WebSphere software

IBM

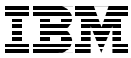
Self-Study Guide:
WebSphere Studio Application Developer and Web Services

- Teach yourself WebSphere Studio Application Developer
- Learn about Web Services
- Create and use Web Services by example

Ueli Wahli

ibm.com/redbooks

Redbooks



International Technical Support Organization

Self-Study Guide: WebSphere Studio Application Developer and Web Services

February 2002

SG24-6407-00

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 393.

First Edition (February 2002)

This edition applies to WebSphere Studio Application Developer Version 4 for use with the Windows 2000 and Windows NT Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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.

Contents

Preface

This IBM Redbook is a self-study guide for the new application development tool WebSphere Studio Application Developer and for Web Services.

WebSphere Studio Application Developer is the new IBM tool for Java development for client and server applications. It provides a Java integrated development environment (IDE) that is designed to provide rapid development for J2EE-based applications. It is well integrated with WebSphere Application Server Version 4 and provides a built-in single server that can be used for testing of J2EE applications.

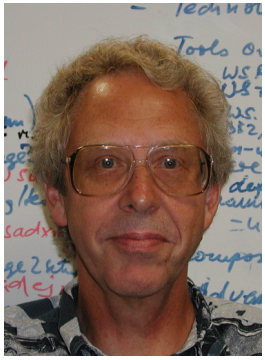
Web Services are a new breed of Web applications. Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services perform callable functions that can be anything from a simple request to complicated business processes. Once a Web Service is deployed and registered, other applications can discover and invoke the deployed service. The foundation for Web Services is based on the simple object access protocol (SOAP), the Web Services description language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) registry.

This redbook consists of two parts, a presentation guide and an exercise guide:

- The presentation guide explains the new tool and Web Services.
- The exercise guide provides detailed instructions to perform exercises using WebSphere Studio Application Developer. The sample code used for the exercises is available for download at the Redbooks Internet site. The sample code also includes the solutions that can be loaded and studied.

The team that wrote this redbook

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



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

Most of the content of this book is based on the redbook *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292, written by:

- **Mark Tomlinson**, IBM Application and Integration Middleware technical sales team, London, England
- **Olaf Zimmermann**, Consulting IT Architect at IBM Global Services, BIS e-business Integration Services, Heidelberg, Germany
- **Wouter Deruyck**, consultant for the EMEA AIM Partner Technical Enablement Team, La Hulpe, Belgium
- **Denise Hendriks**, Managing Director and WebSphere Architect with Perficient, Inc.

Special notice

This publication is intended to help Java developers create client and server applications on the WebSphere platform, including the creation and usage of Web Services. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Studio Application Developer. See the PUBLICATIONS section of the IBM Programming Announcement for 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 ®	Redbooks
Redbooks Logo 	AIX	AlphaWorks
CICS	DB2	IBM Global Network
IBM Registry	IMS	MQSeries
NetRexx	OS/390	S/390
WebSphere	VisualAge	z/OS

Comments welcome

Your comments are important to us!

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

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

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to the address on page ii.

Part 1

Presentations

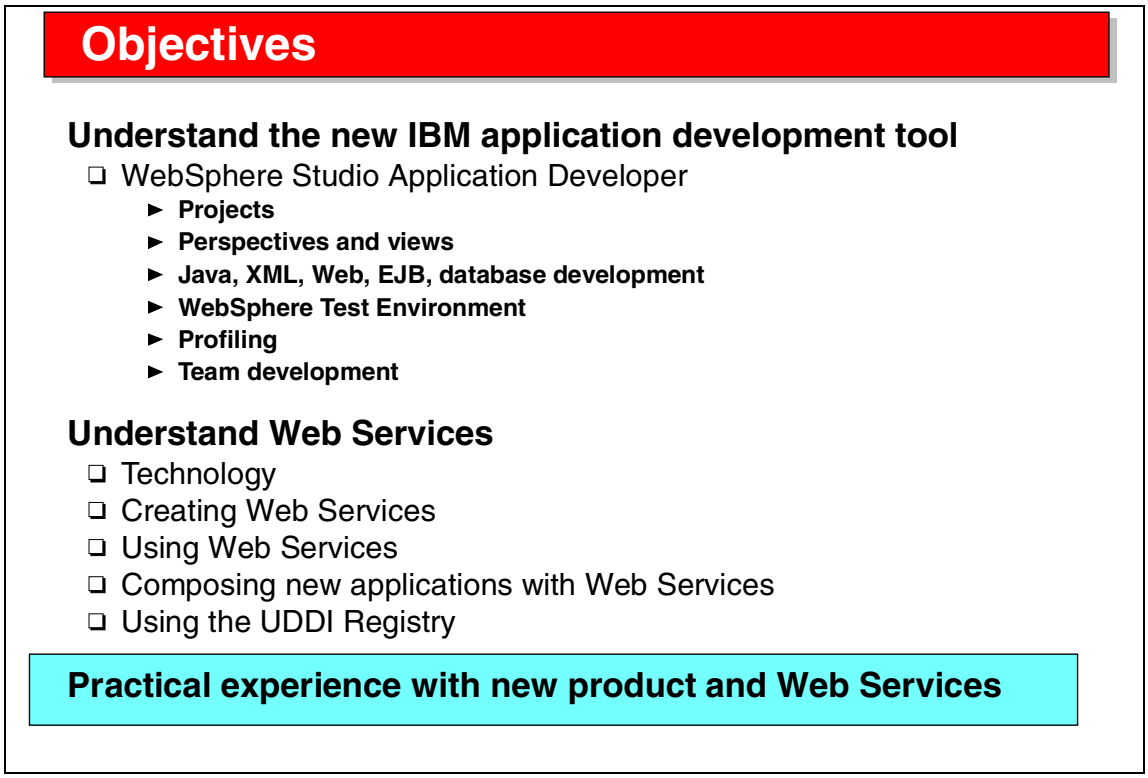
This presentation guide is structured into 14 units:

- A short introduction
- Nine units on WebSphere Studio Application Developer
- Four units on Web Services

1

Workshop Introduction

Visual 1-1 Title



Visual 1-2 Objectives

The objectives for this class are two-fold:

- Understand and work with the WebSphere Studio Application Developer
- Understand the new technology of Web Services

Most of the learning occurs by doing the practical lab exercises using the WebSphere Studio Application Developer and WebSphere Application Server Advanced Edition Single Server.

Prerequisites

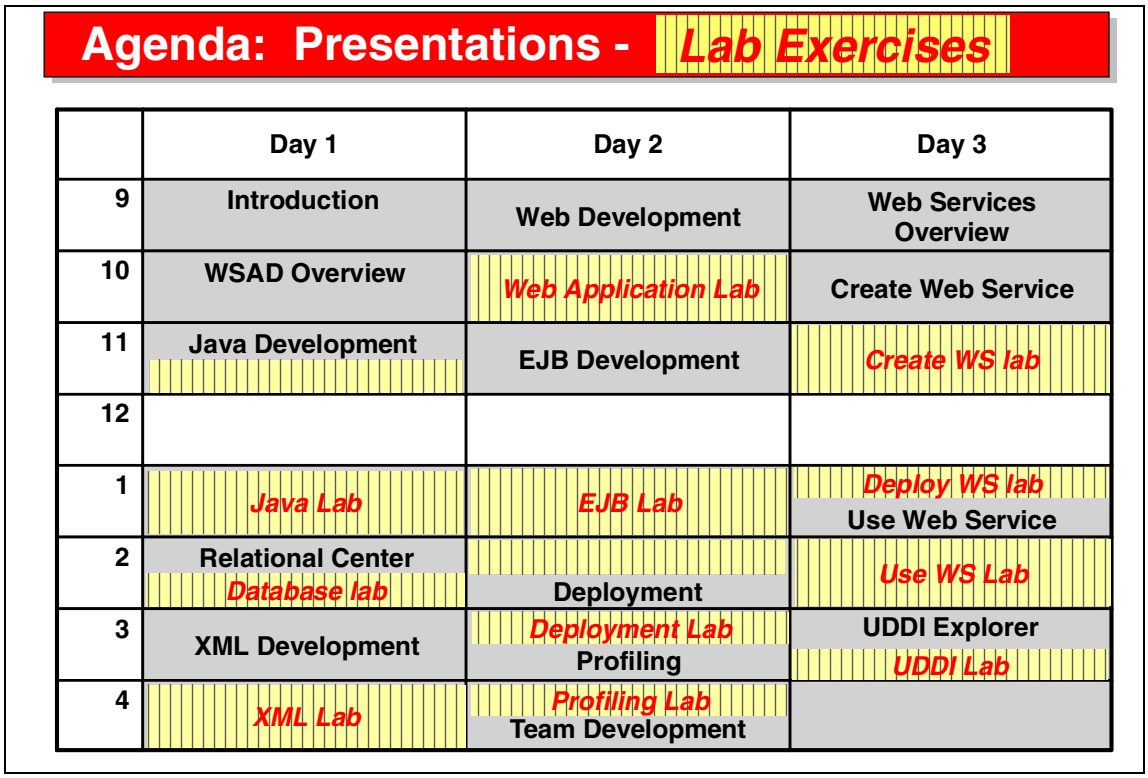
Prerequisites for this class

- Basic understanding of Web Server/Application Server concepts
- Practical experience with VisualAge for Java
  - **Understanding of JavaBeans**
  - **WebSphere Test Environment**
- Some experience with WebSphere Application Server
  - **Installing Web applications**
  - **Configuring an application server**
- Some knowledge of HTML, JSP, servlets, EJB
  - **Model-View-Controller pattern**

Visual 1-3 Prerequisites

Having extensive experience with existing IBM application development products is not a prerequisite for this class. A number of beginners have successfully gone through the class, although beginners may not get through the optional parts of the exercises in the allocated time frame.

Any experience with Java development tools helps, but the most important thing is that you have a basic understanding of servlets and JavaServer Pages (JSPs), because these are used in many of the sample applications.



Visual 1-4 Agenda

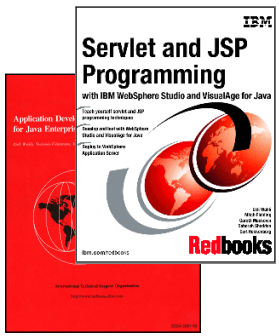
This is the agenda of the 3-day class, consisting of two days on the Application Developer tool, and one day on Web Services.

About half the time is spent on lab exercises.

ITSO Redbooks: source for more information

- International Technical Support Organization
- Homepage: ibm.com/redbooks

PDF files - Redpieces (drafts) - Sample code - Search



- **ITSO San Jose recent AD Redbooks:**
  - WebSphere V4 Appl. Dev. Handbook
  - Programming J2EE APIs with WebSphere
  - EJBs for z/OS and OS/390 CICS Trans. Server
  - EJB Development with VA Java for WebSphere
  - Programming with VA Java Version 3.5
  - Version 3.5 Self-Study Guide: VA Java/Studio
  - How about Version 3.5 of VA Java/Studio
  - Servlet/JSP/EJB Design and Implementation
  - Servlet and JSP Programming
  - XML Files (2 books)

Visual 1-5 ITSO Redbooks

IBM Redbooks are a great source for technical information on application development tools. Here is a list of recent redbooks:

SG24-6134	*WebSphere Version 4 Application Development Handbook*
SG24-6124	*Programming J2EE APIs with WebSphere Application Server*
SG24-6284	*Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server*
SG24-6144	*EJB Development with VisualAge for Java for WebSphere Application Server*
SG24-5264	*Programming with VisualAge for Java Version 3.5*
SG24-6136	*Version 2.5 Self-Study Guide: VisualAge for Java and WebSphere Studio*
SG24-6131	*How about Version 3.5: VisualAge for Java and WebSphere Studio Provide Great New Function*
SG24-5754	*Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server*
SG24-5755	*Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*

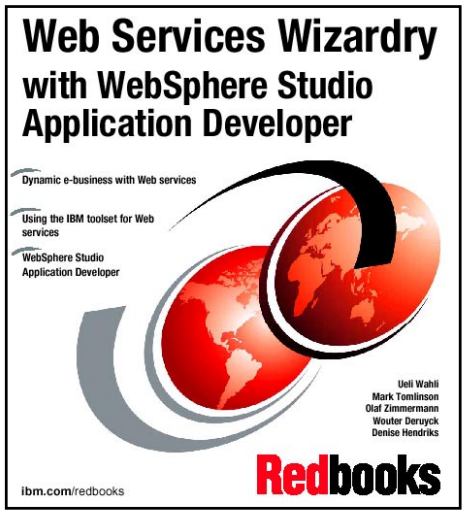
Web Services Redbook

SG24-6292 Web Services Wizardry with WebSphere Studio Application Developer

- Structure similar to workshop
- Comprehensive explanations of Web Services architecture
- More complicated examples
- Currently a Redpiece

Redpieces

http://www.redbooks.ibm.com
- **Redbooks Online**
- **Redpieces**



Visual 1-6 Web Services Redbook

A new IBM Redbook, *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292, is a companion book to this class.

This book covers in great detail many of the concepts that are discussed in this class. The examples in the redbook are more comprehensive than the simple examples that we cover in the class.

Summary

This is a very technical workshop with many hands-on labs

- Learning through practical experience

Hardware:

- Fast machines required
- Memory at least 384 MB, but 512 MB is better

Software:

- DB2 Version 7.2 (or 7.1 Fixpack 3)
- WebSphere Application Server Advanced Single Server
- WebSphere Studio Application Developer
- WebSphere UDDI Registry

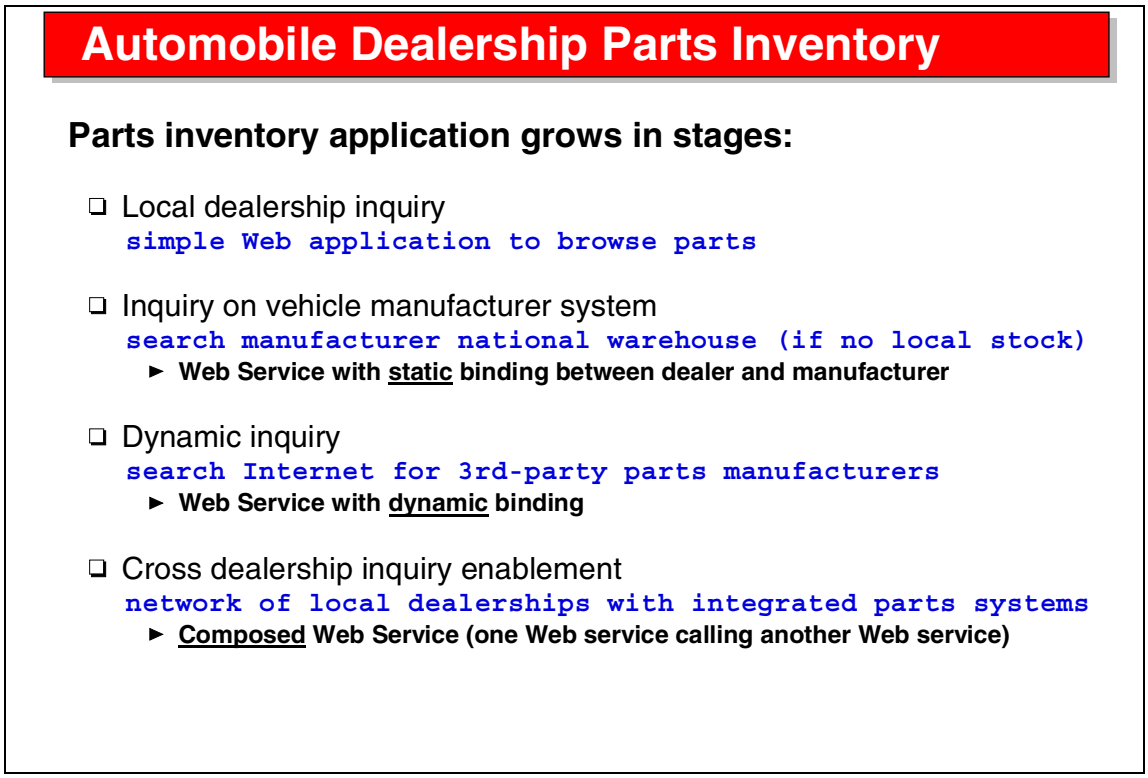
Evaluation forms must be turned in!

Visual 1-7 Summary

To successfully run the lab exercises of this class, you must have fast machines with ample memory. The memory is more important than the processor speed, especially when we run WebSphere Application Server and WebSphere Studio Application Developer at the same time.

Visual 1-8 Sample Application

The class is based on a sample application that is explained in detail in the redbook *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292.

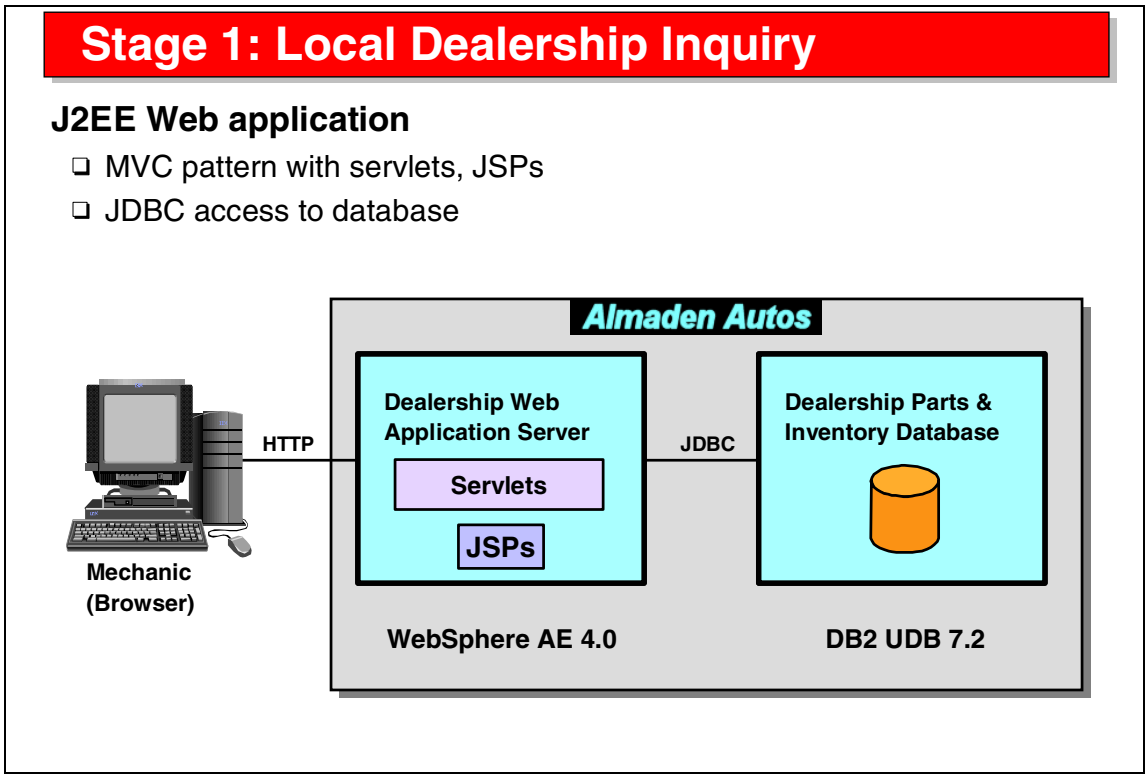


Visual 1-9 Automobile Dealership Parts Inventory

The sample is based on parts inventory applications of an automobile dealership, the manufacturer of the vehicles, and independent parts manufacturers.

The application grows in four stages:

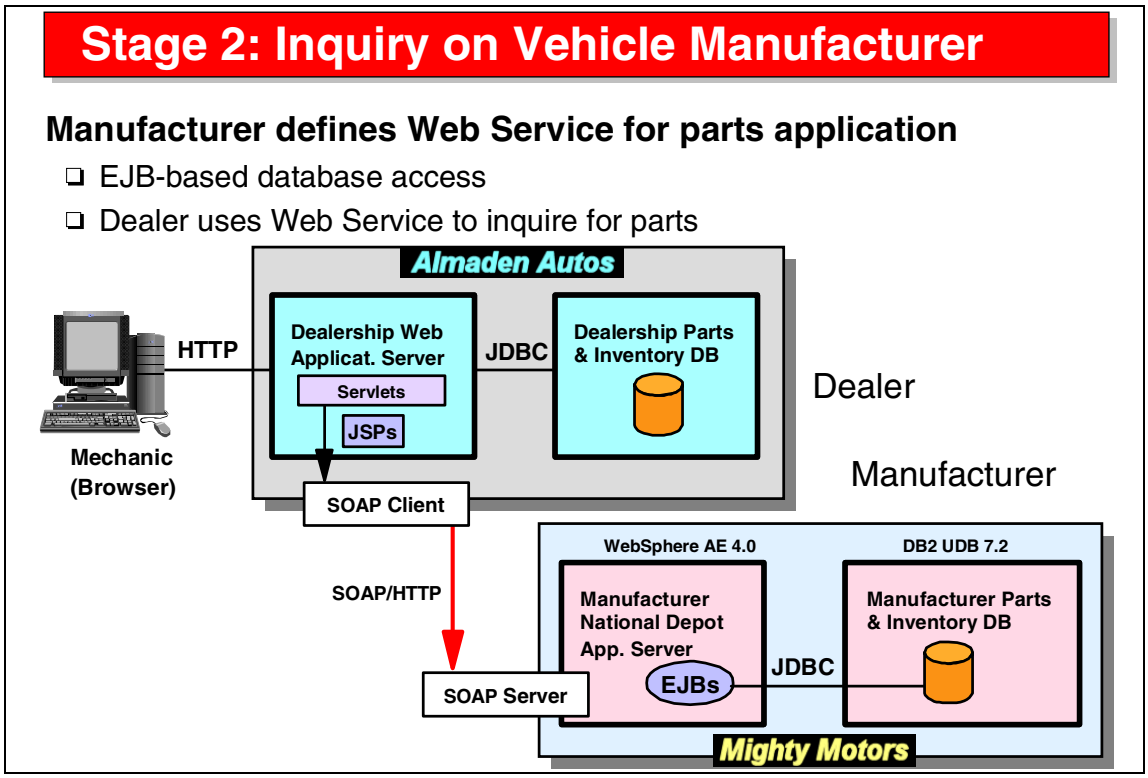
- Local Web application with servlets and JSPs
- Static Web Service, where the local application interacts with a Web Service implemented by the vehicle manufacturer
- Dynamic Web Service, where the local application interacts with multiple Web Services implemented by parts manufacturers
- Composed Web Service, where other dealers invoke the dealership application, which has been converted into a Web Service, which in turn calls other Web Services



Visual 1-10 Stage 1: Local Dealership Inquiry

In stage 1 we have a simple browser-based Web application, with servlets and JSPs, that displays parts information from an underlying DB2 database.

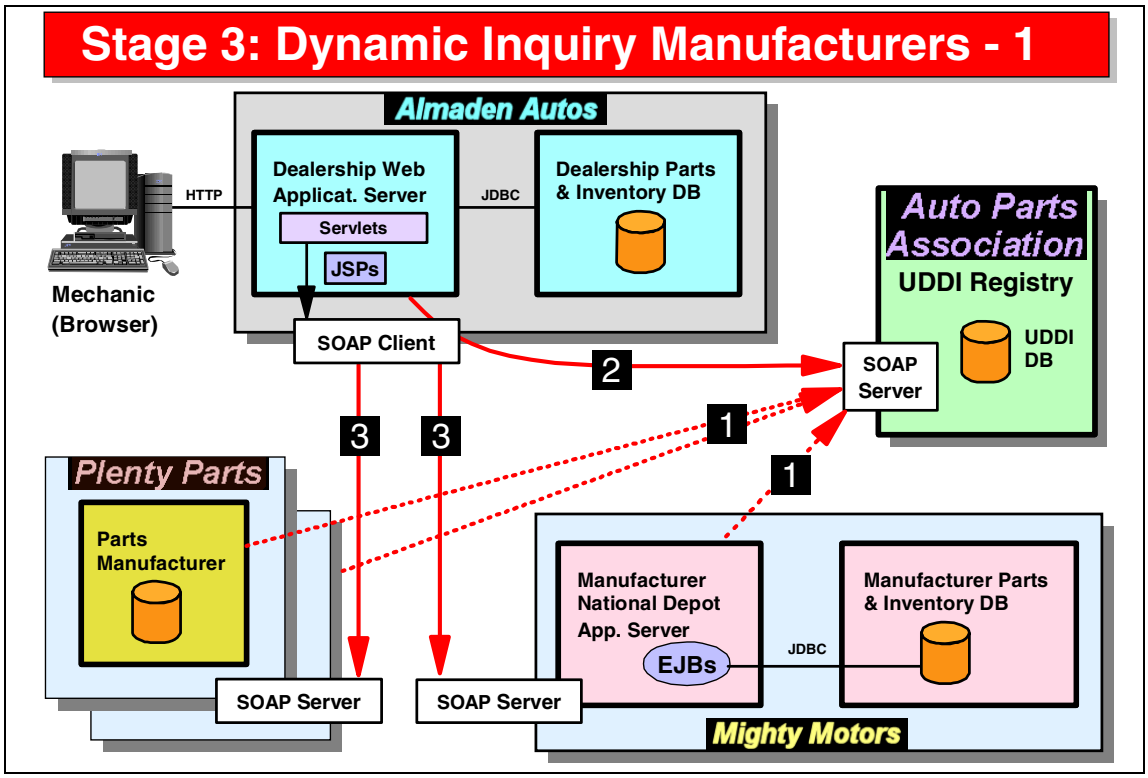
A mechanic can look up the database to find out if a certain part is available in the dealership. If a part is not available, the mechanic has to call the vehicle manufacturer to have a part shipped from the manufacturer's warehouse.



