
the workbench version of the resource and save the merged workbench
resource. Once all the conflicts have been resolved, you are ready to release
your changes to the stream.

5. In the Structure Compare area, select the top-most resource in the hierarchy
and select Catch Up from the context menu.

12.1.3 Releasing

When you have made changes to a resource, you need to release the resource
to the stream for use by other team members. To release a resource to stream:

1. Select the resource you want to release to the stream from the Navigator
View.

2. From the resource’s context menu, select Team -> Synchronize with
Stream.

3. In the Synchronize view, select the Release mode button on the view’s
toolbar. You can see the outgoing changes in the Text Compare area.

4. If there are no conflicts, you can release the resource to the stream.

If there is a conflict, use the Text Compare area to merge your changes with
others. You can copy changes from the repository version of the resource to
the workbench version of the resource and save the merged workbench
resource. Once all the conflicts have been resolved, you are ready to release.

5. In the Structure Compare area, select the top-most resource in the hierarchy
and select Release from the context menu, providing a comment for the
release.
12.1.4 Versioning the project

To create a version of the project in the repository, you must first have the version in the stream.

1. Select the project you want to version from the Navigator view.

2. Select Team -> Version from Workspace from the context menu for the project.

3. A window will pop up, allowing you to select from the following three options:
   - Automatic: each project's version label will be automatically generated.
   - One Name: the version label for each project is one name which is specified by the user.
   - Name Each: for each project, the user is prompted to enter the version label.

Click OK to finish.
Now switch to the Resource perspective and select the Repositories view. You can see the Repository location and resources. Under the Project Versions folder, you will see an entry for your project with its versions underneath.

12.2 For more information

For more information on CVS, see:

http://www.cvshome.com
Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.
The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

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

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

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

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

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.