Visual 1-11 Stage 2: Inquiry on Vehicle Manufacturer

In stage 2, the vehicle manufacturer converts its existing EJB-based Web application into a Web Service.

The dealership then enhances its own Web application to invoke the Web Service. If a part is not available locally, the mechanic can now invoke the Web Services and get immediate feedback as to where the vehicle manufacturer has the requested part. (This could be extended to having the requested part shipped.)



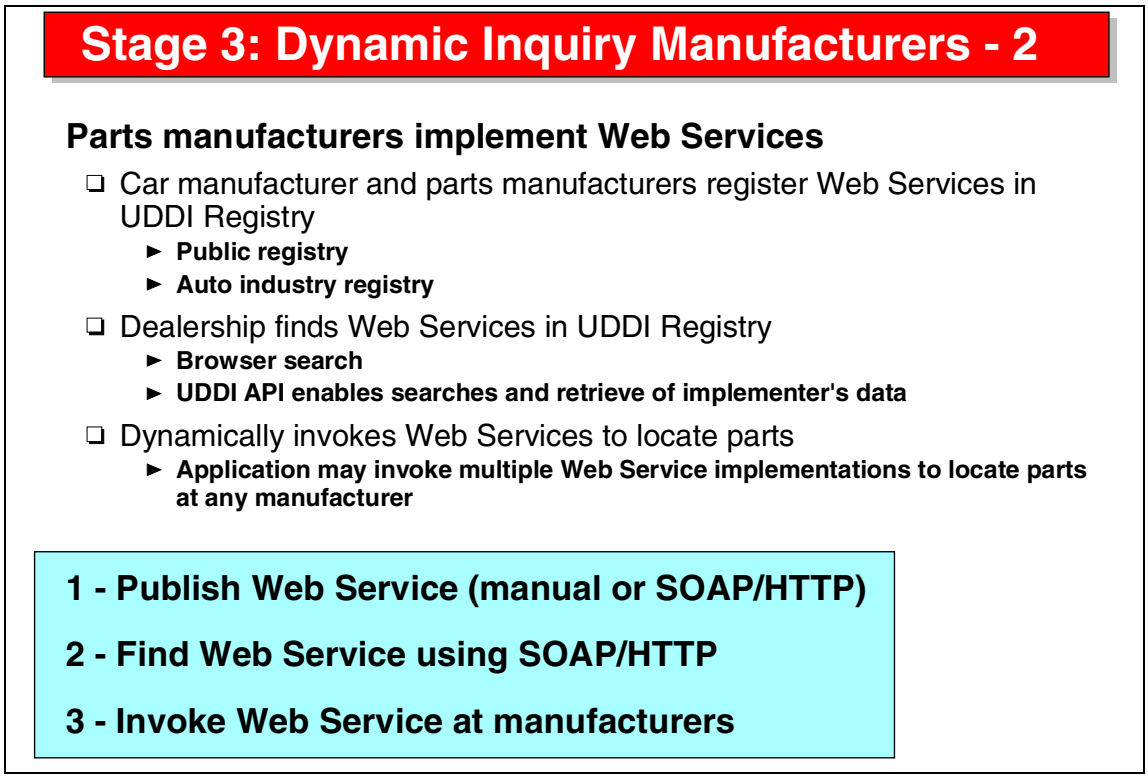
Visual 1-12 Stage 3: Dynamic Inquiry Manufacturers - 1

In stage 3, independent parts manufacturers want to participate in the automated approach.

The auto parts association (a fictional consortium of manufacturers) implements a UDDI registry where manufacturers can publish their parts inventory Web Services (1). The association checks that all Web Services implement the same interface.

The dealership Web application is now enhanced to dynamically find implementations of the parts inventory Web Service by querying the UDDI registry (2), and then invoke the Web Service at the manufacturers' sites (3).

The mechanic is presented with a list of locations where a requested part is available.



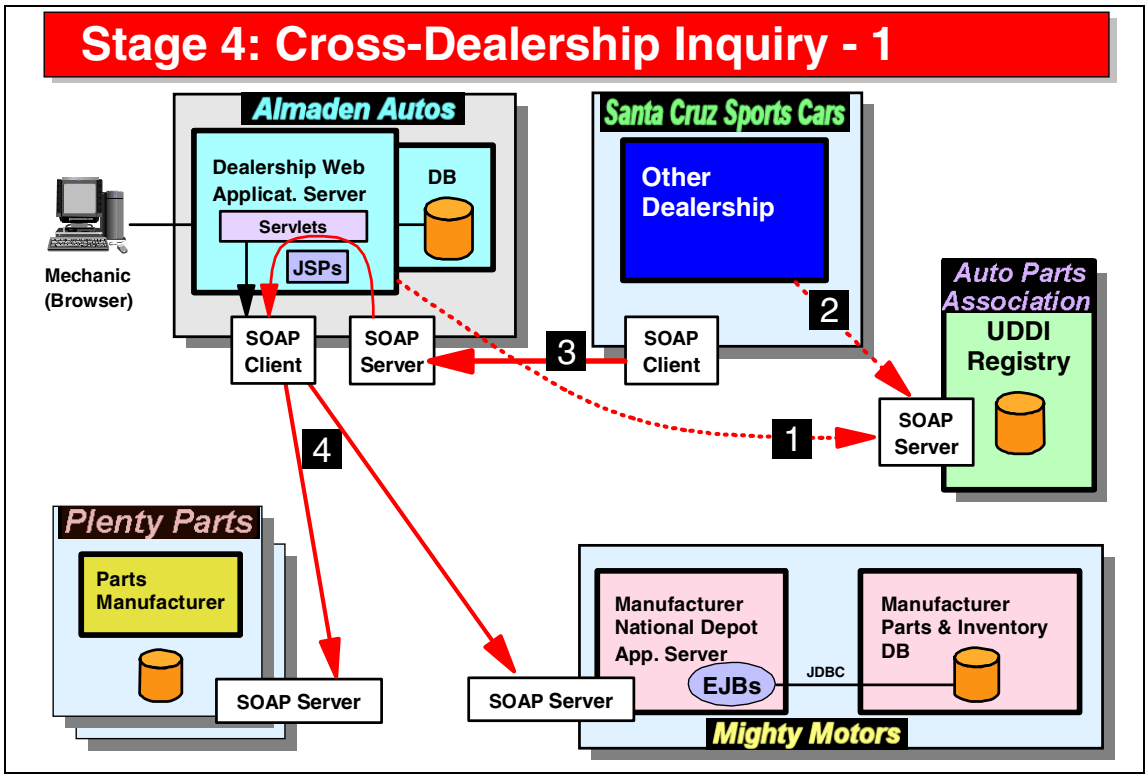
Visual 1-13 Stage 3: Dynamic Inquiry Manufacturers - 2

Parts manufacturers (vehicle manufacturer and independent) publish their Web Services in a UDDI registry.

The UDDI registry could be a public registry where all the entries of parts inventory Web Services are under control of the auto parts association, or it could be a private registry run by the association. Each part manufacturer can then register their Web Service (1).

Web Services can be located by using a browser interface to the registry, or by using an API directly from the dealership application (2).

Each Web Service implementation can then be invoked by the dealership Web application (3).



Visual 1-14 Stage 4: Cross-Dealership Inquiry - 1

In stage 4, the dealership Web application is turned into a parts inventory Web Service as well. Such a service would be available to other dealers that do not want to implement the dynamic Web Service themselves.

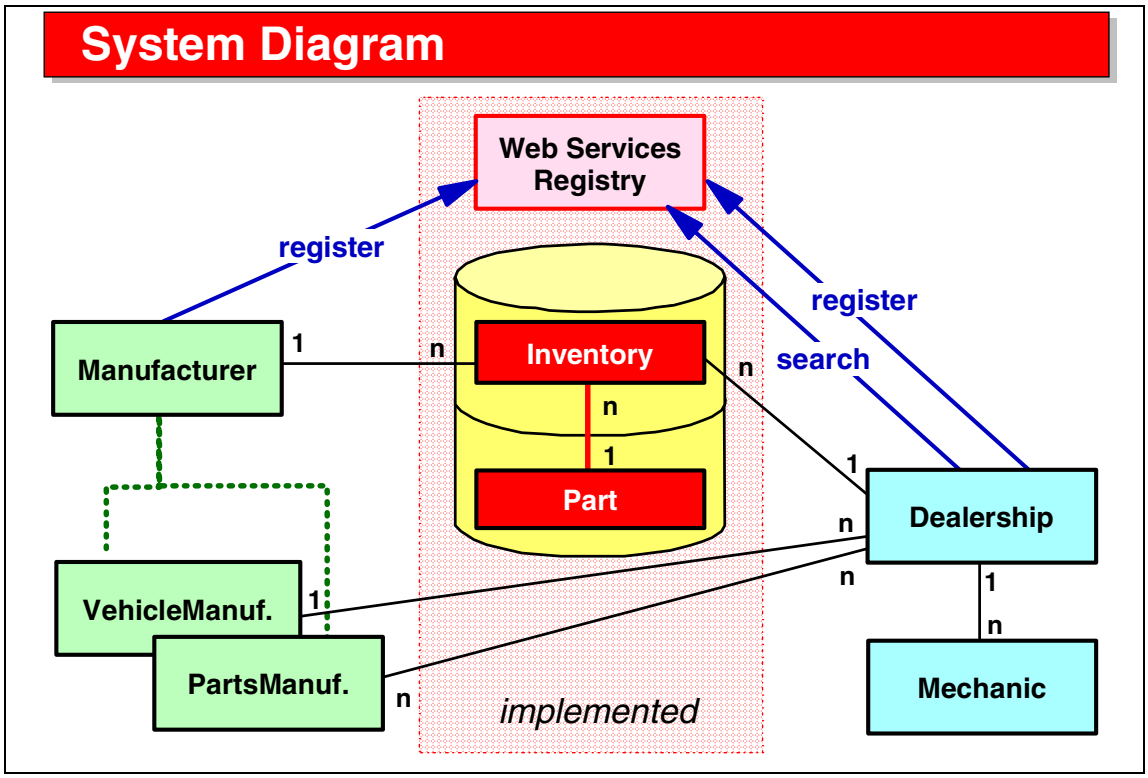
Other dealers can modify their own Web applications to call the primary dealership Web Service, which in turn invokes the manufacturers' Web Services if a part is not available locally.

Visual 1-15 Stage 4: Cross-Dealership Inquiry - 2

The dealership converts their application into a Web Service, and optionally publishes the Web Service to the UDDI registry (1).

Other dealerships can query the registry to locate parts inventory Web Services (2). Then they can invoke the Web Service of the primary dealership (3).

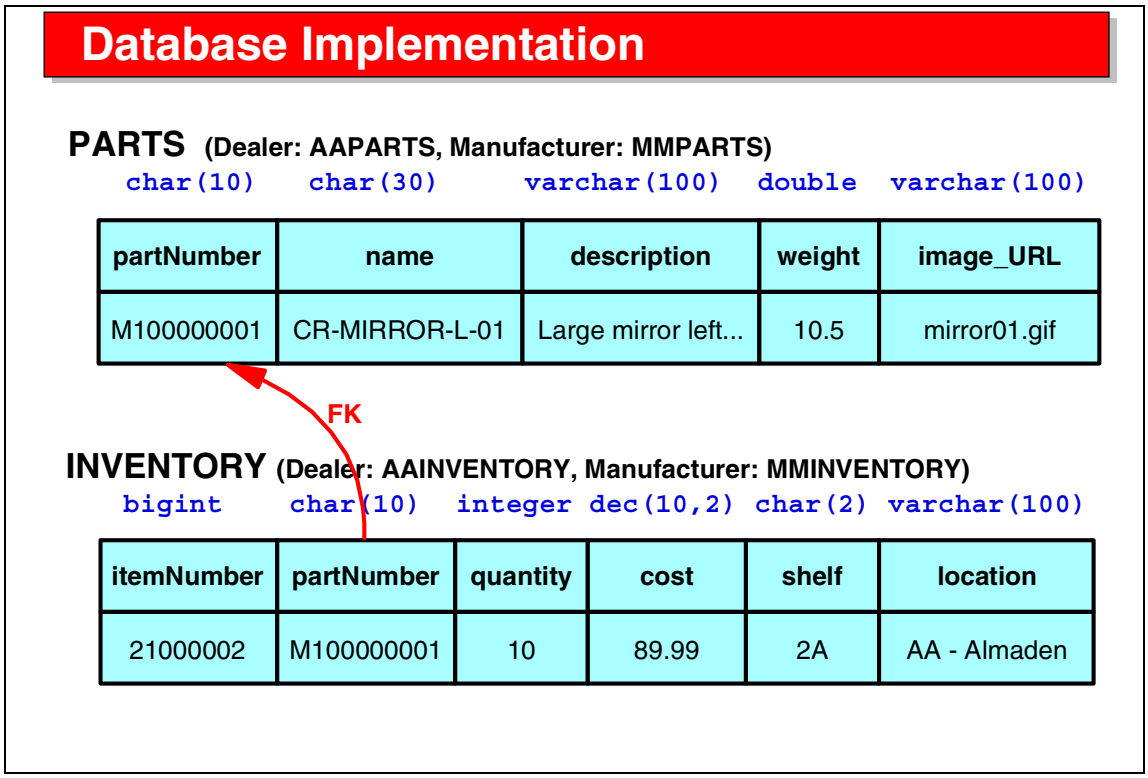
If no parts are available at the primary dealership, the Web Service invokes the manufacturers' Web Services (4).



Visual 1-16 System Diagram

This diagram shows all relationships between the parties involved:

- There are two type of manufacturers (vehicle manufacturers and independent parts manufacturers). They can register their Web Services in a UDDI Registry.
- The dealership, which employs mechanics, can query the registry, and cam also publish its own Web Service to the registry.
- The dealership is related to one vehicle manufacturer, who may have many dealerships.
- The dealership may get parts from many parts manufacturers.
- The dealership and the manufacturer store parts and inventory data in a relational database. There can be many inventory records for one part.
- In the redbook sample application, we implement a database with part and inventory tables for a dealer and a manufacturer, and a UDDI registry.



Visual 1-17 Database Implementation

The relational database contains two sets of two tables:

- AAPARTS and AAINVENTORY for the dealership
- MMPARTS and MMINVENTORY for the manufacturer

In this example, for the sake of simplicity, the two sets of tables are identical, but in real life the layout of the tables would be different.

A number of different data types are used for the columns. The choices were made to illustrate how the Application Developer deals with the different types. These choices do not reflect real life and best performance.

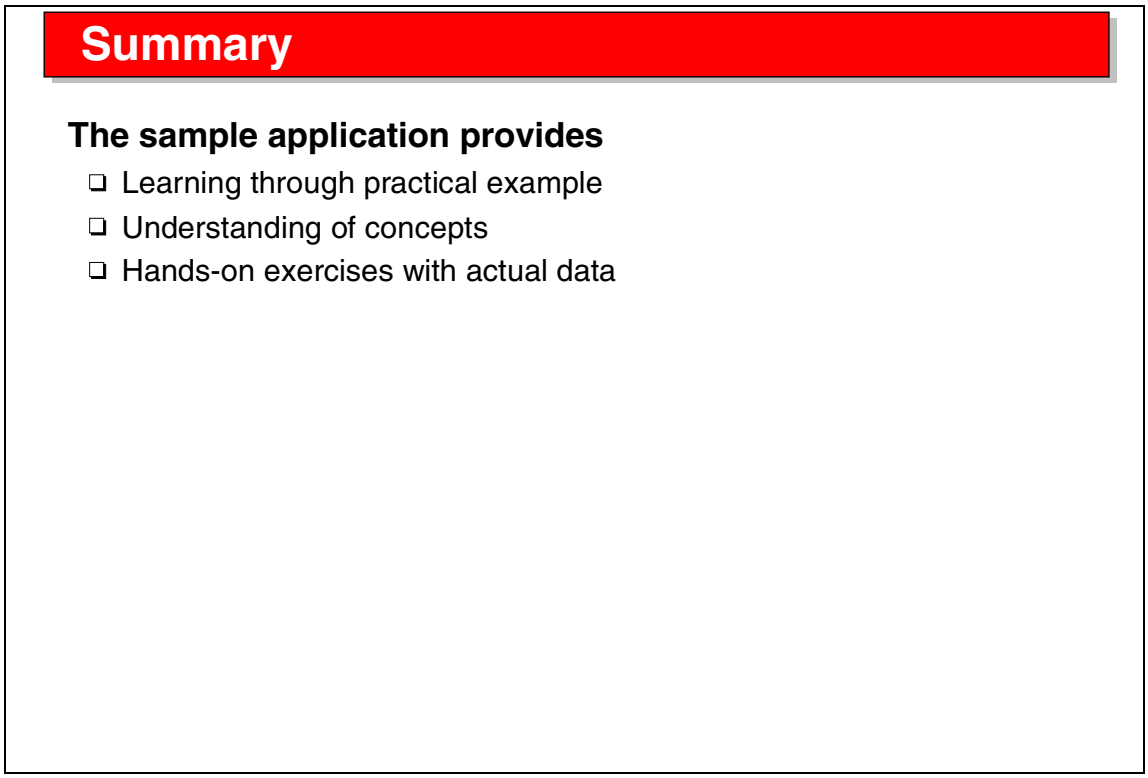
Lab Exercises

Use WebSphere Studio Application Developer to:

- Implement local dealership inquiry
  - Java development, test, debugging
  - Database development
  - XML Development
  - Web development
- Implement manufacturer EJB application
  - EJB development
  - Deployment to WebSphere AEs
  - Analyze performance (profiling)
- Create and use Web Service
  - Create Web Service for manufacturer application
  - Deployment of Web Service to WebSphere AEs
  - Use Web Service in dealer application
- Dynamic Web Services
  - UDDI registry

Visual 1-18 Lab Exercises

The class consists of lectures and 11 exercises that illustrate many of the facilities of the Application Developer.



Visual 1-19 Summary

The sample application illustrates many application development tasks and provides hands-on practice with most of the tools built into the Application Developer.

2

Application Developer: Overview

Visual 2-1 Title

Objectives

Understand the new set of tools

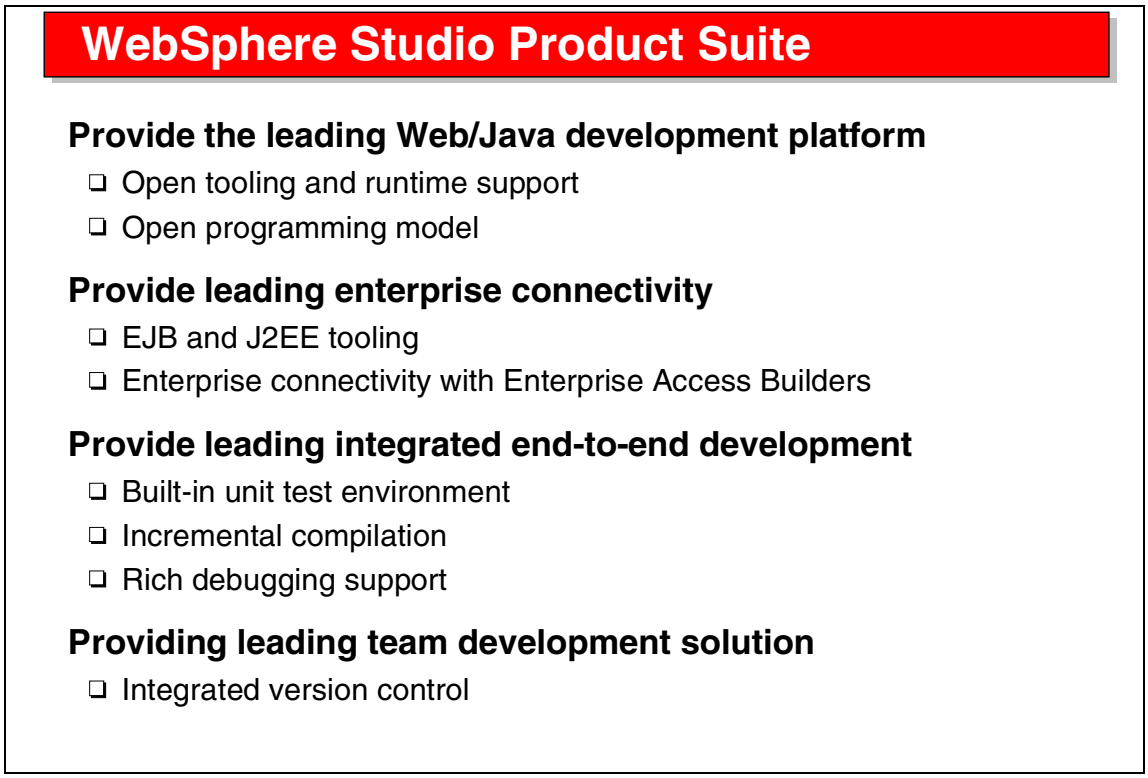
- WebSphere Studio Workbench
- WebSphere Studio Site Developer
- WebSphere Studio Application Developer
- WebSphere Studio Enterprise Developer

Understand the basic functionality of the WebSphere Studio Application Developer

- Projects
- Perspectives
- Tooling
  - **Java, Web, J2EE, Web Services, XML, Database, Tracing, Team**

Visual 2-2 Objectives

The objectives of this unit are to provide an overview of the new WebSphere Studio product suite, and a basic understanding of the functionality of the Application Developer.

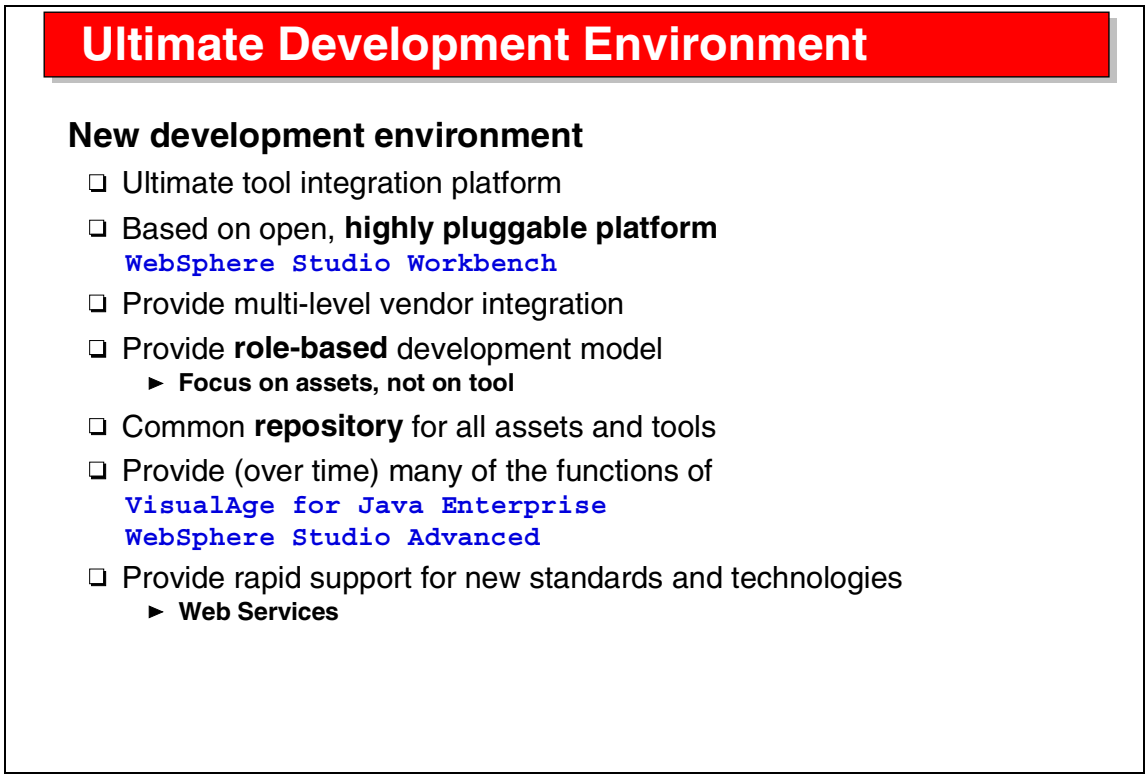


Visual 2-3 WebSphere Studio Product Suite

The WebSphere Studio product suite replaces, over time, the existing Java and Web application development tools, VisualAge for Java and WebSphere Studio (classic).

The new tools are built on an open platform and provide leading edge:

- EJB and J2EE tooling, based on J2EE 1.2 specifications
- Enterprise connectivity (not in first release)
- Built-in unit test environment (WebSphere Application Server and Tomcat)
- Incremental compilation
- Rich debugging (including remote debugging)
- Integrated team environment



Visual 2-4 Ultimate Development Environment

This ultimate development environment is built on a highly pluggable open source platform, the WebSphere Studio Workbench.

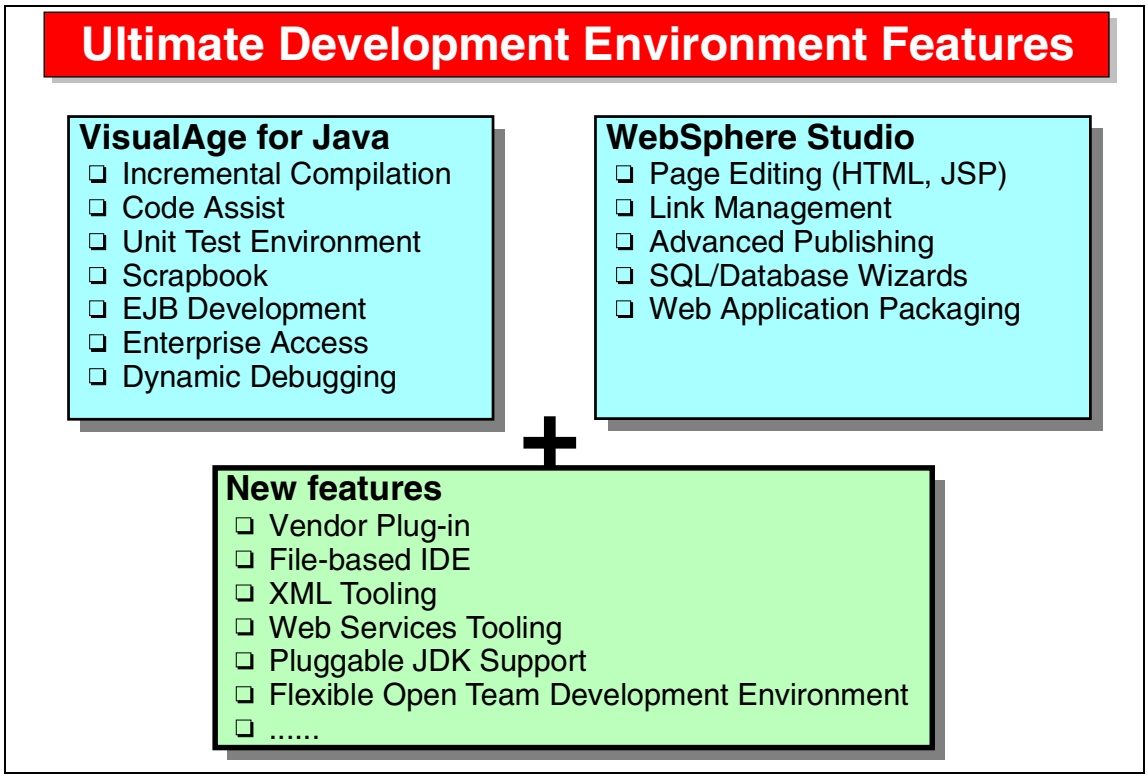
On top of this platform, IBM implemented the Application Developer, and other vendors are encouraged to integrate their tools onto the platform.

The development environment is tailored for role-based development, with appropriate user interface functions for specific roles in the development process.

All assets are stored in a file-based repository.

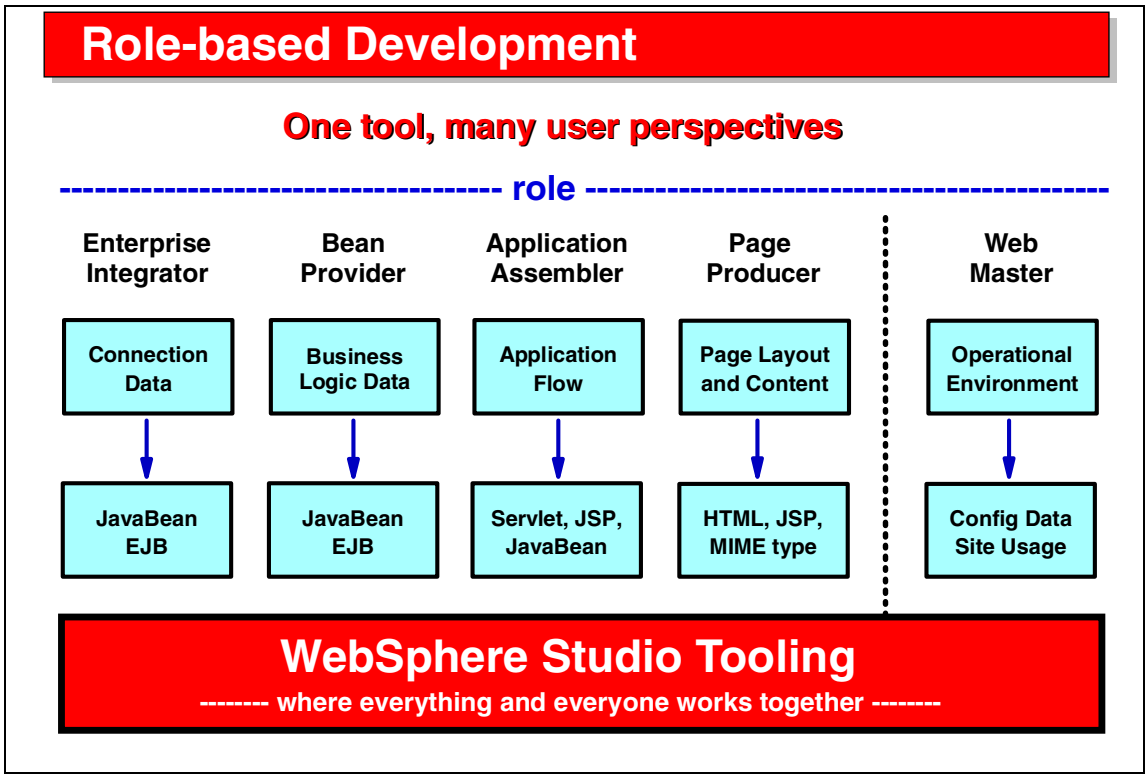
Over time, all functions of VisualAge for Java and WebSphere Studio (classic) will be implemented in the Application Developer.

In addition, Web Services are supported already in the new tool.



Visual 2-5 Ultimate Development Environment Features

The new tool combines the major functions of the existing tools, VisualAge for Java and WebSphere Studio (classic), plus a set of new functions that were not available in the old set of tools.



Visual 2-6 Role-based Development

The new tool provides functions based on the multiple roles in the development process of Web-based applications.

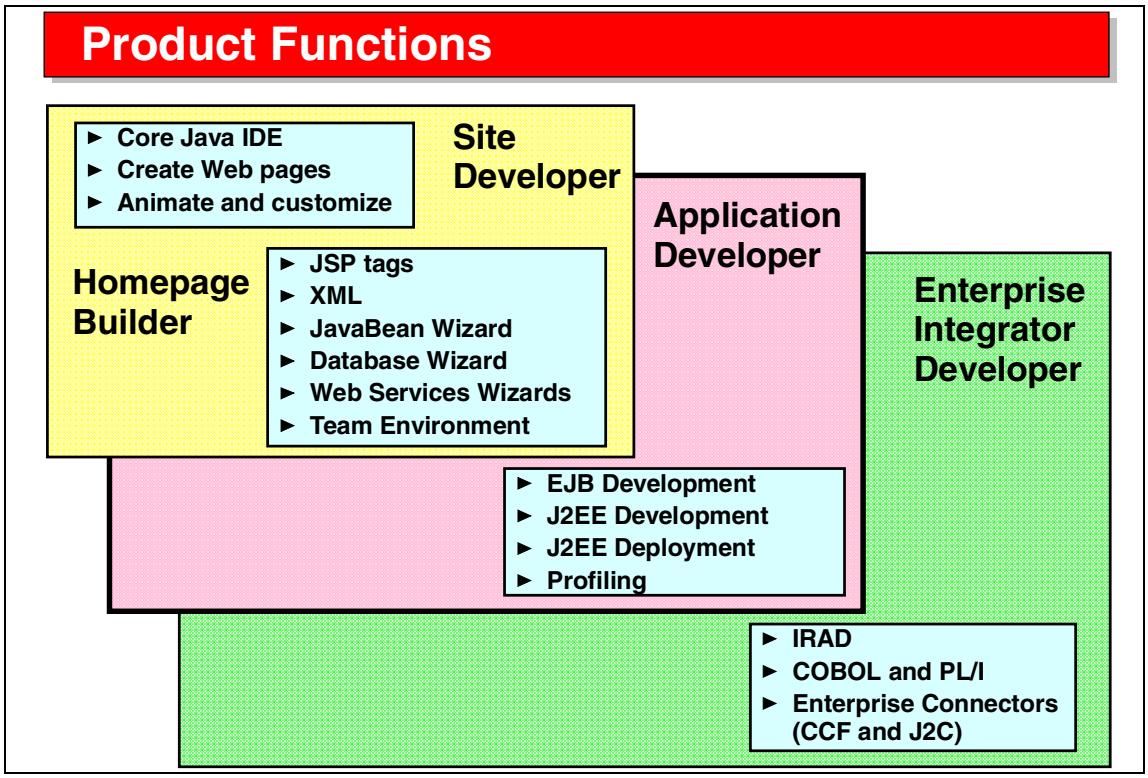
Tailored user interfaces and tools are provided for each role through perspectives. A perspective incorporates a set of views of the underlying resources, and a set of tools to manipulate those resources.

Visual 2-7 WebSphere Studio Branding

The suite of WebSphere Studio products is composed of:

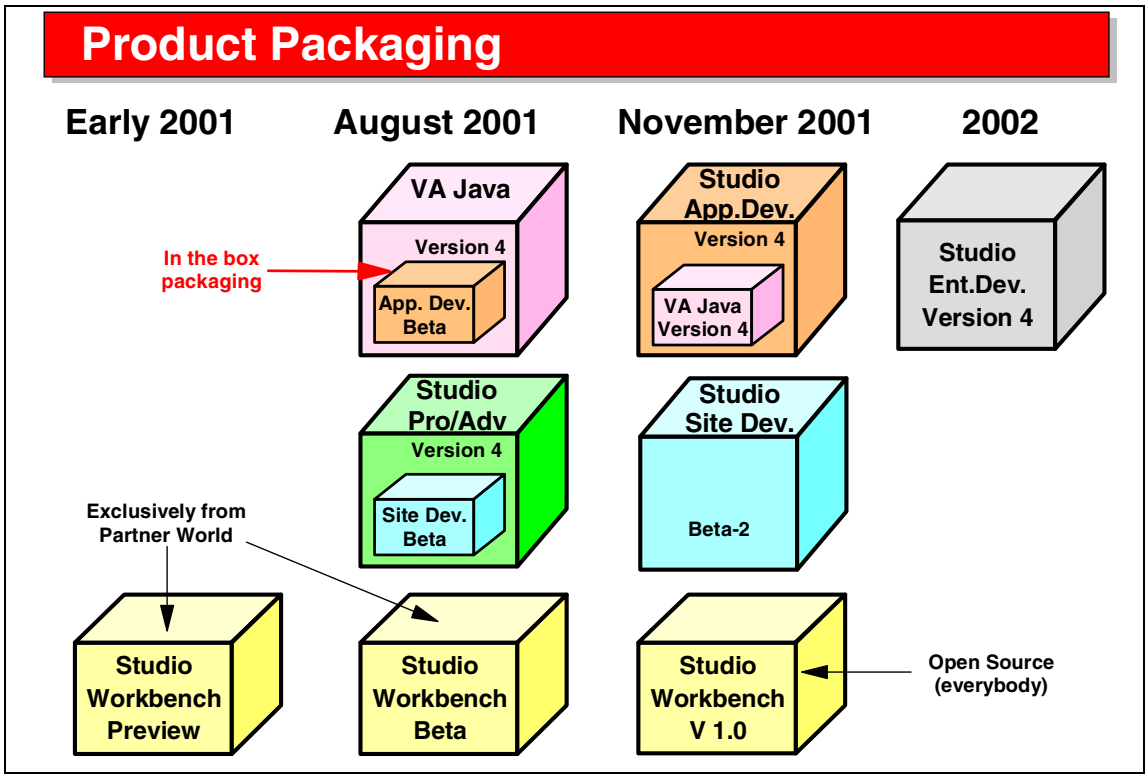
- Workbench—open source platform with underlying technology for tool builders
- Site Developer—Web application development with servlets, JSPs, XML, and Web Services
- Application Developer—adds J2EE with EJBs and full database support to the Site Developer
- Enterprise Integrator—adds connectors and flow modeling
- Enterprise Developer—adds support for z/OS and eRad (the VisualAge Generator technology)

Note: Enterprise Integrator and Enterprise Developer are tentative names.



Visual 2-8 Product Functions

This diagram shows the increase in functionality from the simple Homepage Builder (which is not based on the Studio Workbench), to the Site Developer, Application Developer, and the Enterprise Integrator and Developer.



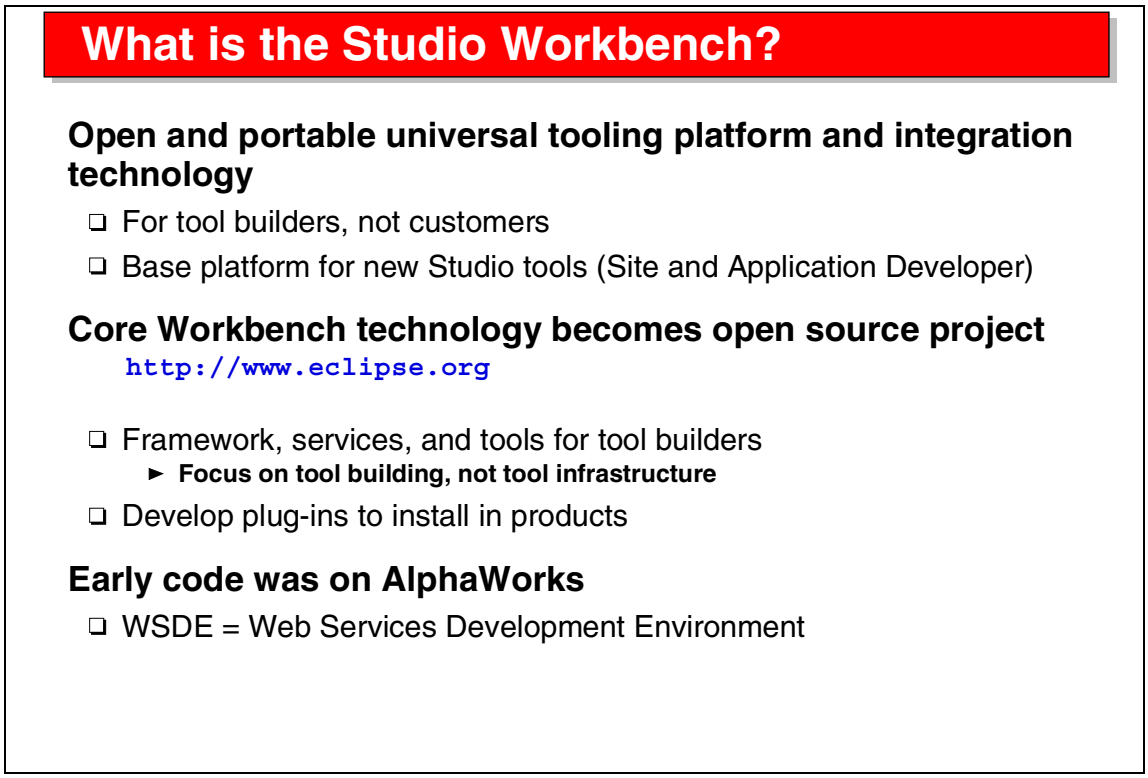
Visual 2-9 Product Packaging

In August 2001, the beta code of the Application Developer was made available together with VisualAge for Java Version 4.

In November 2001, the final Application Developer product was made available, and a copy of VisualAge for Java is included with the product. At the same time, the Studio Workbench was made open source for any vendor/developer.

The Site Developer has been available as beta code since November 2001, and will become available in the first half of 2002.

The Enterprise Integrator and Enterprise Developer will become available during 2002.



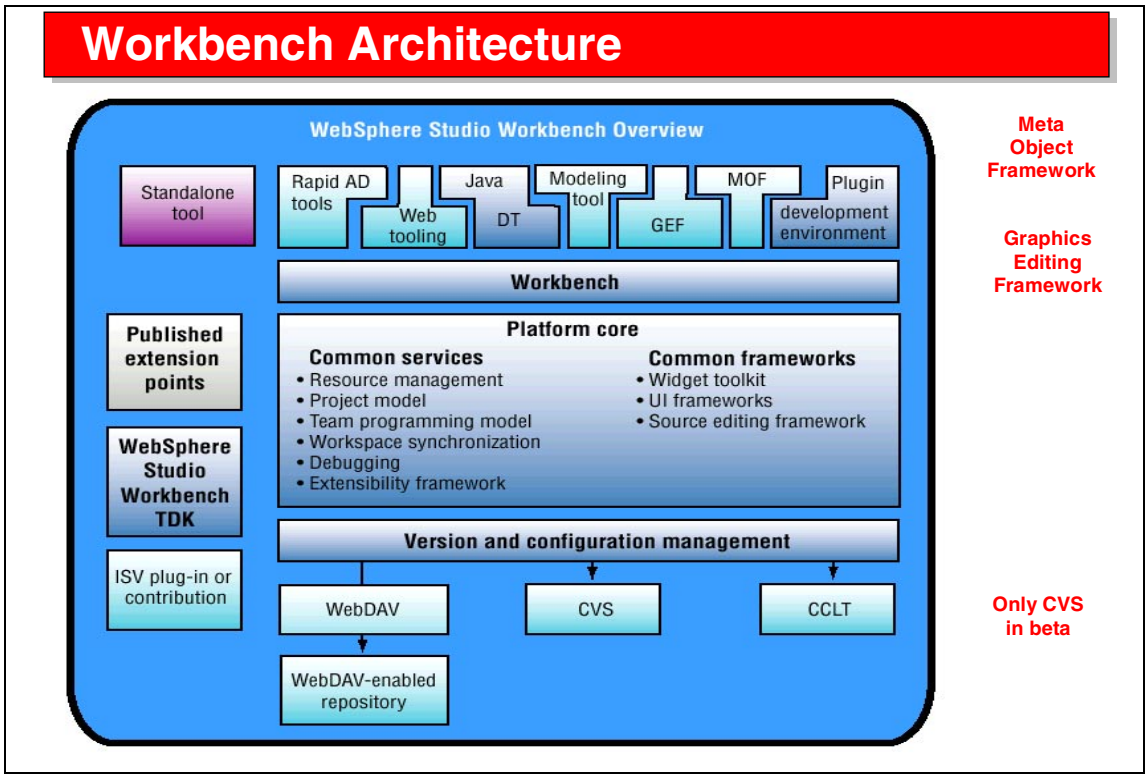
Visual 2-10 What is the Studio Workbench?

The Studio Workbench contains the underlying technology for the Site and Application Developer, and for any vendor that wants to integrate to the platform.

The Workbench is open source and available at:

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

Tool builders can use the Workbench to integrate their tools to the platform by creating plug-ins that can be added to the Site or Application Developer.



Visual 2-11 Workbench Architecture

The basic architecture is shown in this diagram.

The platform core provides the integration points (called extension points) for plug-ins. Plug-ins are used to add tools to the platform by implementing some of the extension points.

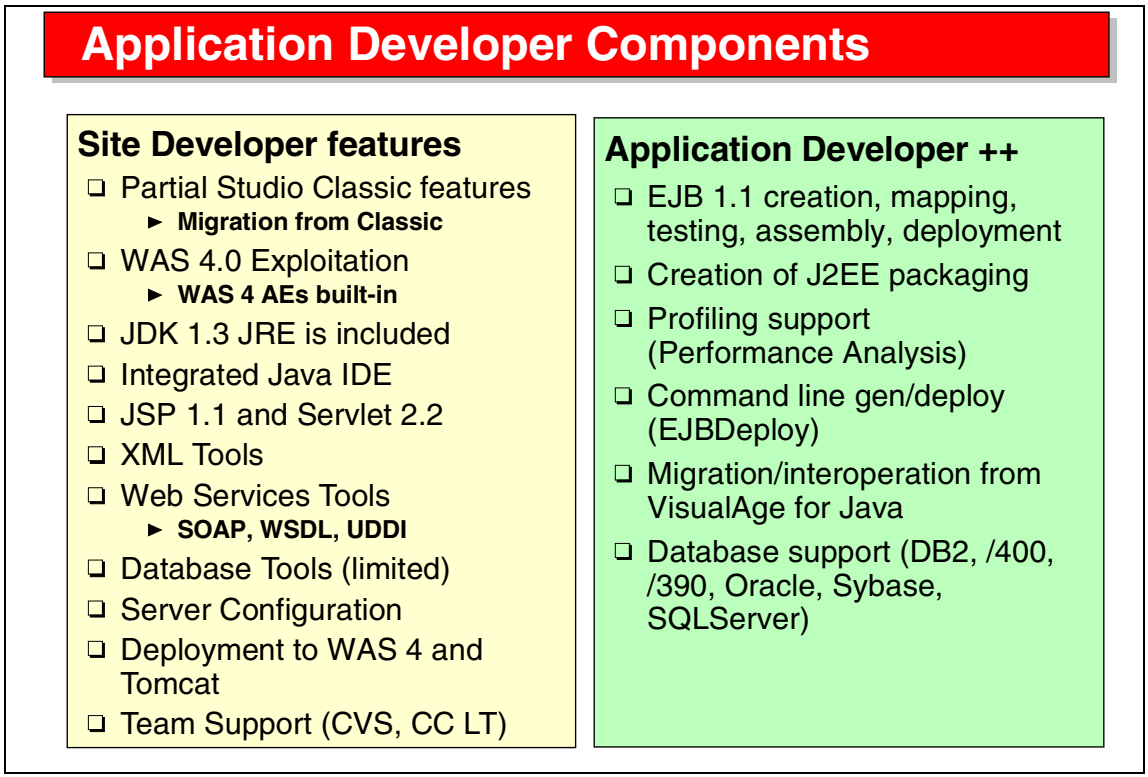
One set of extension points is for version and configuration management. This is currently implemented by Concurrent Version System (CVS) and ClearCase Light (CCLT), and will be supported by WebDAV in the future.

Frameworks can be used by tools for functionality:

- Graphics Editing Framework (visual builders)
- Meta Object Framework (to store tool data in XMI files)
- Java Development Tooling (compilation)

Visual 2-12 Application Developer Overview

In this section we provide an overview of the functions of the Application Developer.



Visual 2-13 Application Developer Components

The features of the Application Developer can be grouped into features that are also available in the Site Developer, and features that are exclusive to the Application Developer:

- Full EJB 1.1 support
- J2EE 1.2 support
- Profiling (performance analysis tools)
- EJB deployment
- Migration and interoperation with VisualAge for Java
- Full database support for many platforms and database vendors

Prerequisites and Platforms

Hardware

- Pentium II or better, 256 MB
  `For realistic work environment 384MB or 512MB`
- Disk: 400 MB (70 MB TEMP for installation)

Software

- Windows NT 4 (SP 6a) or 2000 (SP 1), Windows 98/ME, Linux (beta)
- Database support:
  - **DB2 UDB 6.1, 7.1, 7.2, DB2/390 7.1 FP3, DB2/400**
  - **Oracle 8i R3 8.1.7, Sybase Adaptive 11.9.2, 12.0**
  - **SQL Server 7.0 SP2 and 2000 (Merant driver)**
  - **Informix 7.31 and 9.21 (Merant driver)**

Deployment platforms

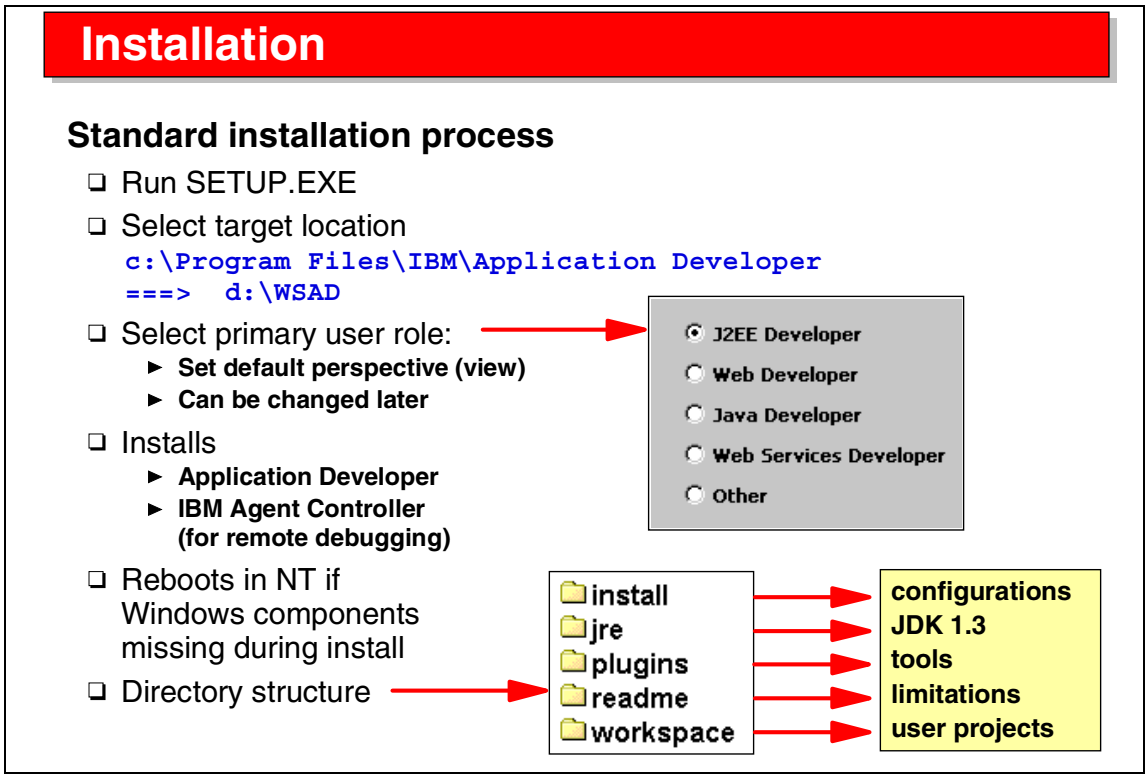
- Dependent on Application Server (for example, WebSphere AE)
  - **Windows NT 4 SP6a, Windows 2000 and 2000 Server**
  - **AIX 4.3.3, /390, /400, Solaris, HP-UX, Linux**
  - **Windows 98/ME**

Visual 2-14 Prerequisites and Platforms

The Application Developer requires 384 or 512 MB of memory for decent operation with use of an application server and a relational database at the same time.

The development environment is limited to Windows (200, NT, 98, ME) for now, with Linux coming in 2002.

Deployment of Web applications is supported for all platforms where WebSphere Application Server is available.

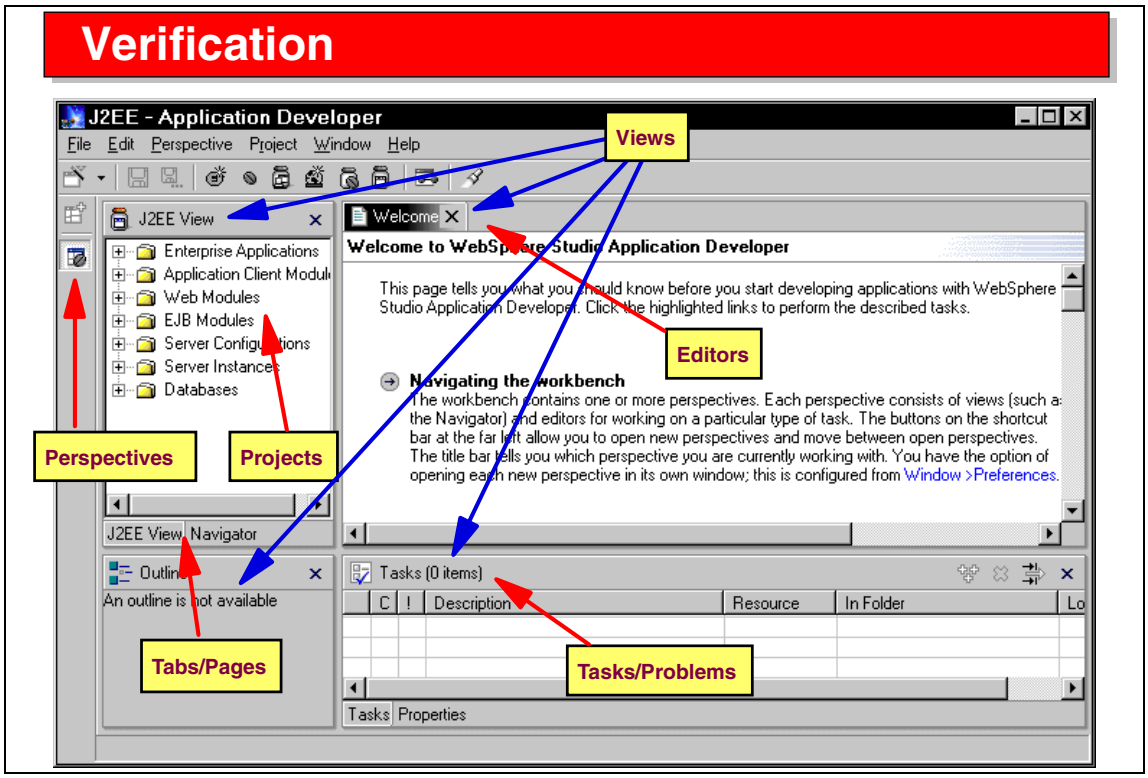


Visual 2-15 Installation

Installation of the Application Developer is straightforward.

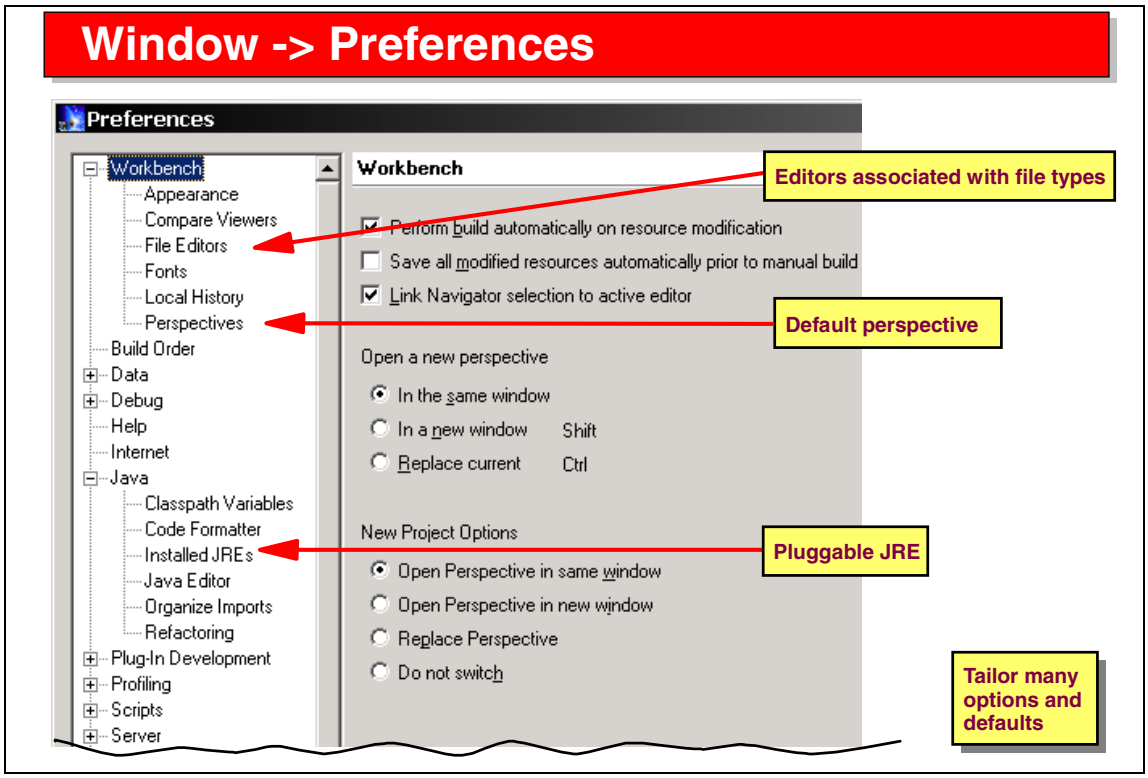
You have to choose the installation directory and a primary user role (which can be changed later).

The process also installs the IBM Agent Controller, which is required for remote debugging and performance analysis. The Agent Controller must be manually installed on other machines and platforms where deployed code is executing for debugging or performance analysis.



Visual 2-16 Verification

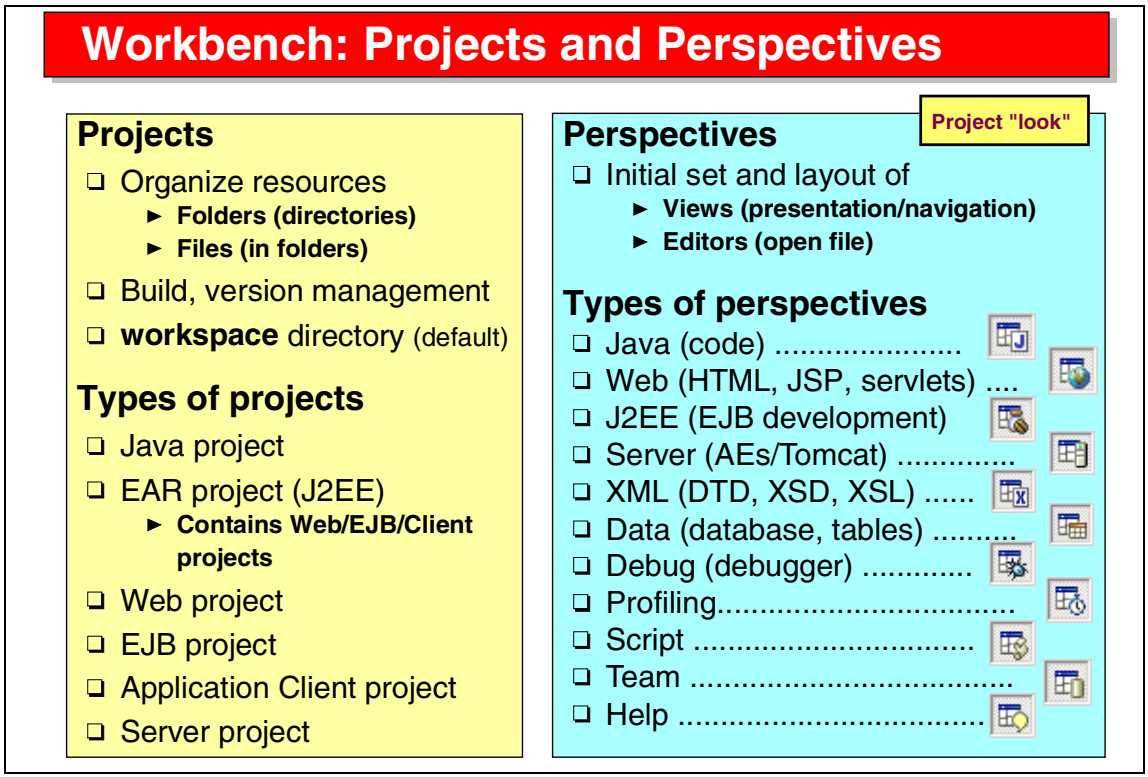
When the product is started after installation, a welcome panel is displayed.



Visual 2-17 Window Preferences

Many defaults can be set in the Window Preferences dialog (select *Window -> Preferences* in the menu bar).

For example, you can configure additional editors based on the file type, or install additional Java runtime code (JRE) that can be used for certain projects.



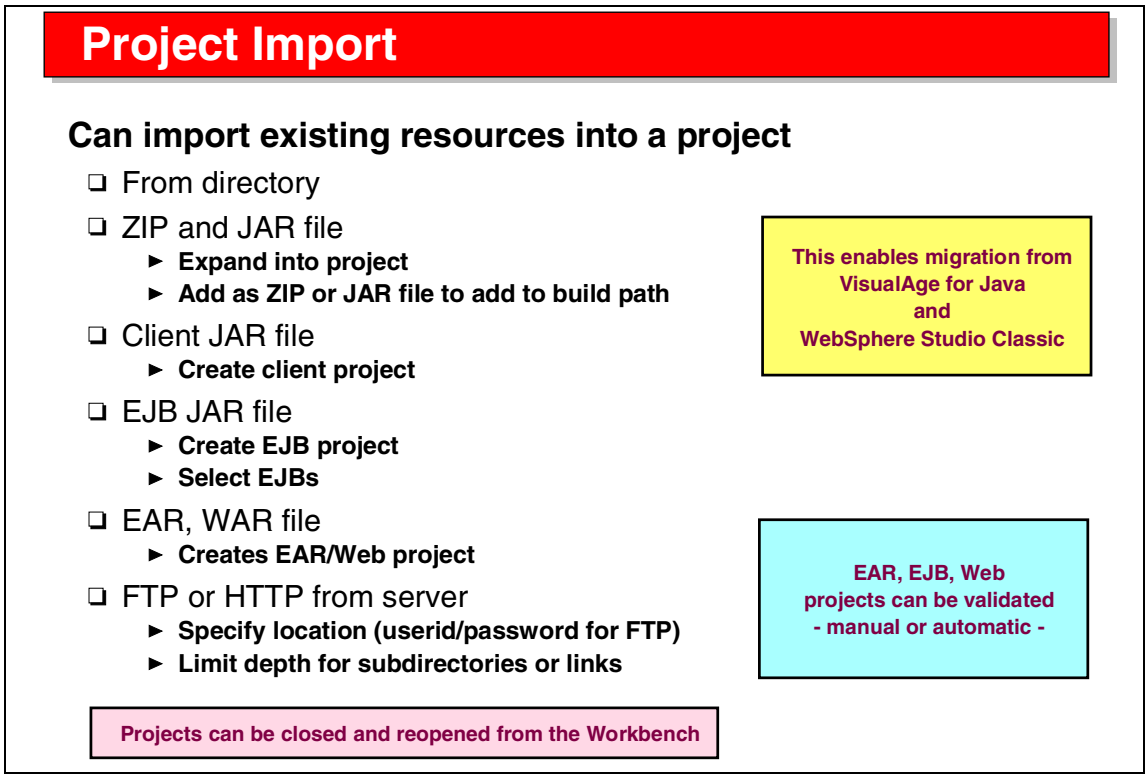
Visual 2-18 Workbench: Projects and Perspectives

The Application Developer organizes all resources (data, programs) into projects. Projects are composed of folders (directories) and files. By default, projects and their folders are stored in a workspace directory.

Each project has a type: Java projects are for stand-alone applications; Web projects for Web applications; EJB projects for EJB development; EAR projects tie together Web and EJB (and Client) projects into a J2EE hierarchy; Server projects are used to define application servers for testing.

Perspectives are the way a developer sees the projects. Perspectives are tailored for certain tasks, based on the role of the developer. For example, in the Java perspective you can compile Java code; in the Web perspective, you can edit and customize Web applications; in the J2EE perspective, you develop J2EE hierarchies and EJBs.

A perspective is a set of views, arranged into the Workbench window, and a set of editors and tools that are used to manipulate the resources.



Visual 2-19 Project Import

In most development efforts you have existing resources. These resources can be imported into Application Developer projects in many ways:

- Files from a directory (Java source, HTML, JSPs...)
- ZIP and JAR files (import as individual files for editing, or leave as ZIP/JAR file if only used for compilation)
- EJB JAR files with existing EJB definitions (for example, from VisualAge for Java)
- EAR and WAR files from existing Web applications and J2EE archives
- FTP or HTTP access to existing Web sites for import of HTML and associated files

Projects can be closed to save memory.

Project Validation

Comprehensive support for running validation

- EJB project: EJB validator, Map validator
- EAR project: EAR validator
- Web project: WAR validator
- Server project: Configuration files

Manual validation

- Project -> Validate Project

Automatic validation (on save)

- Project -> Properties -> Validation
  - **Can specify which validators to run**
  - **Is somewhat expensive**

Validation results are shown in the task list

Visual 2-20 Project Validation

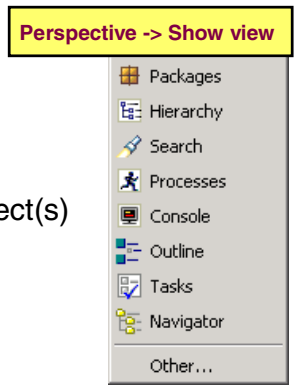
Comprehensive validation is provided for many types of projects, for example, the EJB validator checks that EJB 1.1 specifications are followed in the Java code.

Projects can be validated automatically when a resource is saved (this is the default), or validation can be performed manually on demand.

Perspectives

Focus on task at hand

- Provides "views" into the e-business application
  - **By role (Java, data, EJB)**
  - **By task (develop, test, debug, deploy)**
- Views can be opened, closed, rearranged
  - **Move view to other pane, stack behind other views**
  - **Move view outside of Workbench as separate window**
  - **Maximize view with double-click**
    - **code editing**
- Multiple perspectives can be opened for same project(s)
- Perspectives can be customized
  - **Add views**
  - **Make look similar to other Java tools**
- Certain tasks can only be performed in one perspective and view



Icon to open new perspectives: 

Visual 2-21 Perspectives

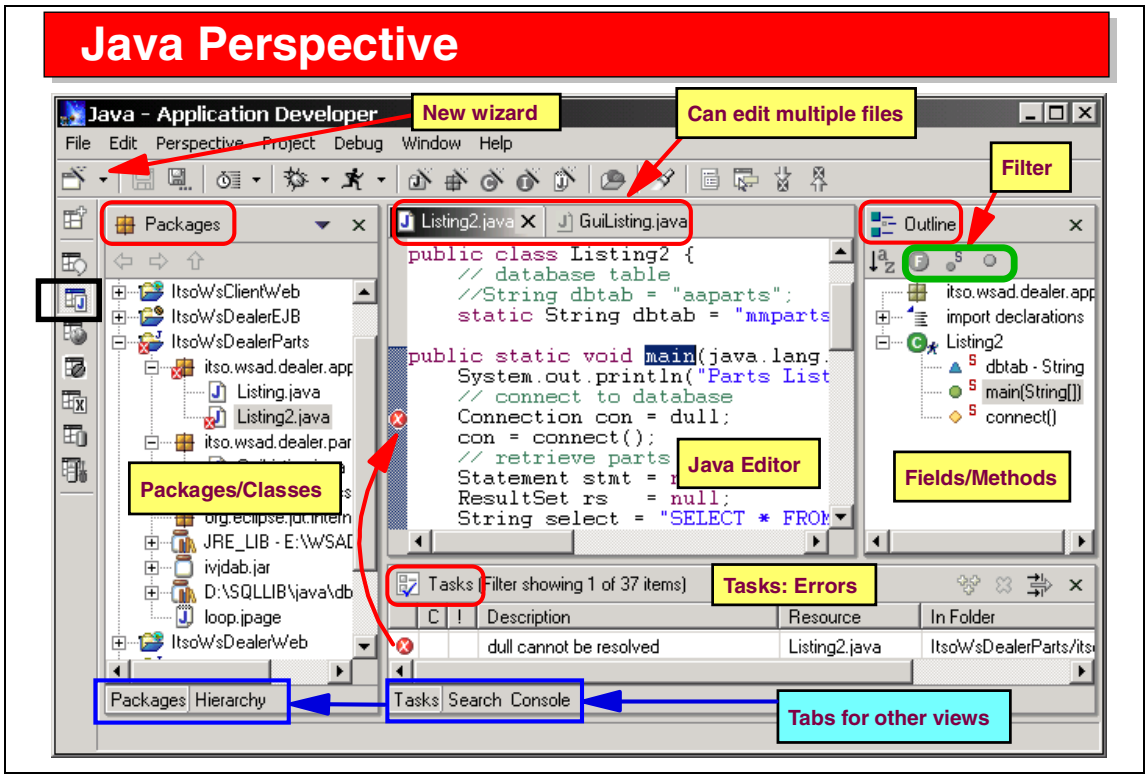
Perspectives are tailored for a developer role or for a certain task.

A perspective provides a number of views of the underlying project resources. The layout of the Application Developer window can be tailored by moving views to the edge, between other views, behind other views (as tabs), or as separate windows. Views can be added to a perspective, and the new layout can be saved as a new perspective.

In general you see one perspective in the Application Developer window, but you can have multiple perspectives open and switch between them. The more perspectives are open, the more memory is consumed. In each open perspective you can have multiple files (resources) open.

Certain tasks can only be performed in one perspective and view (for example, EJB development), while other tasks can be performed in many perspectives (for example, Java editing).

Good practice: close files before switching perspectives!



Visual 2-22 Java Perspective

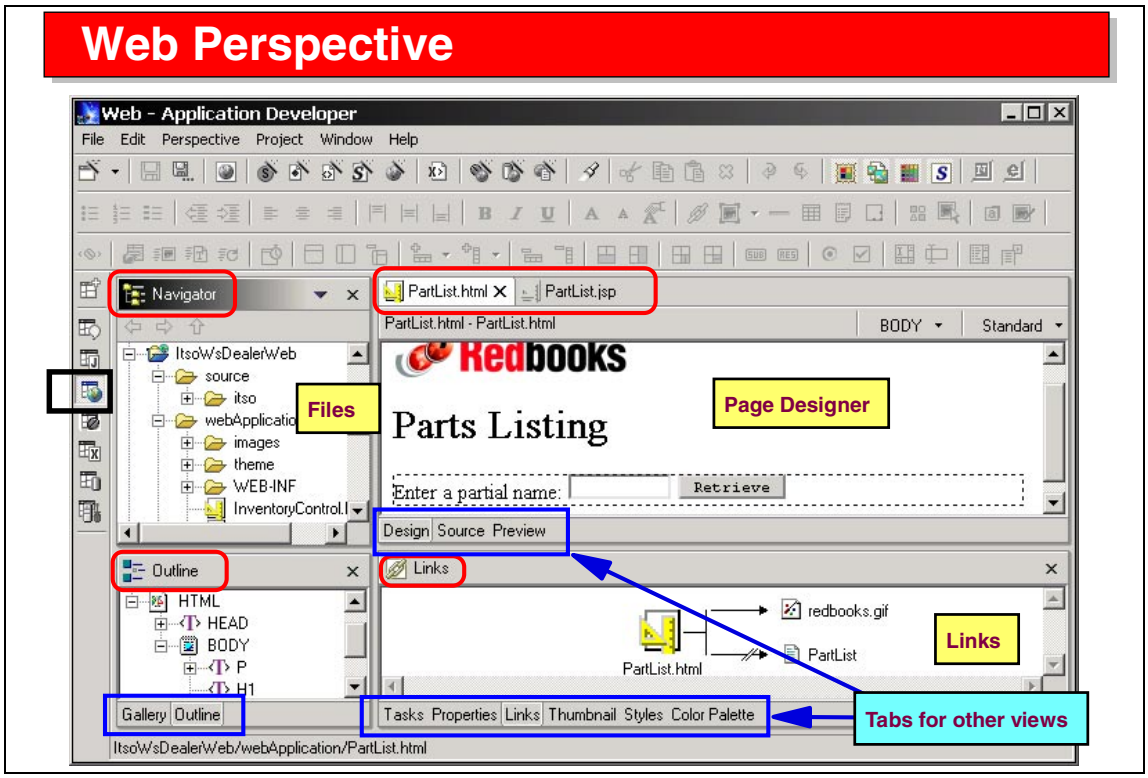
The Java Perspective contains four panes by default:

- Left—Packages and Hierarchy view
- Middle—reserved for editors (multiple files can be edited)
- Right—Outline view (shows outline of currently selected file in editor pane)
- Bottom—Tasks (error messages), Search (result of search operations), and Console (program output) views

Tabs are used to select a view in a pane.

Each perspective has a tailored toolbar with icons for often-used operations.

On the top left is the new icon or wizard. The pull-down arrow displays a selection of resources that can be created, whereas the icon button displays the new wizard that can be used to create any of the supported resource types or tools.

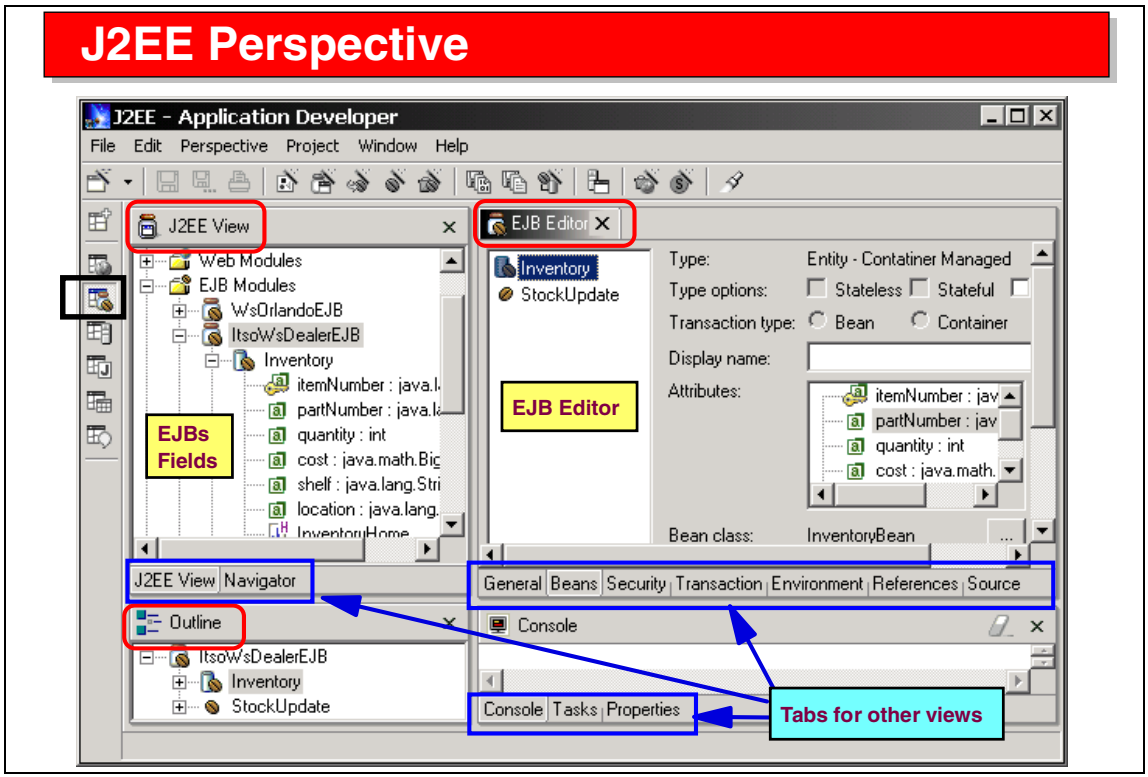


Visual 2-23 Web Perspective

The Web Perspective contains four panes:

- Top left—Navigator view (displays the folders and files of the projects)
- Top right—reserved for editors
- Bottom left—Outline view (of current editor) or Gallery (for HTML/JSP files)
- Bottom right—Tasks (errors), Properties (of selected resource), Links (of Web resources), Thumbnail, Styles, Color, Palette (Web resources)

One of the supported editors is the Page Designer for HTML and JSP files. The Page Designer itself has three tabs to display the Design (WYSIWYG), Source (HTML source code), or Preview (browser) view.



Visual 2-24 J2EE Perspective

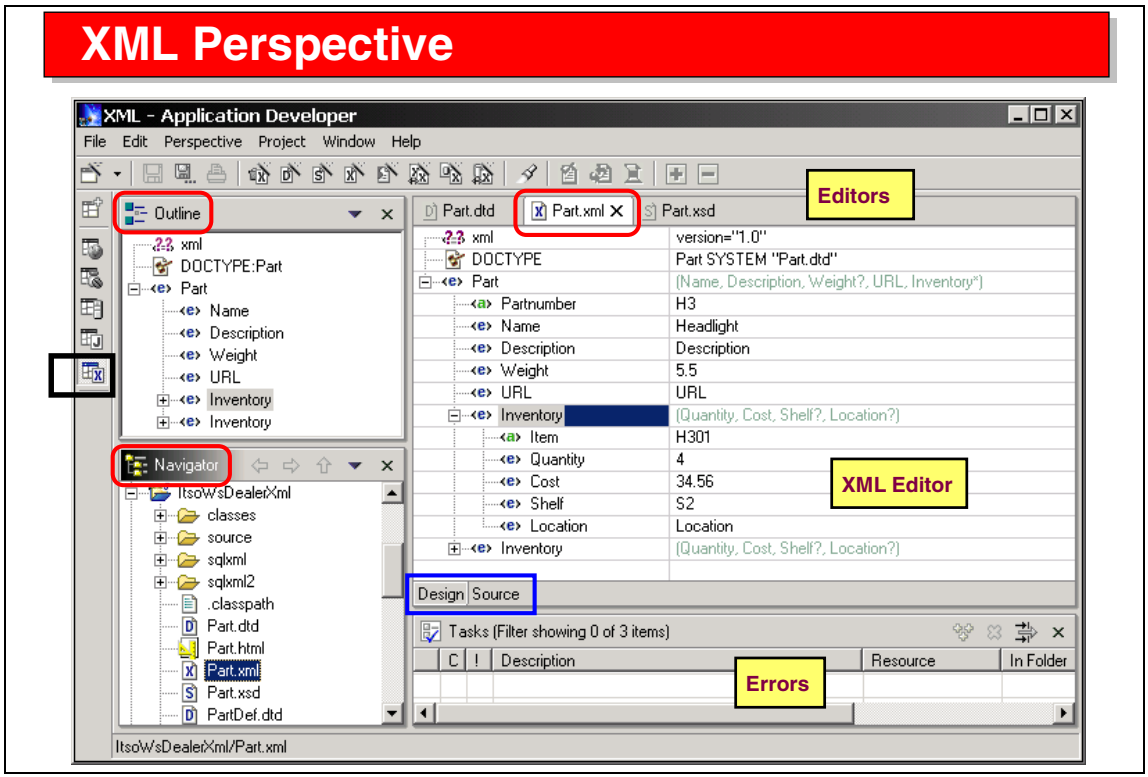
The J2EE Perspective is used for management of J2EE deployment descriptors (EAR, enterprise archives), and for development of EJBs.

The J2EE view is the only view where entity and session EJBs can be developed. This view displays a logical view of the EJBs with their fields, key, and main underlying Java files (bean class, home and remote interface, key class).

The Navigator view displays all the project resources, including the control files (XMI files) that are used to store the EJB design (meta) information.

An EJB Editor is provided to define and manipulate EJB deployment information, such as JNDI names, transaction attributes, read-only methods, and security information.

An EJB Extension Editor is provided to define IBM extensions of the EJB specification, such as associations and custom finders.

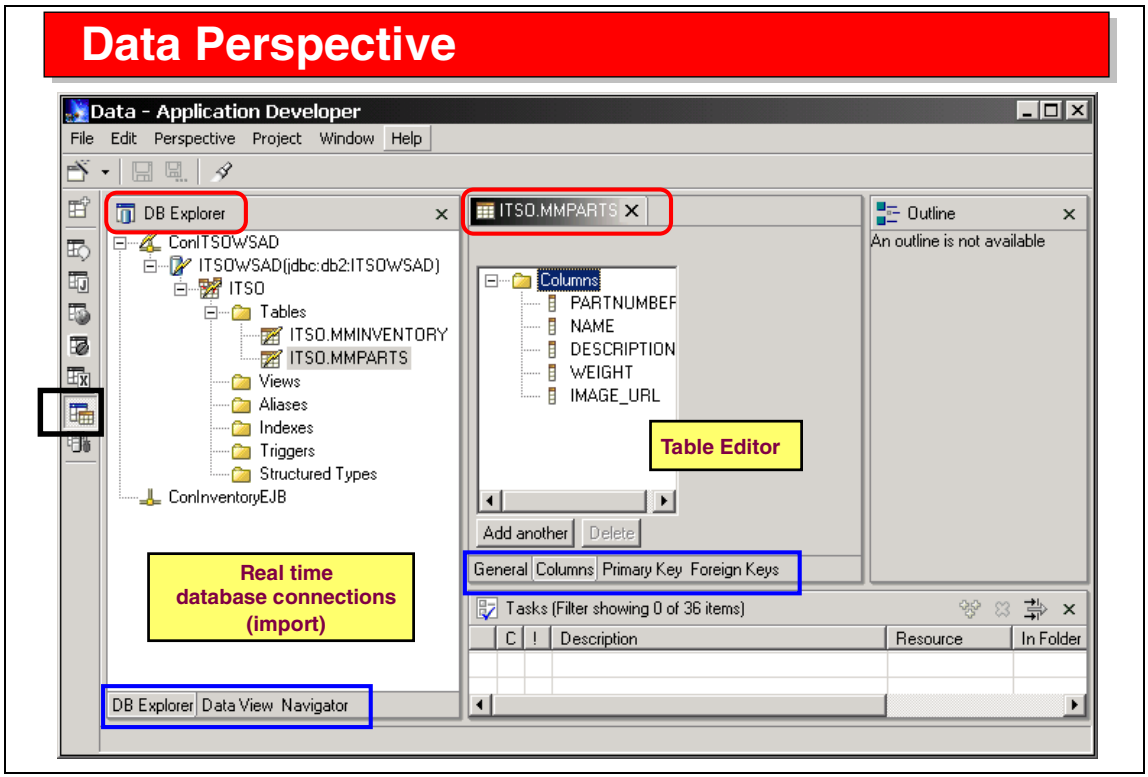


Visual 2-25 XML Perspective

The XML perspective is provided for developing XML applications, or adding XML functionality to Web applications.

XML editors for XML files, DTDs, and XML schemas are provided for manipulation of XML resource files.

XML tools are provided to convert XML descriptors (DTDs and schemas), generate files, generate XSL style sheets for XML manipulation, or convert database data into XML data.



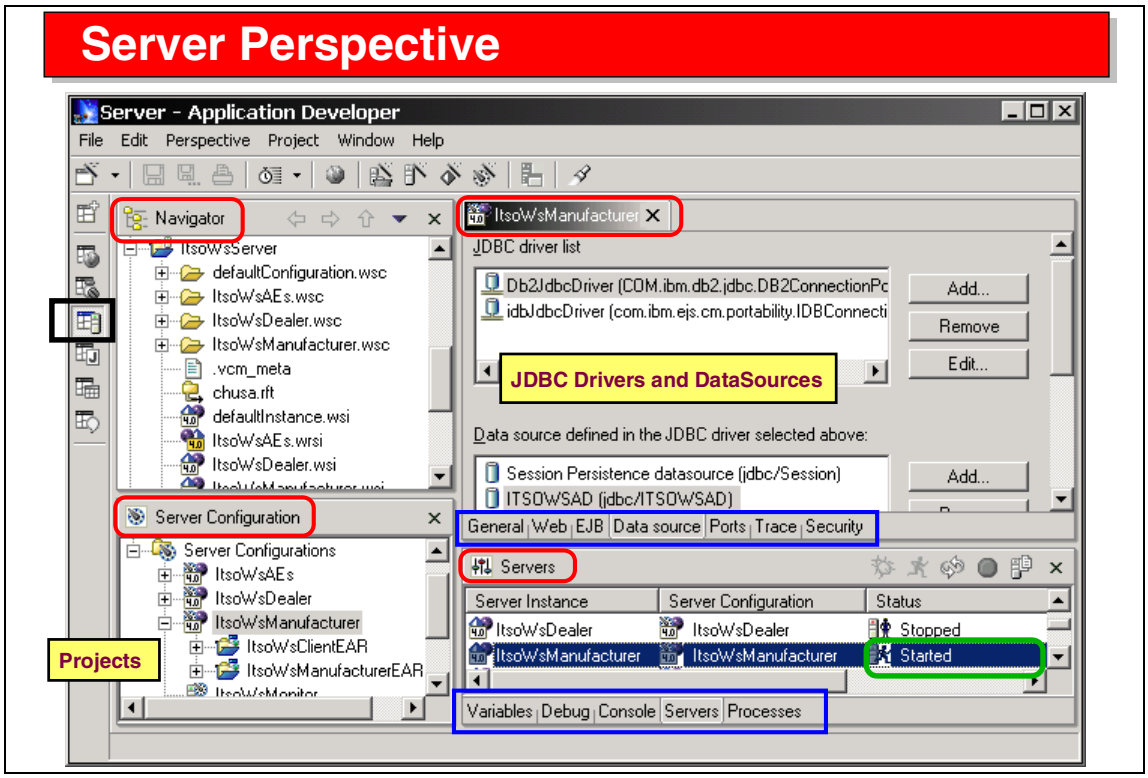
Visual 2-26 Data Perspective

The Data Perspective provides views and tools for definition and maintenance of descriptors for database, schemas, and table definitions.

The Data view shows database descriptors (XMI files) in a hierarchical format. Editors are provided to define database, schema, and table descriptors. Tools are provided to generate SQL DDL files for such descriptors, or to create a descriptor from an existing DDL file.

The DB Explorer view enables connections to database definitions in real time, and enables you to import existing descriptors as local resource files (in XMI format).

SQL statements can be defined graphically, using a wizard. Such statements can be executed for testing, and can be used by XML tools to convert database data into XML files. Statements can also be used by Web development tools to create skeleton Web applications (servlets and JSPs) that access databases.



Visual 2-27 Server Perspective

In the Server Perspective we maintain definitions of application servers for testing of Web applications, EJBs, and Web Services.

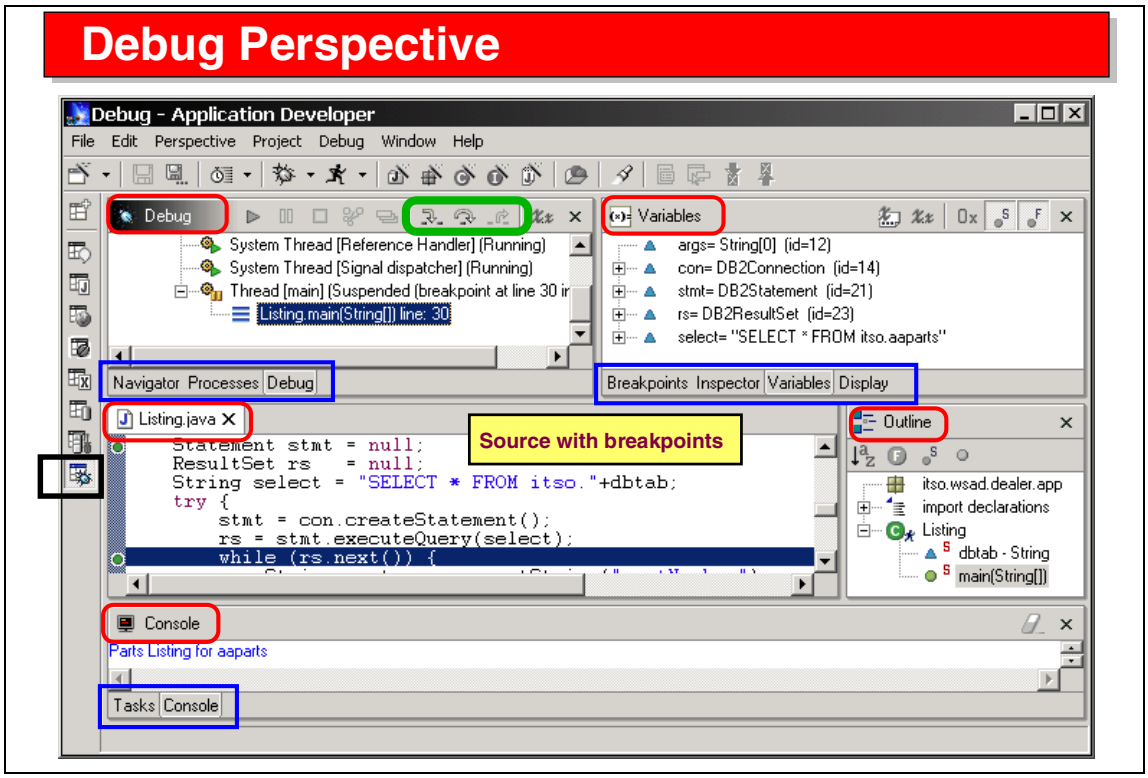
Server configurations define the type of server (WebSphere or Tomcat), and are configured with JDBC drivers and data sources.

Projects are associated with servers. When a server is started, the associated projects are loaded and their code can be executed.

Servers can be started in normal or debug mode. In debug mode, breakpoints can be placed into servlets and JSPs for debugging purposes.

Icons are provided to create a server project or server instances and configurations.

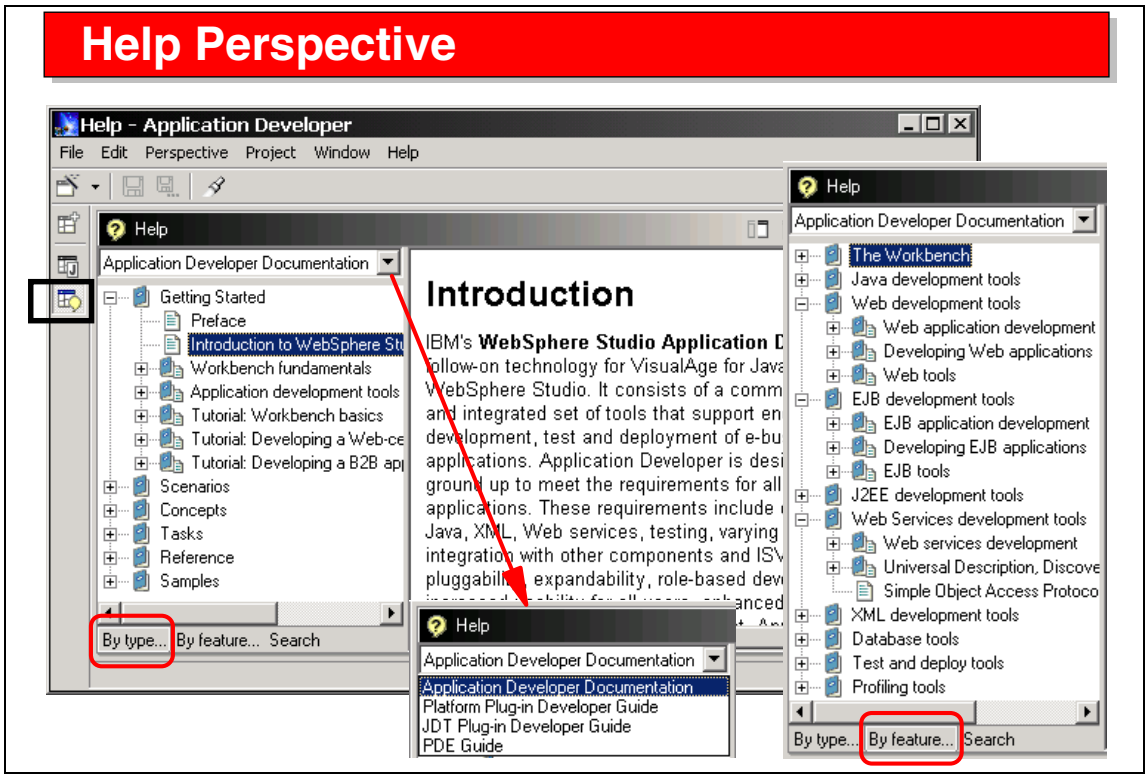
You can use the Server perspective to edit resources and run or debug projects.



Visual 2-28 Debug Perspective

The Debug perspective enables debugging of Java code.

Views are provided to display the source code (where breakpoints can be set), running processes, variables (with their values at the current breakpoint), evaluated expressions, and an inspector for in-depth analysis of data.



Visual 2-29 Help Perspective

Online help is provided through the Help Perspective.

Documentation can be accessed by type (shown at left), or by feature (shown at right).

The help facility also covers the underlying Workbench with a platform plug-in developer guide, a Java Development Tools (JDT) guide, and a PDE guide.

Workbench Key Features

Performance

Customizable perspectives

- Role-based development (Web developer, Java developer, DBA)
- Use same project artifacts (files)

Pluggable development environment

- Java and ActiveX plug-in support
- IBM and ISVs use same architecture to extend the Workbench

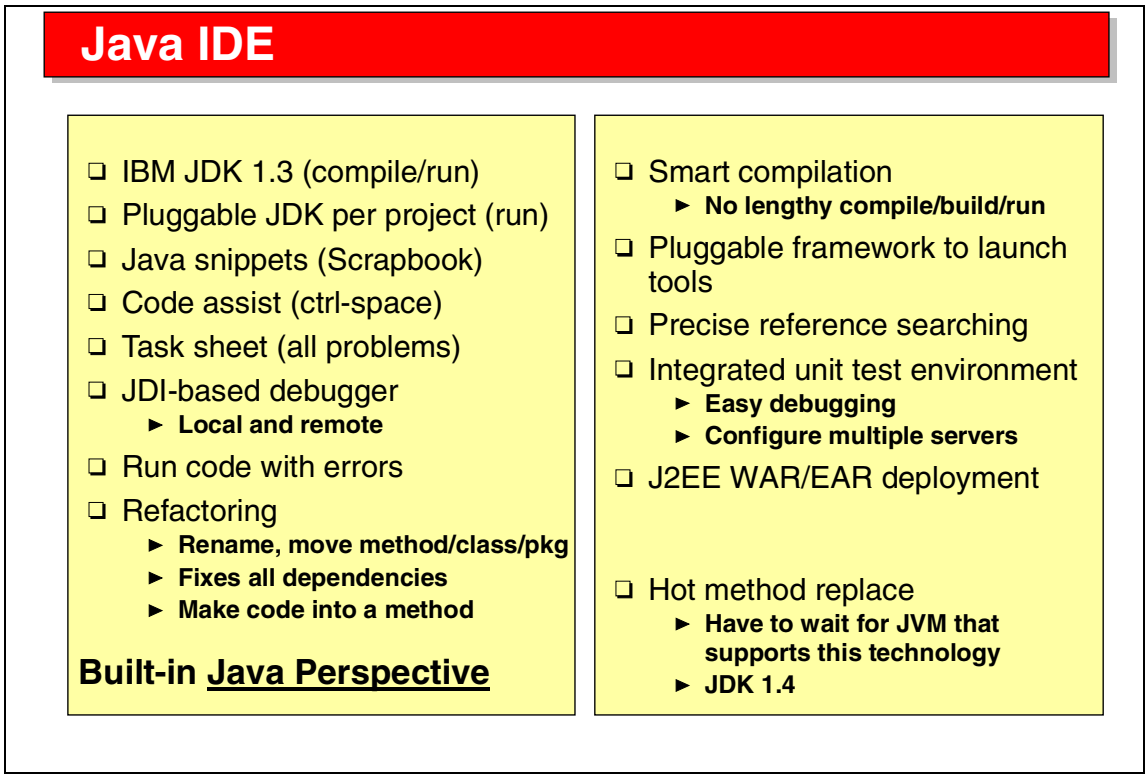
Support for automated builds

- Ant support
- Command line EJB generation

Visual 2-30 Workbench Key Features

Key features of the Workbench include:

- Customizable perspectives for role-based development
- Pluggable environment that enables other tool vendors
- Automated build facilities through the built-in Ant support

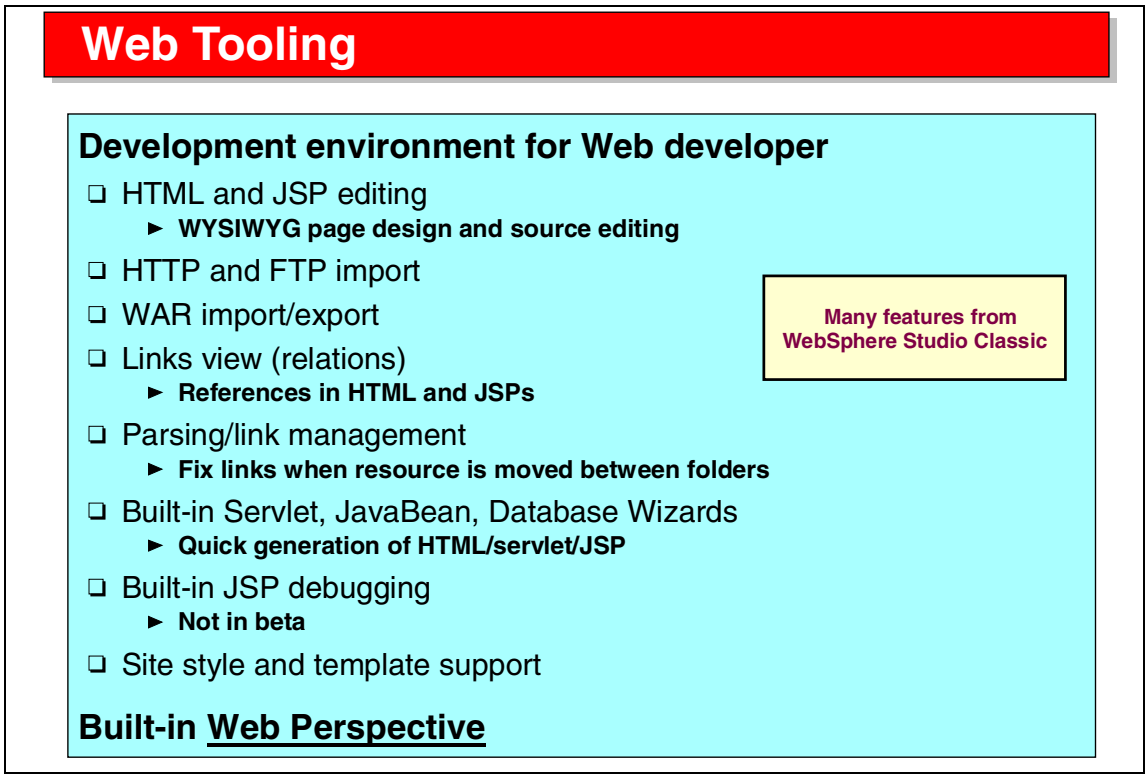


Visual 2-31 Java IDE

The Java integrated development environment (IDE) is built on top of the IBM JDK 1.3 that is used for all compilations and as a default runtime JRE.

Facilities include:

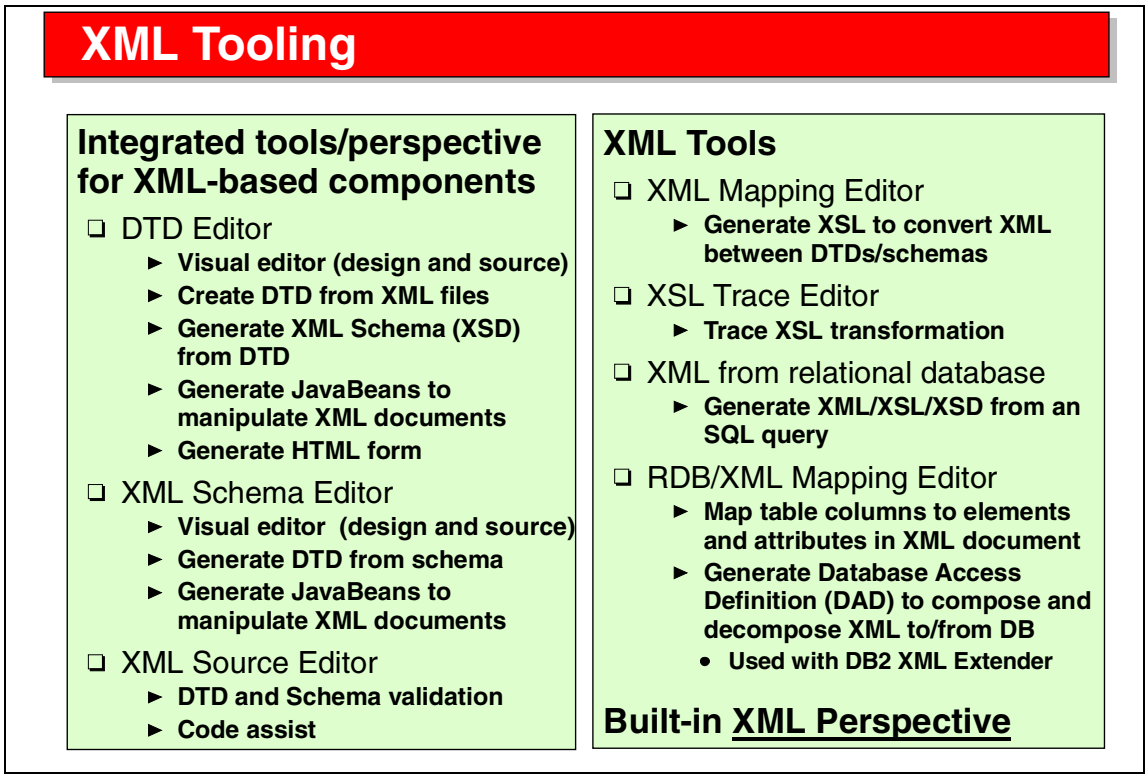
- Pluggable JDK—a JDK can be assigned to each project for running its code
- Scrapbook—files with Java snippets that can be executed
- Code assist—entering ctrl-space displays the valid functions
- Tasks view—displays all problems
- Debugger—for local or remote debugging
- Refactoring—rename, move, make new method with update of all references
- Search—by Java construct or text
- Build—smart, incremental
- Test environment—application or server



Visual 2-32 Web Tooling

The Web tooling support includes:

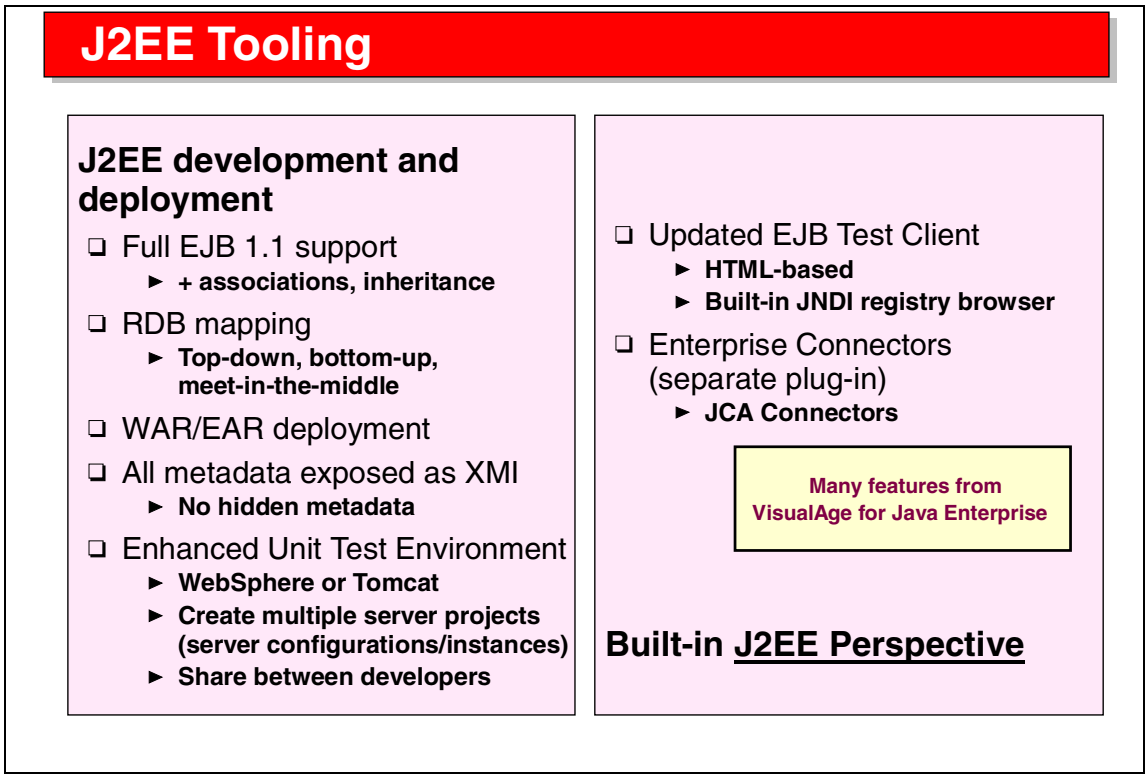
- HTML and JSP editing with the Page Designer
- Site import using FTP or HTTP
- WAR import (existing Web applications) and export (to application server)
- Links view and management of links between related files
- Wizards to generate servlets and JSPs for database and JavaBean applications
- JSP debugging at source code level



Visual 2-33 XML Tooling

The XML tooling support includes:

- Editors for XML files, DTDs, and XML schemas
- Code assist (ctrl-space)
- Utilities to generate XML from DTD, DTD from XML, schema from DTD, DTD from schema, JavaBeans from DTD or schema
- XSL mapping to convert XML files between two DTDs or schemas, including a trace editor to display XSL execution
- Generate XML and HTML from database access through a query
- Mapping between XML and relational databases for usage with the DB2 XML Extender (generate database access definition (DAD) to compose/decompose XML data into table columns)



Visual 2-34 J2EE Tooling

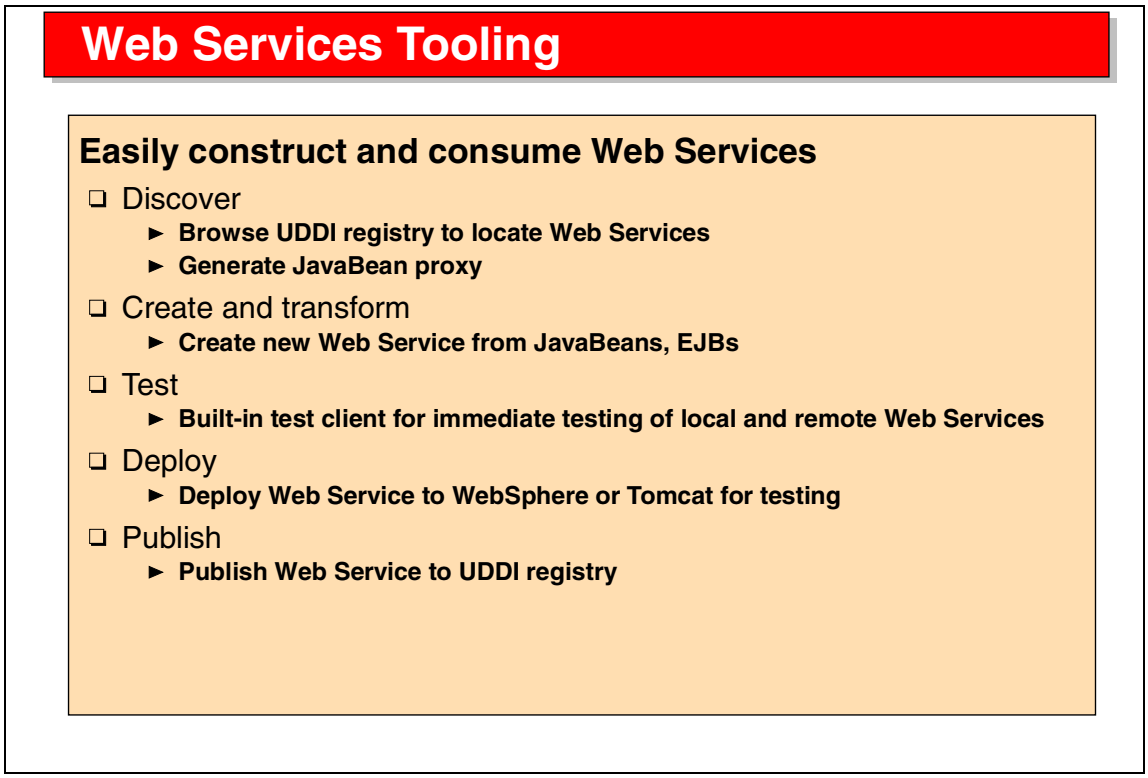
J2EE tooling enables full J2EE 1.2 support including EJB 1.1 specification.

Full EJB 1.1 support is provided with IBM extensions for associations, inheritance, custom finders, and access beans. The mapping between entity EJBs and relational tables can be performed top-down (tables from EJBs), bottom-up (EJBs from tables), or meet-in-the-middle. EJB deployment information is kept in XMI files and is not hidden, as in VisualAge for Java.

WebSphere Application Server AEs is built into the Application Developer for testing, and Tomcat is supported as well (but must be installed). The test environment supports servlets, JSPs, and EJBs (EJB only on WebSphere).

A new universal test client (UTC) is provided for testing of EJBs (and for Web Services).

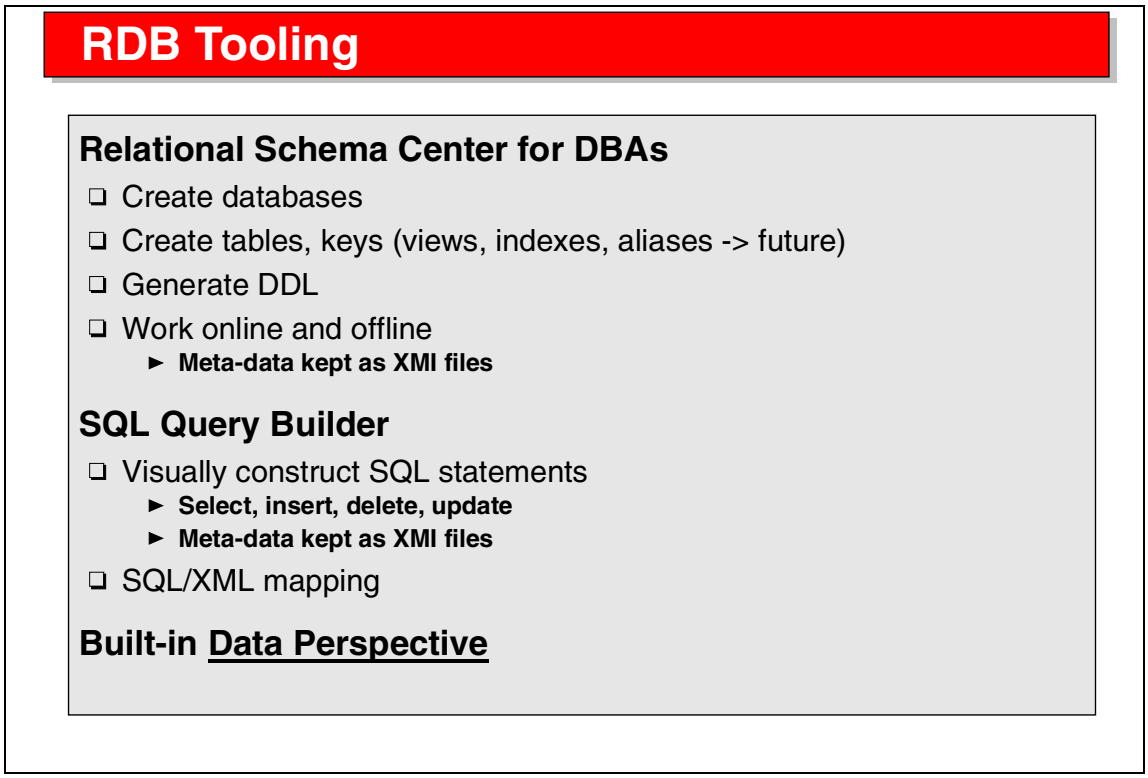
Deployment is made very easy. EAR projects collect Web and EJB projects, and can easily be exported into an EAR file for deployment into a J2EE-enabled application server.



Visual 2-35 Web Services Tooling

The Web Services tooling enables creating, testing, and consuming of Web Services. This includes:

- Interaction with a UDDI registry to browse the registry and retrieve Web Service specifications, or publish new Web Services
- Create Web Services for existing applications based on a JavaBean or an EJB
- Test Web Services using the universal test client
- Deploy Web Services to an application server

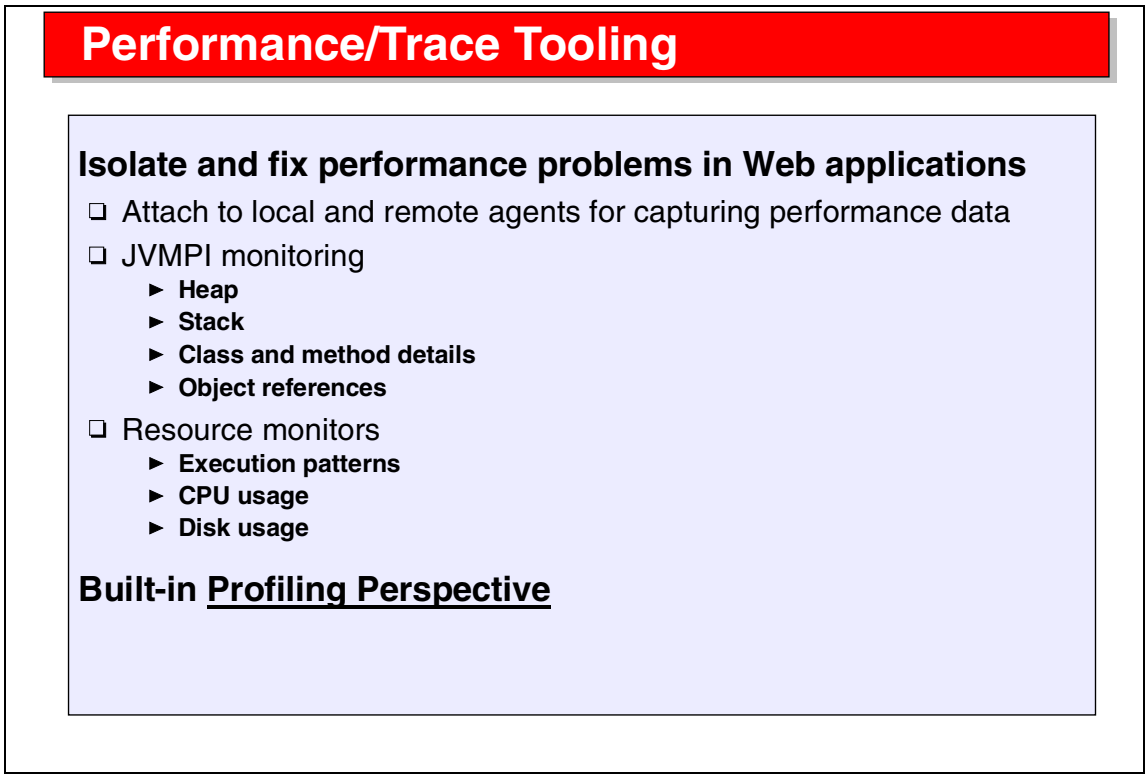


Visual 2-36 RDB Tooling

RDB tooling is based around the relational schema center, which includes:

- Interactively create database, schema, and table descriptors.
- Retrieve existing database, schema, and table descriptors by connecting to a database.
- Generate the DDL from descriptors.
- Execute DDL to create descriptors.
- Create and execute SQL statements. Together with the XML tooling, SQL results can be converted into XML and HTML.

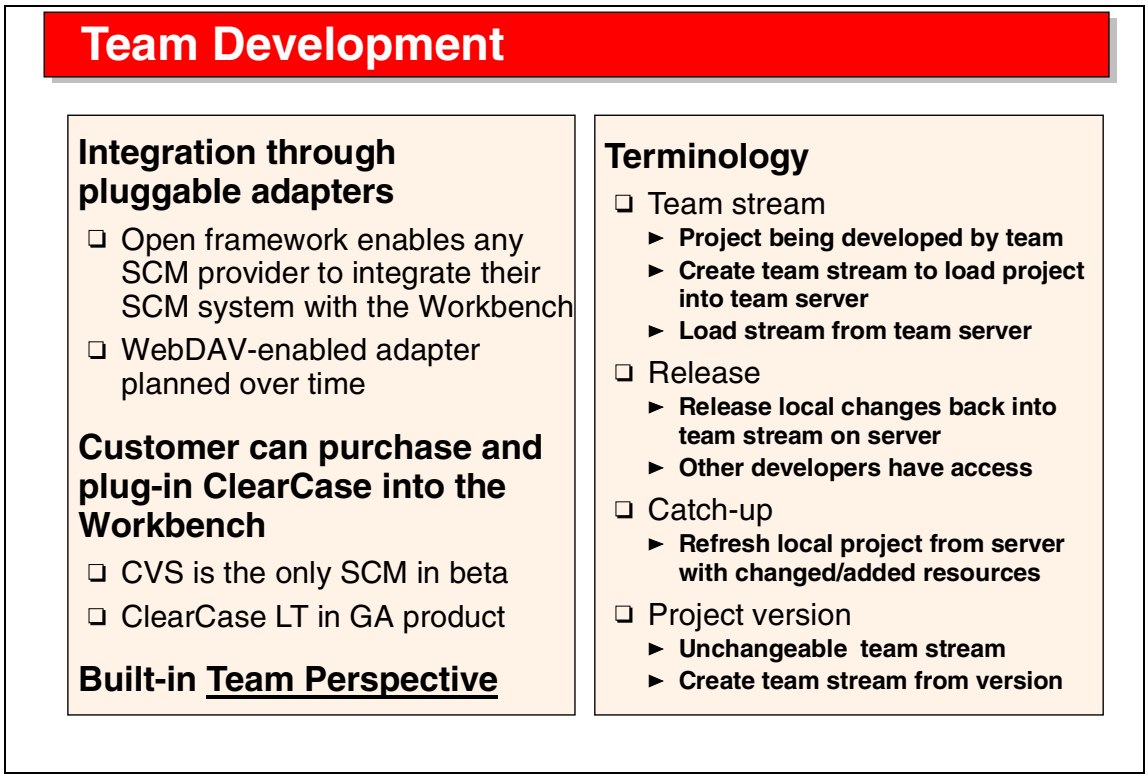
Descriptors are kept in XMI files, and editors are provided to maintain the descriptors.



Visual 2-37 Performance/Trace Tooling

Performance and trace tooling enables the analysis of Java applications by measuring the time spent in classes and individual methods of these classes.

Events are captured in a local or remote Java Virtual Machine and forwarded through an agent to the Application Developer for analysis. Results are displayed in tables and graphically.



Visual 2-38 Team Development

The Studio Workbench provides extension points for configuration management products for team development.

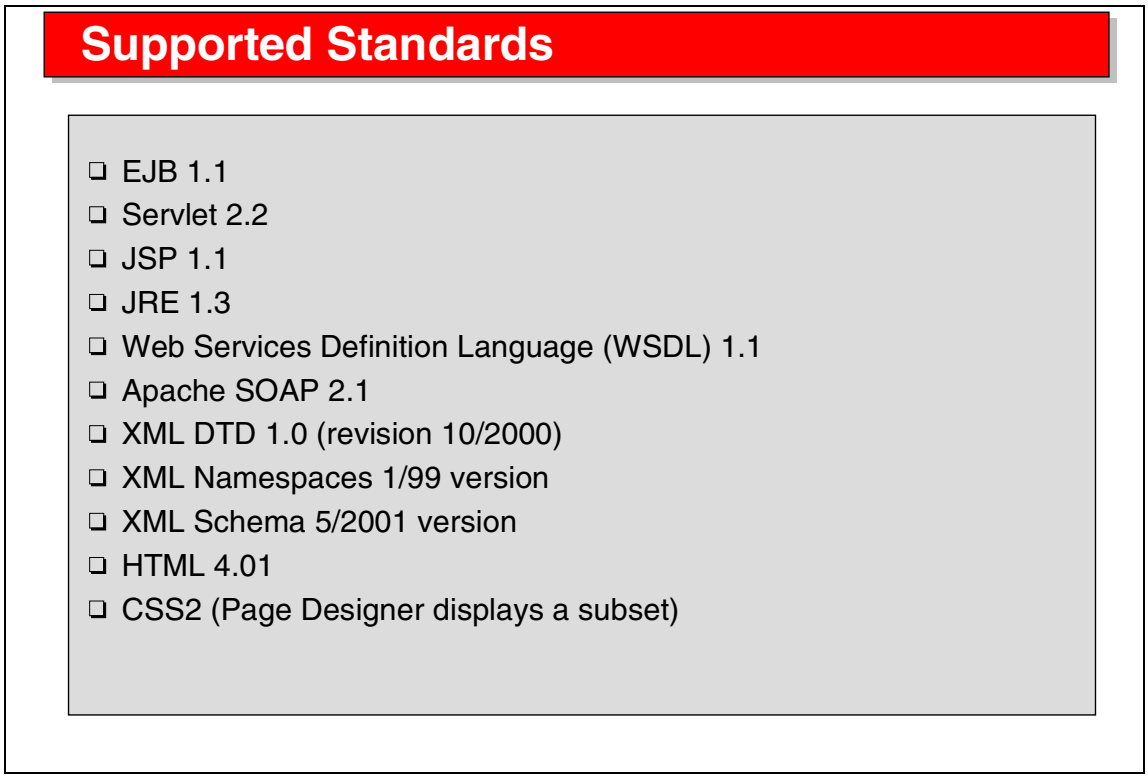
The Application Developer ships with support for two underlying team products, CVS and ClearCase LT. Full-function ClearCase support is available from Rational.

Projects can be assigned to streams in a shared repository, and multiple developers can work on a shared project.

A developer can synchronize a project with the project in the team repository:

- Changes made by the developer can be released to the team stream
- Changes made by other developers can be added to the developer's project (catch-up)

Projects can be versioned.



Visual 2-39 Supported Standards

This is the list of supported standards and specifications in the Application Developer.

Summary

Everyone integrates

- Integration at industry level enables tools to interoperate
  - **Open standards**
  - **Integration in vendor's lab (not on developer workstation)**
- Entire industry can integrate and interoperate
  - **Modeling: business process/applications**
  - **Content: Web pages, graphics**
  - **Business logic: Java, JavaBeans, EJBs, servlets**
  - **Application assembly: J2EE, scripting, rule-based**
  - **Operation: site management**
- IBM takes application development to a higher level

WebSphere Studio Application Developer

Leading Web/J2EE Development Environment
and Integration Platform for IBM and Vendors

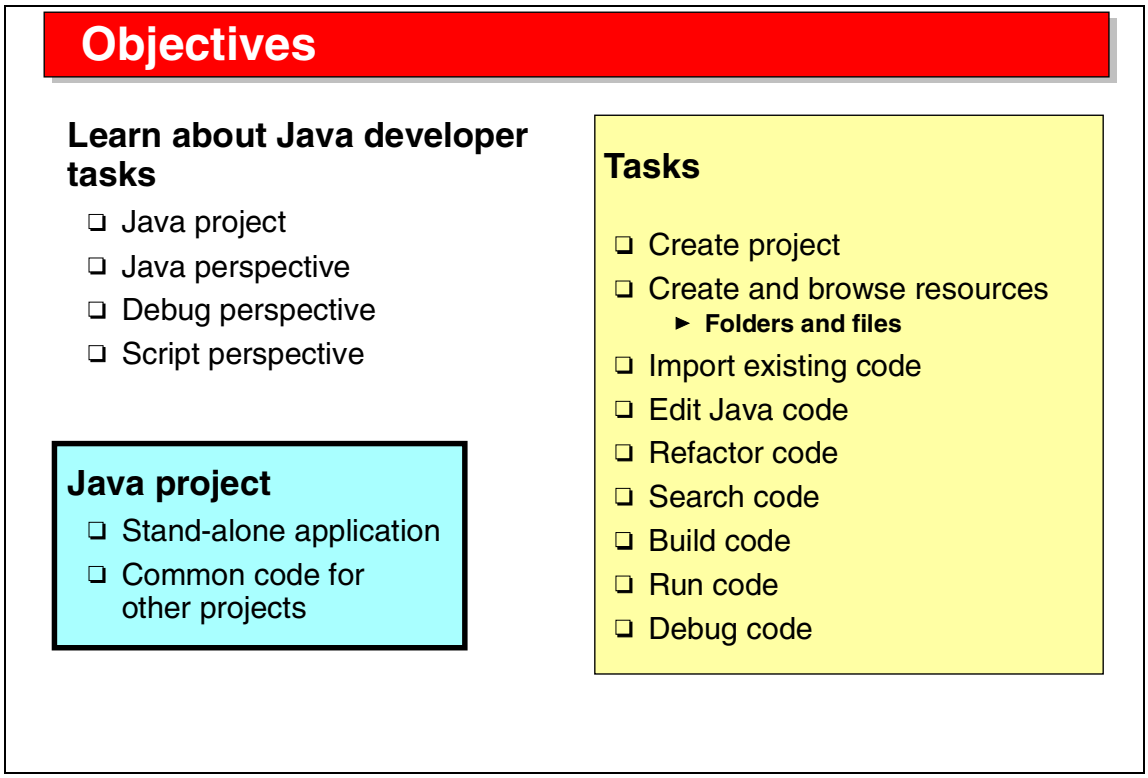
Visual 2-40 Summary

The Application Developer provides the basis for integration of application development for Java, Web site, Web application, EJB, Web Services, and XML development.

3

Application Developer: Java Development

Visual 3-1 Title



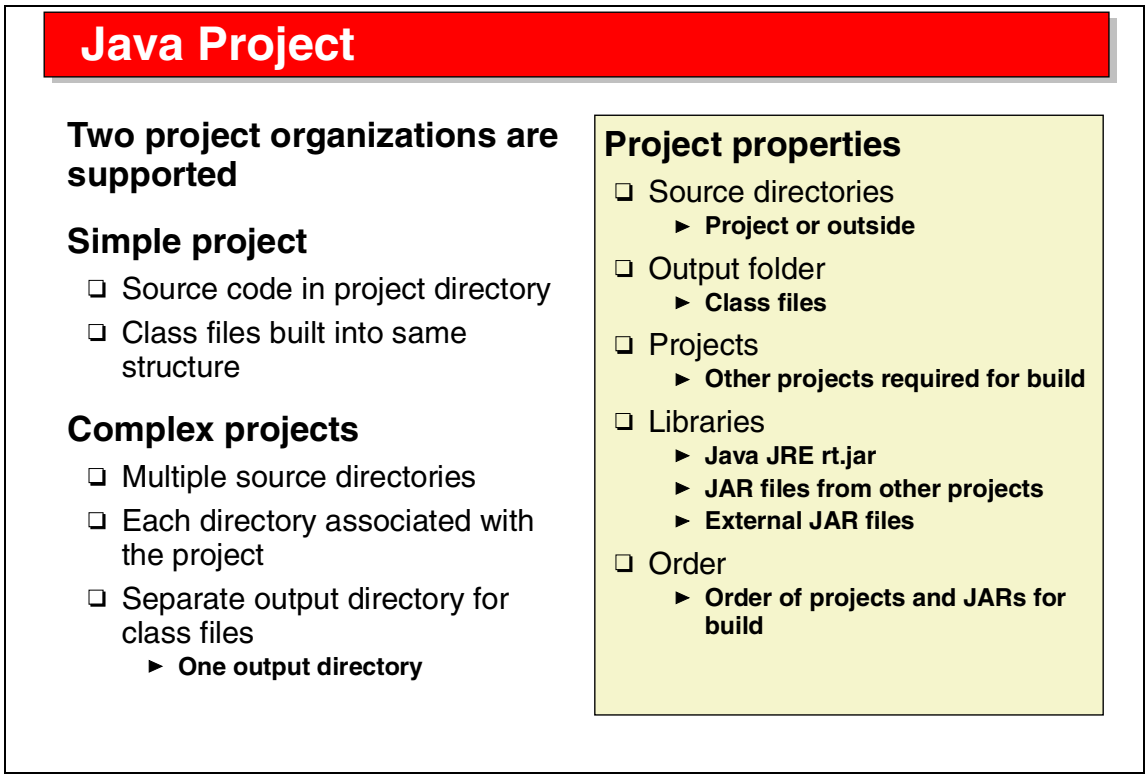
Visual 3-2 Objectives

The objectives for this unit are to understand the development of Java application with the Application Developer.

A Java project is used to develop stand-alone Java applications, or to develop base Java code (JavaBeans, utilities) that are shared between multiple higher-level projects (Web, EJB, XML projects).

A Java developer will use the Java perspective to edit and build (compile) code, the Debug perspective to test code, and the Script perspective to work with small scrapbooks.

The tasks involved in Java development are listed on the right side, starting from creating a Java project as the base for all development activities.



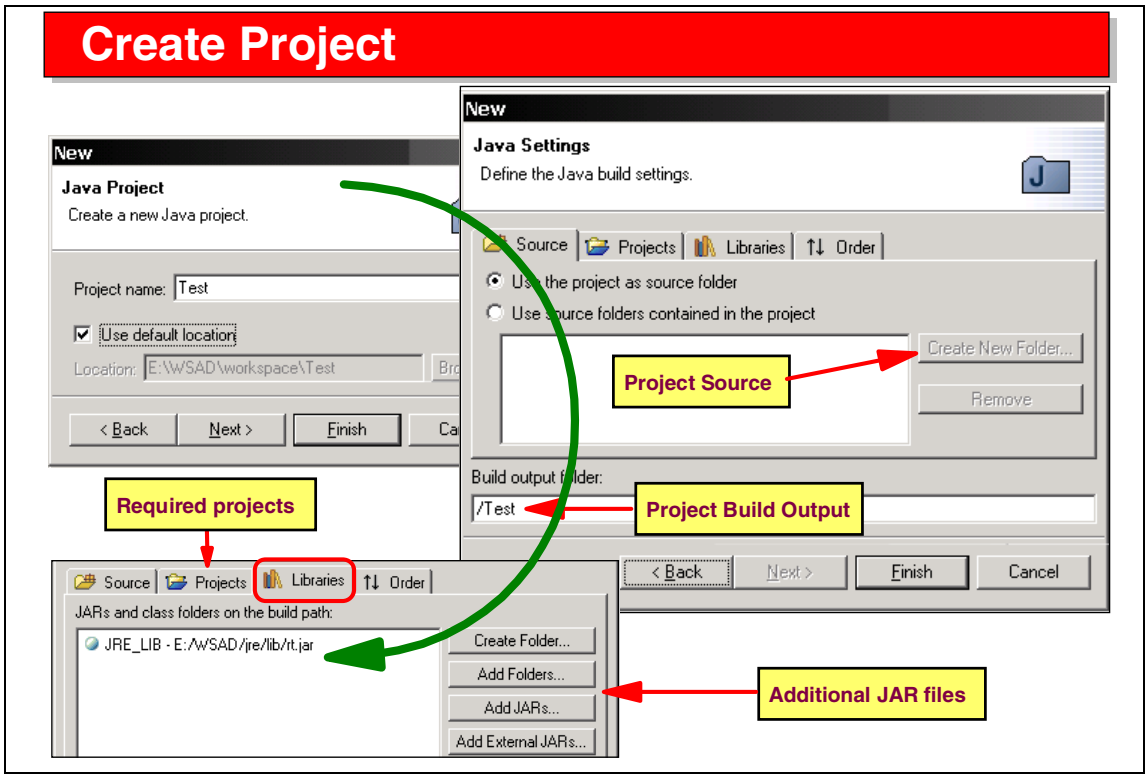
Visual 3-3 Java Project

The resources (files) of a Java project can be kept in one place (simple) or spread over multiple locations on the hard drive(s):

- In a simple project, all files are kept in one main project folder and subfolders. Compiled class files are kept together with the Java source files in this organization.
- In a complex project, Java source files are kept in multiple folders (and subfolders), but all the compiled class files are kept in one output folder.

One of the main project properties is the setup of the build path used to compile the Java source files. The build path includes the Java runtime (JRE) and:

- Other projects
- JAR files from other projects or external JAR files (for example from underlying products, such as JDBC drivers from a relational database, connectors)



Visual 3-4 Create Project

A SmartGuide is provided to create a Java project:

- A name must be provided.
- The default location can be set for a simple project.
- For a complex project, the source folders can be associated with the project.
- The build output location (for class files).
- The build path, consisting of additional projects and libraries (JAR files).
- The order in which the projects and JAR files are to be used.

JAR files can be copied into the project, or they can be referenced (point to an external location). Variables can be set up for reference JAR files, which makes it easier for team development where the actual directory may differ between multiple developers' work stations.

Create Project Resources

Creating resources

- Create package
  - **Creates directories according to package name**
  - **Shows package name in view**
- Create class

  `SmartGuide: specify`
  - **Superclass**
  - **Interfaces to implement**
  - **Method stubs:**
    - **main, constructors**
    - **inherited abstract methods**
- Create interface
  - **Specify extended interfaces**
- Create Scrapbook page (.jpage)
  - **Try sample code**

Importing resources

- From directory
- ZIP file
- JAR file

Migration from VisualAge for Java

- Export into directory or JAR file
- Import into WSAD

Missing:

- Visual Composition Editor
- BeanInfo page

Visual 3-5 Create Project Resources

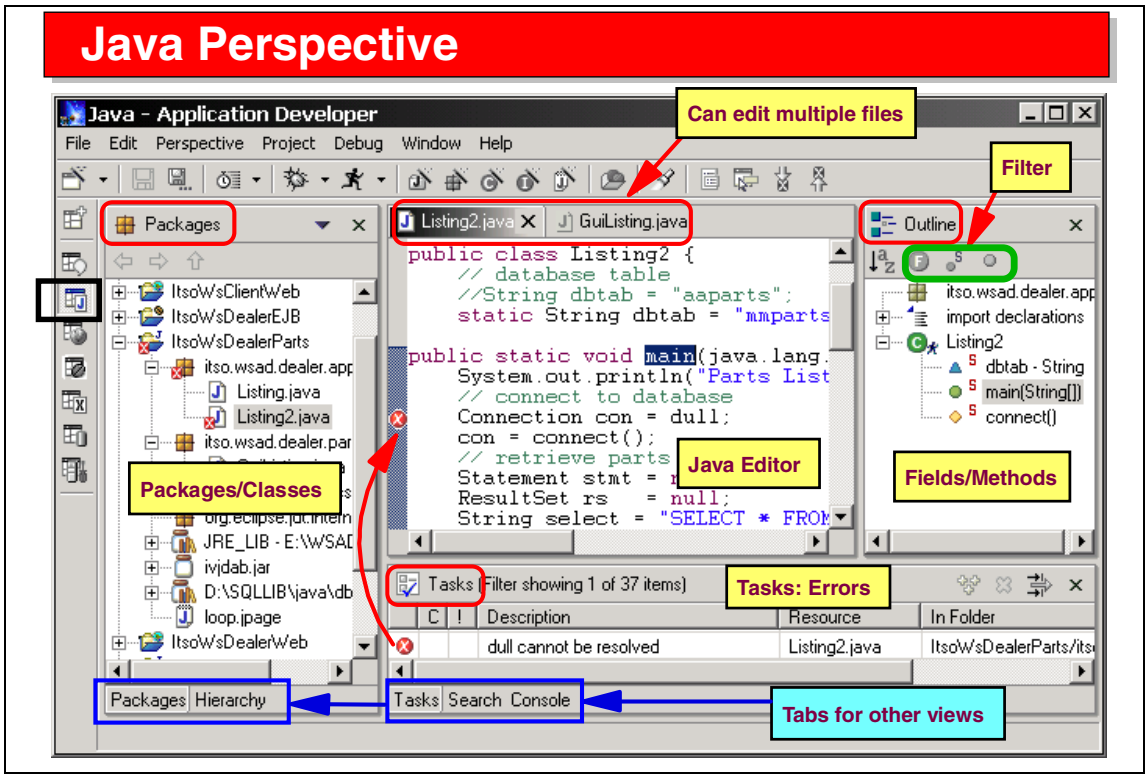
Project resources in a Java project are mainly Java source files, organized by Java packages. The package view of the Java perspective shows the fully qualified package names, instead of showing subfolders for each level.

SmartGuidse are provided to create Java classes and interfaces. For a Java class, you can specify which interfaces are implemented and what method stubs should be generated.

A Java scrapbook is a file named .jpage. It can contain code fragments that can be executed.

Existing resources can be imported into a Java project. Individual files, whole directories, or ZIP and JAR files can be imported.

To migrate existing code from VisualAge for Java, export the code into a directory or JAR file, then import into the Application Developer.



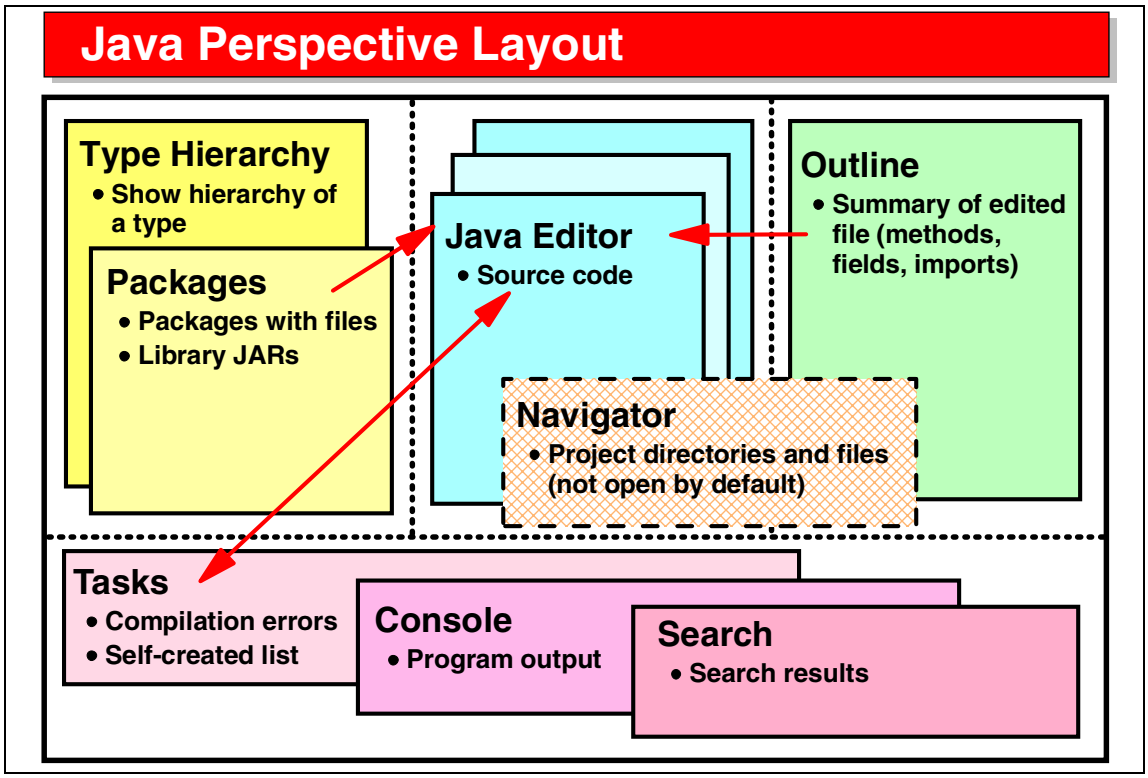
Visual 3-6 Java Perspective

The Java Perspective contains four panes by default:

- Left—Packages and Hierarchy view
- Middle—reserved for editors (multiple files can be edited)
- Right—Outline view (shows outline of currently selected file in editor pane)
- Bottom—Tasks (error messages), Search (result of search operations), and Console (program output) views

Notice the icons in the toolbar:

- New icon to create resources or run tools
- Debug icon to run a class in debug mode
- Run icon to execute a class
- Create icons for project, package, class, interface, scrapbook
- Open any type in the editor (enter a name in a dialog)
- Search (open search dialog)
- Icons for editing and for jumping between errors



Visual 3-7 Java Perspective Layout

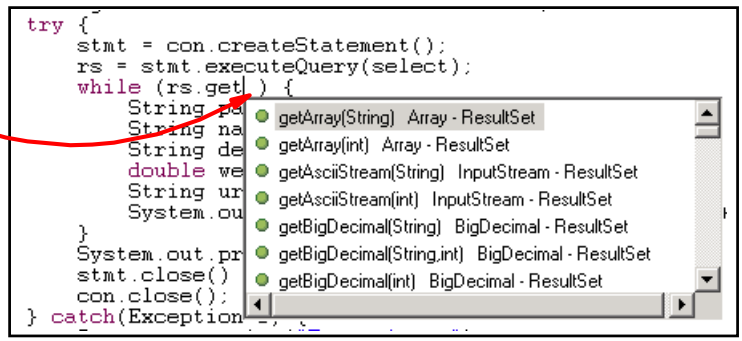
This diagram shows the default organization of the Java perspective:

- The Packages view shows the resources organized by packages.
- The Navigator view shows the resources organized by folders and subfolders. (This view is not open by default, but can be added.)
- The Type Hierarchy view can be opened for a selected type to show sub/super classes.
- The Search view shows the results of search operations.
- The Console view shows the output when executing programs.
- The Tasks view shows compilation errors or self-created tasks lists. Double-clicking on an error opens the Java source file with the error.
- Editor windows show Java source code.
- The Outline view shows the outline (imports, class, fields, methods) of the currently selected editor window. Clicking on an item in the outline positions the editor window.

Java Editor

- Bound to outline view
  - **Select field/method in outline, positions and highlights source in editor**
- Syntax highlighting
  - **Keywords, comments, Strings, Javadoc comments**
- Hover help on fields and methods
- Editing icons
  - **Show source of selected element only**
  - **Show hover help, goto next/previous problem**

- Code assist (ctrl-space)
- Bookmarks
  - **Add Bookmark on line or file**
  - **Open Bookmarks view**



Visual 3-8 Java Editor

The Java editor works in conjunction with the Outline view. Selecting an item in the outline positions the editor window to that part of the code. You can also choose to only show the selected item in the editor window, instead of showing the whole file. Icons allow you to filter out fields, static, or public members.

The Java editor uses colors to highlight keywords, strings, and comments.

When holding the mouse cursor over a variable or method, its definition is displayed as hover help text.

Ctrl-space invokes the code assist feature, which displays possible method calls that can be inserted at the selected point.

Bookmarks can be assigned to a line in the file, or to a whole file. From the Bookmark view (which must be opened), you can select a bookmark to open the associated file at the associated line of code.

Search

Text search

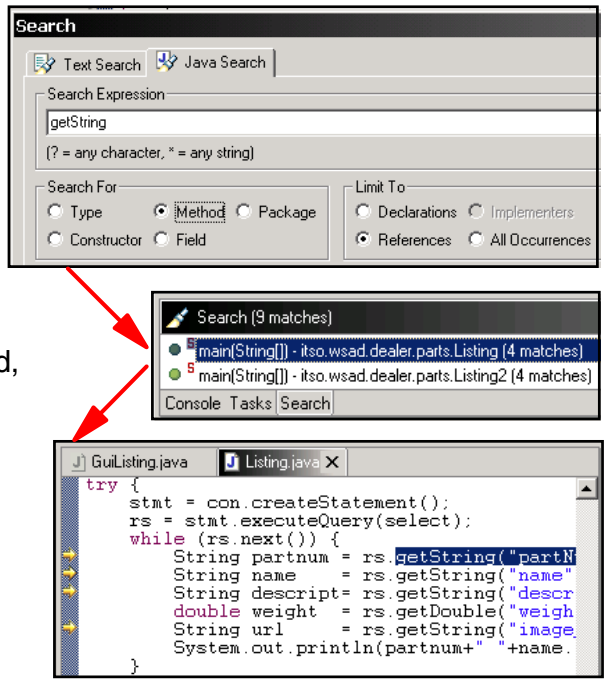
- Pattern match with wildcards
- Find any occurrence of text

Java search

- Based on Java language elements
- Faster than text search
- Search for type, method, field, constructor
- Limit to declarations, references, implementers

Results

- Search view shows files
- Open file and yellow arrows show matches



Visual 3-9 Search

Two types of searches are supported:

- Text search scans the whole file to find any matching given string.
- Java search only searches for Java constructs (type, constructor, method, field), and you can search for declarations or references or both.

Search results are displayed in the Search view. Double-click on a result file and it opens in the editor, with yellow arrows showing the lines that match the search.

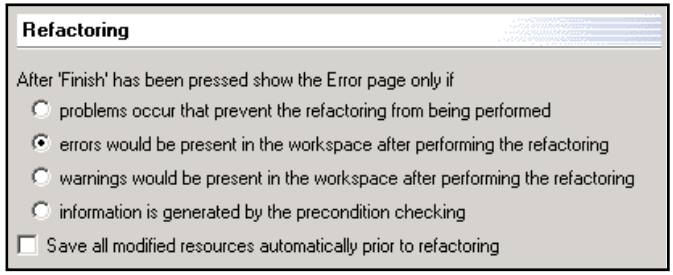
Editing Refactoring

Reorganize code but preserve semantics

- Rename of type, method, field
- Move file to another package (drag/drop or context menu)
- Change method parameter name
- Extract a method
  - **Select code and make it a method**
  - **Method call inserted at original location**

All references are automatically changed

- Preview of changes before commit
- Optional error page
  - **Window -> Preferences -> Java -> Refactoring**



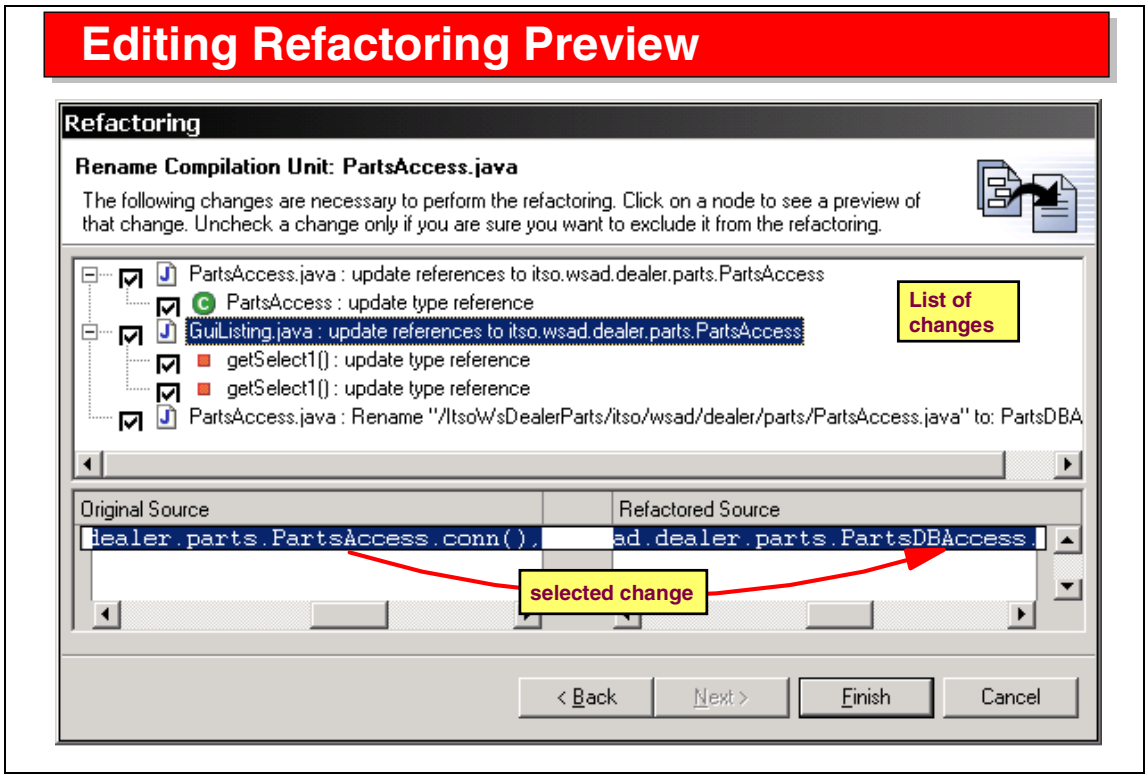
Visual 3-10 Edit Refactoring

The refactoring function enables a developer to make changes to files and have all references updated automatically.

Possible refactoring activities include:

- Rename a type, method, or field
- Move a type to another package
- Change parameters in a method
- Extract a part of an existing method as a new method, and insert a call into the existing method to invoke the new method

A preview window displays all the files that would be changed by the refactoring activity. Each update can be viewed individually and selected for change, or all changes can be applied.



Visual 3-11 Edit Refactoring Preview

This is an example of the preview window:

- The classes that require change are displayed in a list
- For each class, the individual line changes are displayed as a sublist
- Selecting an item in the list displays the original and changed source code in the bottom pane

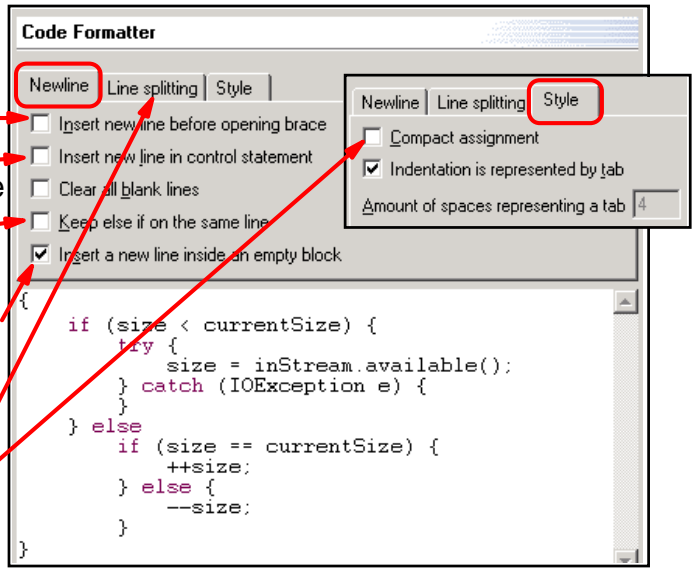
By default, all changes are applied when the Finish action is selected. Individual changes can be excluded by deselecting the update before the Finish action.

Code Formatting

Source code formatting controlled through preferences

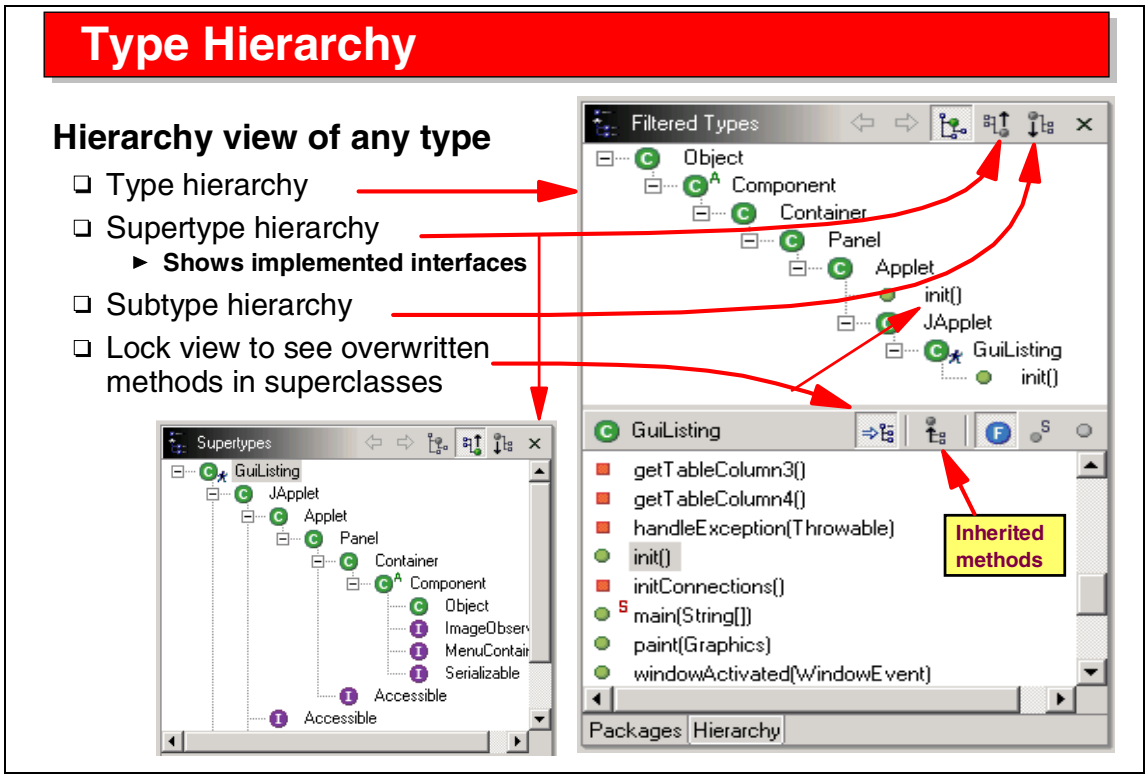
- Java -> Code Formatter

- ❑ Brace on new line
- ❑ catch/else on new line
- ❑ else if on same line
- ❑ Two lines for empty {}
- ❑ Maximum line length
- ❑ No space before =



Visual 3-12 Code Formatting

Java source code is formatted according to rules set up in the preferences dialog.



Visual 3-13 Type Hierarchy

The Type Hierarchy view can be opened for any type.

Icons are provided to display the hierarchy top-down (type hierarchy and subtype hierarchy) or bottom-up (supertype hierarchy).

The supertype hierarchy also shows interfaces that are implemented for each class in the hierarchy.

The lock view icon can be used to show from which classes a selected method is inherited. In our example, the init method (selected in the GuiListing class) is inherited from the Applet class.

Other icons can be used to display all methods (including inherited methods), or filter out fields, static, or public members.

Scrapbook

Create file of type Scrapbook

Script Perspective

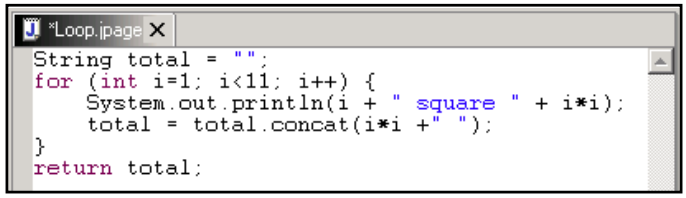
- name.jpage

Enter any Java code

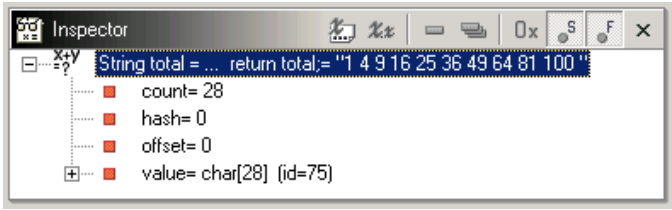
- Set package for execute

Select code and execute

- Display
  - **Result in edit view:**
- Inspect
  - **Result in Inspector view**
- Run ==> Console



(java.lang.String) 1 4 9 16 25 36 49 64 81 100



Visual 3-14 Scrapbook

A scrapbook is a file of type .jpage. You can enter any Java code in a scrapbook, select some or all of the code, and execute it.

The execution context can be set to any Java package. Execution can be to display the result in a pop-up window, to open an inspector window, or to just run with possible output in the Console.

Building Projects

Full build of all projects

- Project -> Rebuild All

Manual build of project (context menu)

- Build project ==> Incremental
- Rebuild project ==> Full

Automatic build on *save* of resource

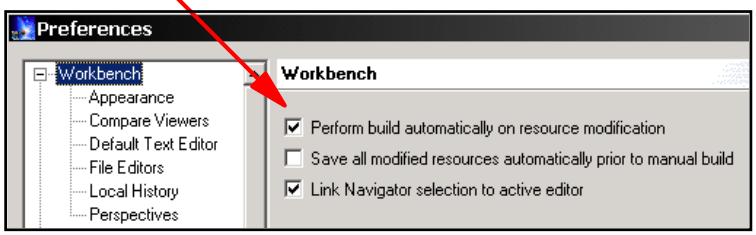
- Window -> Preferences -> Workbench

Build path defined for project

- Other projects
- JAR files in Workbench
- JAR files outside Workbench

Ant is built into Workbench

- Build files are XML
- Run Ant for XML build file
  - **Wizard prompts for target**



Visual 3-15 Building Projects

Building a project means to compile the Java source code.

By default, build is automatic every time you save Java code. Automatic build is set up in the preferences dialog.

With manual build, you can select to build only changed files or you can force a complete project build.

The most important activity is to set up the build path for the project, consisting of other projects, JAR files in the Workbench, and external JAR files (from other products). You can change the build path by opening the project's properties.

You can also use the built-in Ant feature to create your own XML build files. However, this task will not be explored in this class.

Debugging

Set breakpoints in code

- **Double-click**
- **Context in left border of editor**
- **Breakpoint on exception**
- **Breakpoint in inner classes**

Click Debug icon

Debug Perspective (next)

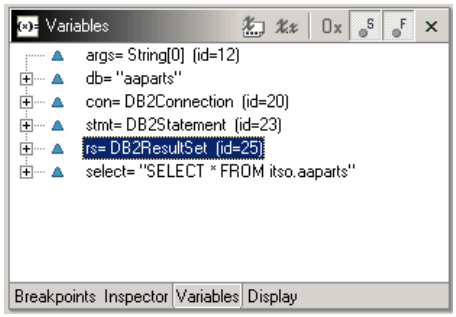
Debugger control icons

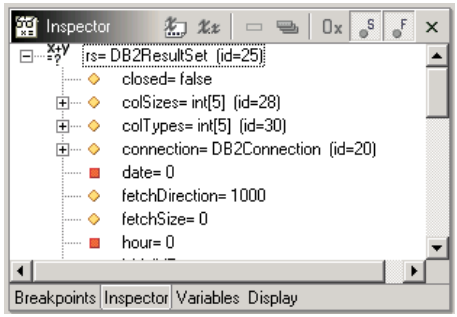
- ❑ Step into/over
- ❑ Run to return
- ❑ Resume

Variables and Inspector

Evaluate expressions (Display view)

Local and remote debugging





Visual 3-16 Debugging

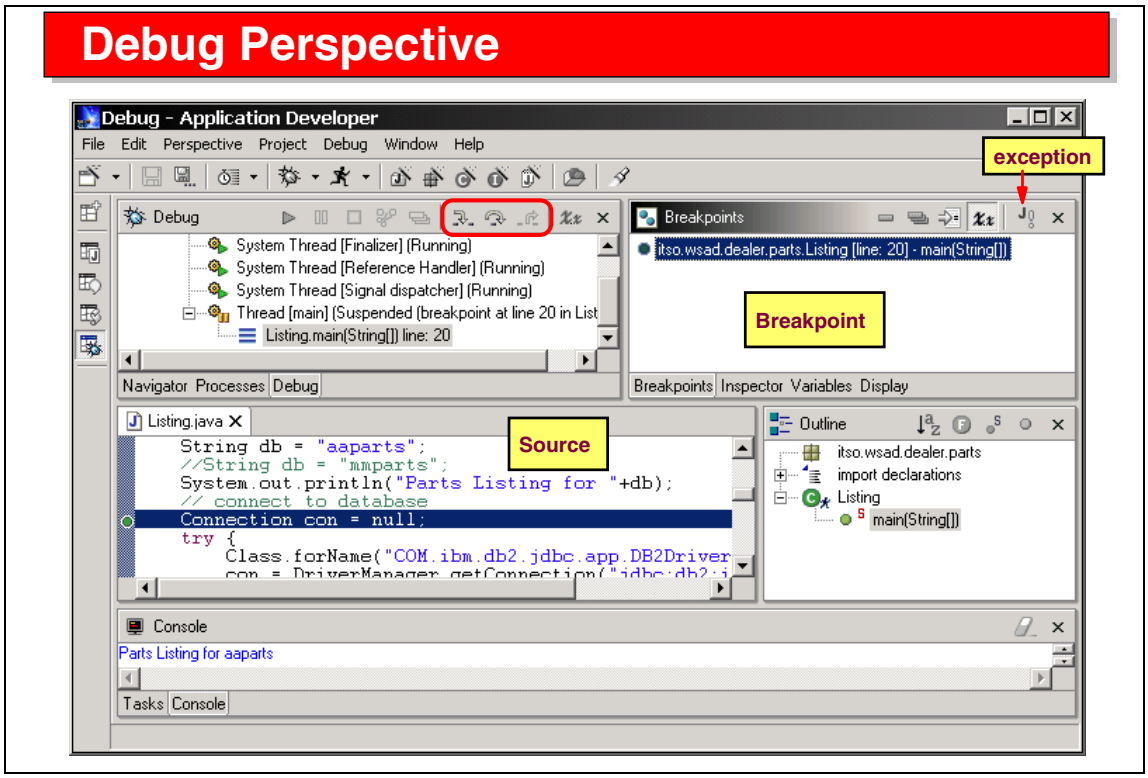
To debug Java source code, you set breakpoints in the source code. A breakpoint can be set on a line of code, for when a certain exception occurs.

To start a program in debug mode, use the debug icon, and the Debug perspective opens (next visual).

In the Debug perspective you use icons to step into a line of code, step over a line of code, or to run to the end of a method (step return).

The Variables and Inspector view can be used to analyze the values of variables.

Debugging is supported on the local machine (where the Application Developer runs) or on a remote machine (with the IBM Agent Controller installed).



Visual 3-17 Debug Perspective

The Debug perspective shows:

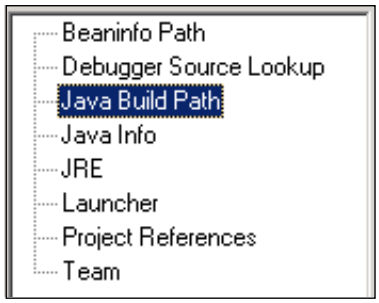
- The Processes view with running processes
- The source of the Java program with the current line
- The Outline view of the source code
- The Console output
- The Breakpoint view will all the breakpoints
- The Inspector and Variables views with the current values
- The Display view where expressions can be evaluated

For easy debugging, you may want to make the Variables view a separate window that you can move away from the Application Developer window.

Project Properties

Open project properties (context menu)

- Java build path
  - **Source, projects, JAR files, order**
- Alternate Java runtime (JRE)
  - **Default or custom**
- Default launcher for Run/Debug
- Project references (to other projects)
- Debugger source lookup
  - **Build classpath or custom lookup**
- BeanInfo path (enable introspection)
- Team
  - **Location of repository for project**



Visual 3-18 Project Properties

The project properties dialog can be used to set up:

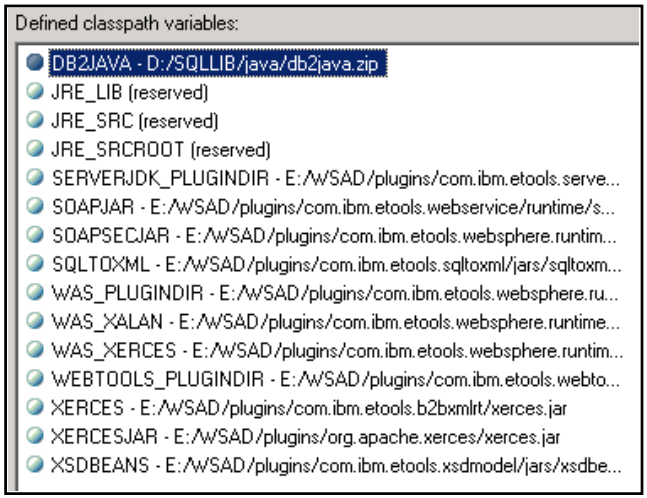
- The Java build path
- A custom Java runtime library
- The default launcher (launch a Java program or an application server)
- The location of source code, for example for JAR files that are used by the application
- The location of the team repository

Java Preferences

Window -> Preferences

System-wide properties

- Java classpath variables
  - **Locate external JAR files without specifying absolute directory**
- Code formatter
  - **Already discussed**
- Installed JREs
  - **Can add alternates and assign to projects**
- Java editor font
- Organize imports
  - **Can specify order of import statement by package names (wildcards)**
- Refactoring
  - **Already discussed**



Visual 3-19 Java Preferences

Many system-wide preferences for Java development are set up in the preferences dialog (*Window -> Preferences*). These include:

- Variable to refer to external JAR files instead of specifying directory locations (many variables that point to JAR files in the Application Developer itself are predefined)
- Code formatting rules
- Installed Java runtime libraries (to be associated with a project)
- Font used by the Java editor
- The organization (sequence) of import statements that should be used in Java code
- Refactoring defaults (when to display an error page)

Summary

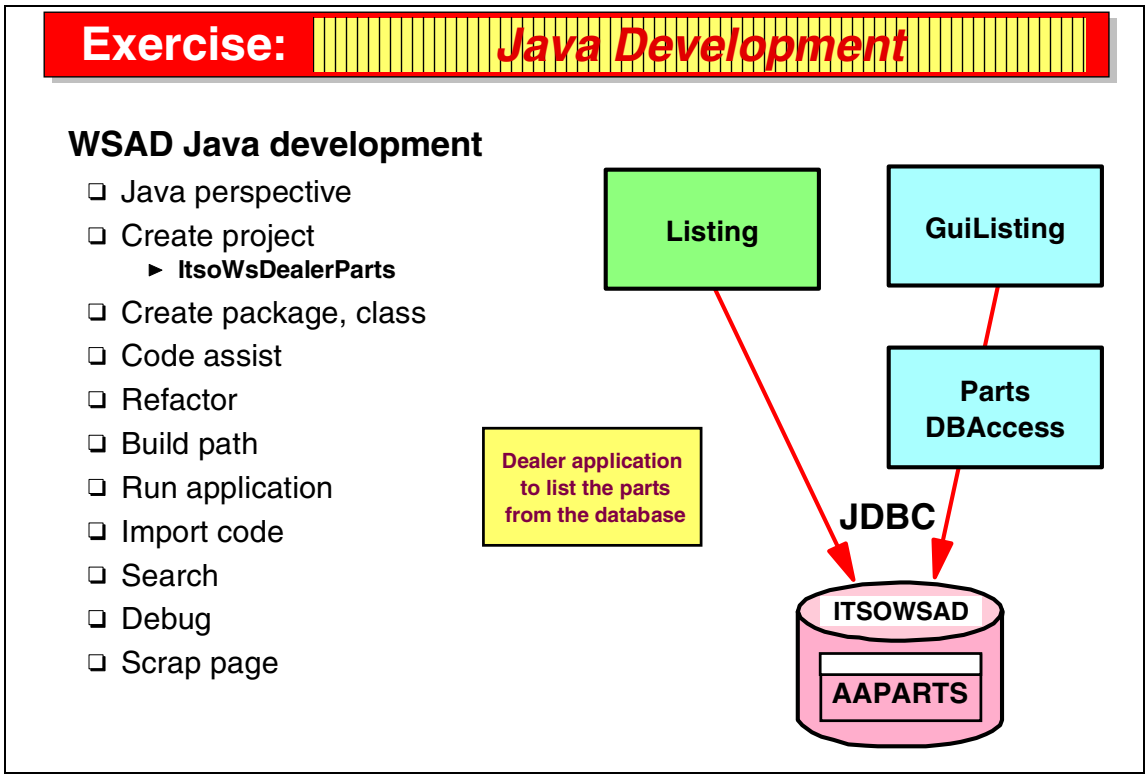
Java projects and Java development provide

- Flexible project arrangement
  - **Source directories entirely inside project**
  - **Source directories outside project**
- Powerful editing
  - **Code formatting and assistance**
  - **Refactoring**
- Debugging
  - **Step through code and inspect variables and expressions**
- Flexible build
  - **Automatic or manual**
  - **Full or incremental**
- Java and debug perspective provide easy access to all functions

Visual 3-20 Summary

Java projects are the base for Java code development.

Other project types, such as Web and EJB projects, use these basic facilities as well, because these projects also deal with Java code and require Java editing and compilation.



Visual 3-21 Exercise: Java Development

The Java development exercise guides you through many of the tasks discussed in the presentation.

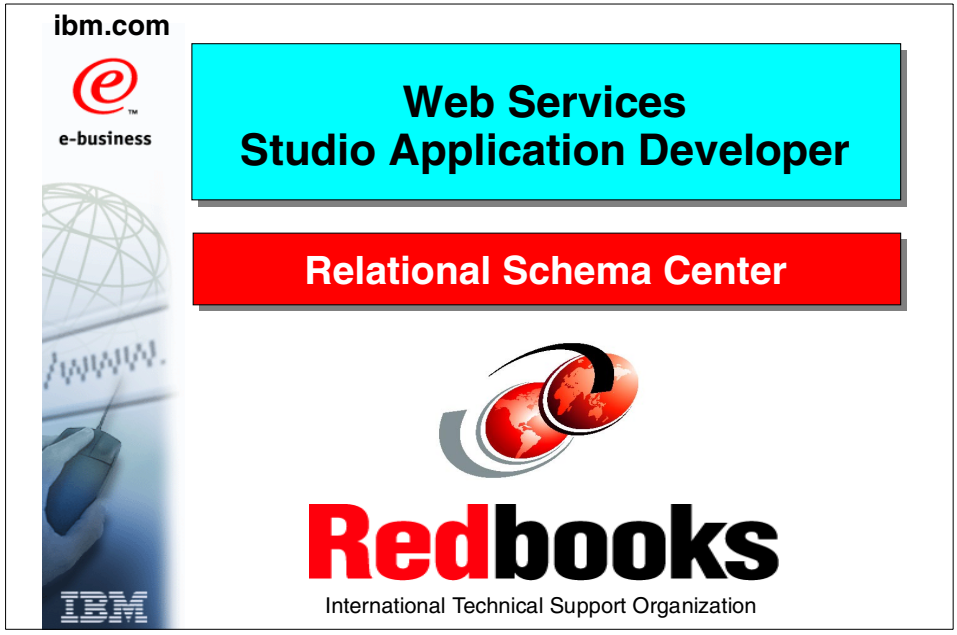
In this exercise you work with two Java programs:

- Listing, a simple Java program with direct JDBC access to the parts tables in the ITSOWSAD database
- GuiListing, a Java GUI program that was developed with VisualAge for Java and that you will import into the Application Developer. This program uses the data access beans of VisualAge for Java for database access, and you have to import the JAR file with the underlying support classes as well.

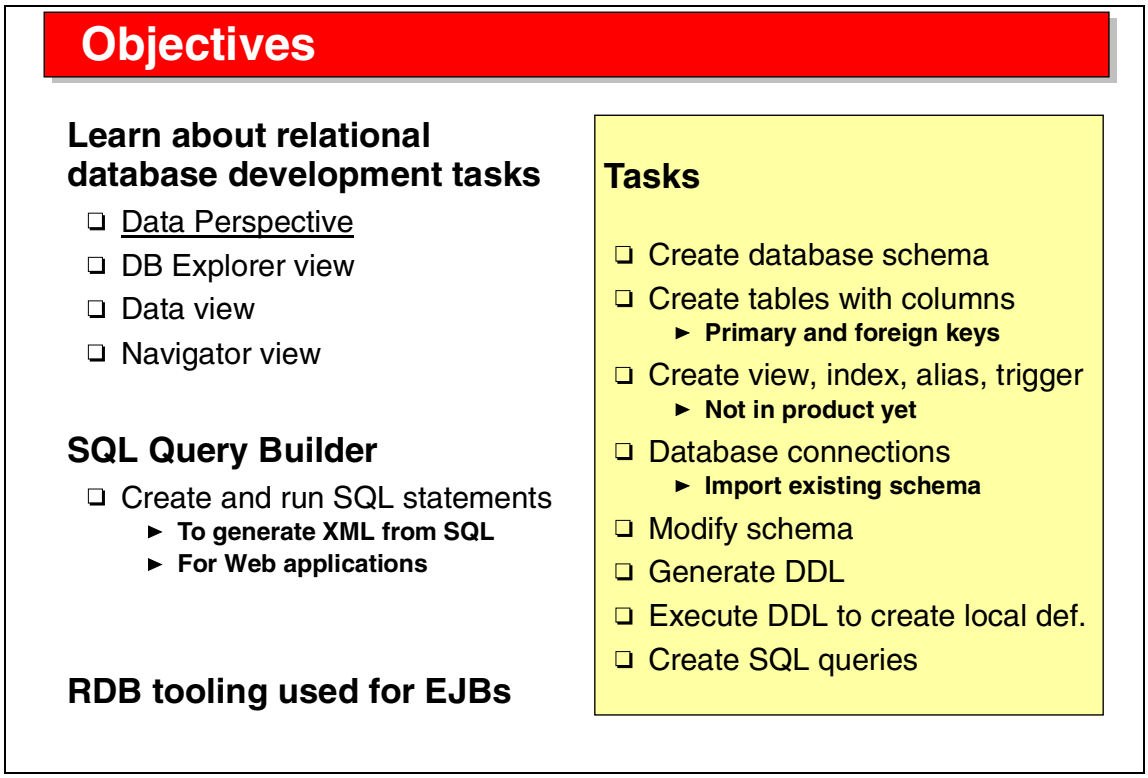
See Exercise 1, “Java development” on page 299 for the instructions for this exercise.

4

Application Developer: Relational Schema Center



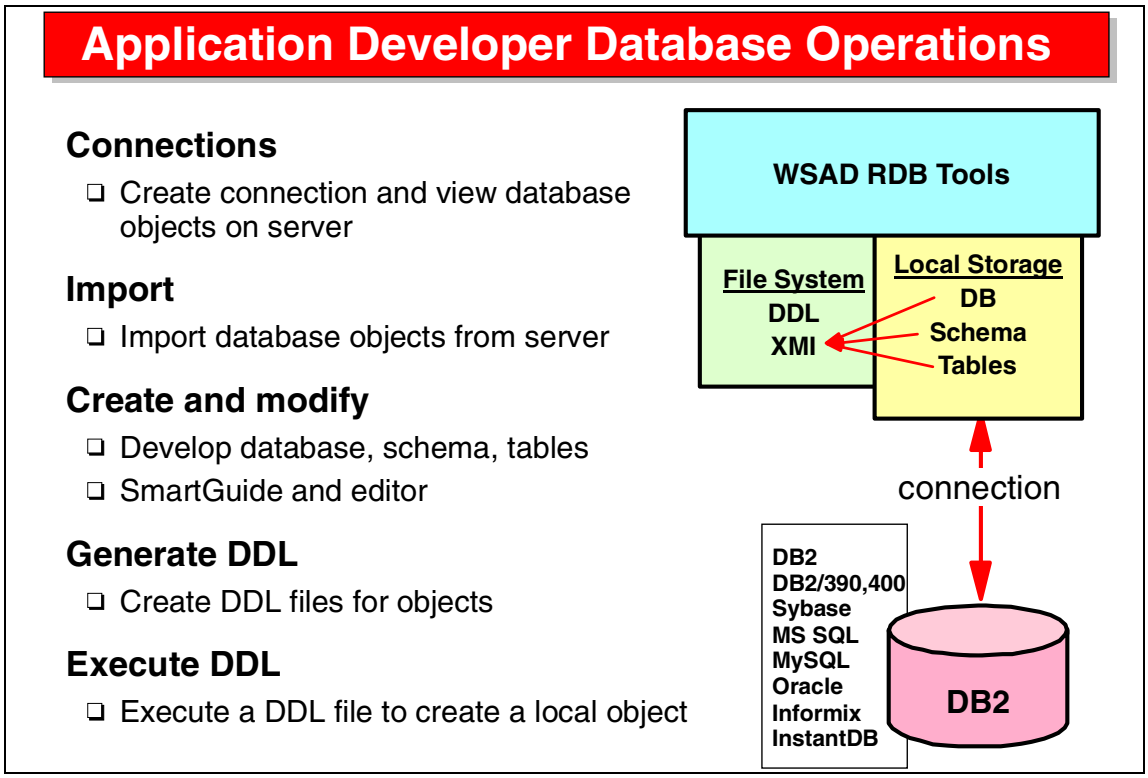
Visual 4-1 Title



Visual 4-2 Objectives

The objectives of this unit are:

- Understand the relational schema center that provides tools for relational database development tasks
- Understand the Data Perspective with the DB Explorer and Data views
- Understand the SQL Query Builder
- Understand the tasks and editors available for managing database descriptors



Visual 4-3 Application Developer Database Operations

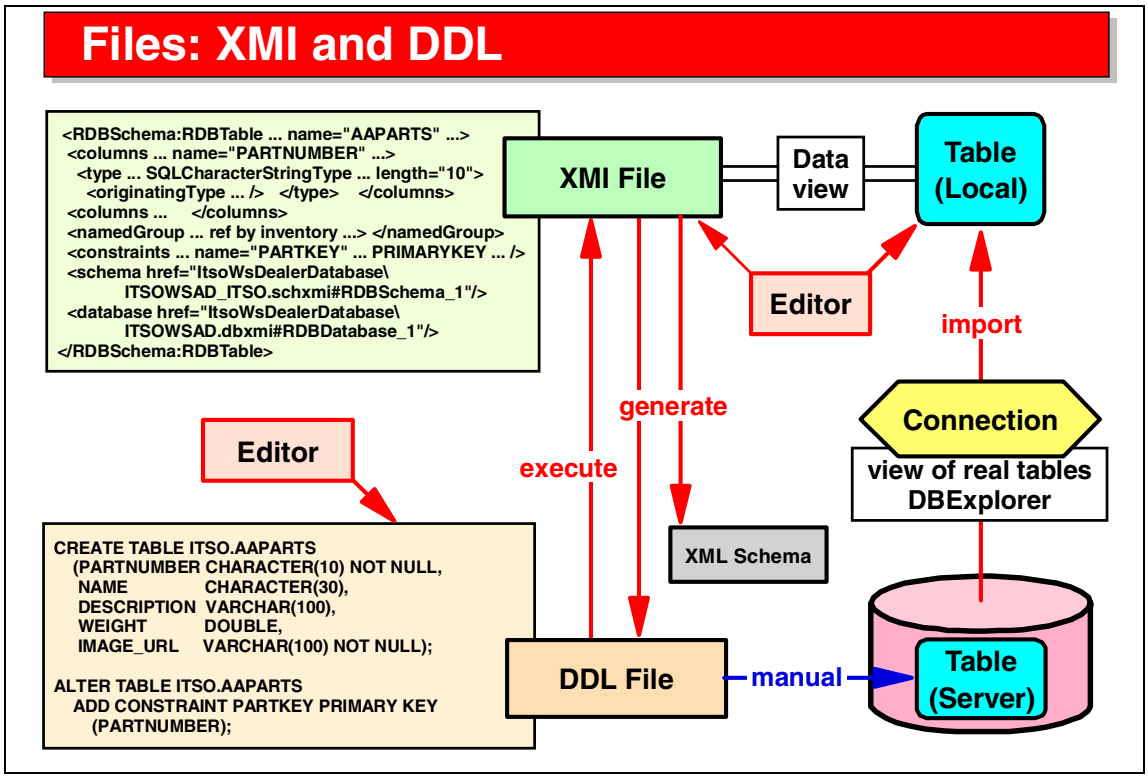
The Application Developer provides:

- Local storage (XMI files) to store database descriptors (database, schema, and tables for now)
- Active connections to databases to retrieve existing definitions and to build and run SQL queries

Connections are used to retrieve existing descriptors into the local storage format (XMI).

Editors and tools are used to create and manipulate local descriptors, generate DDL from local descriptors, and run existing DDL to create local descriptors.

All major database systems are supported.



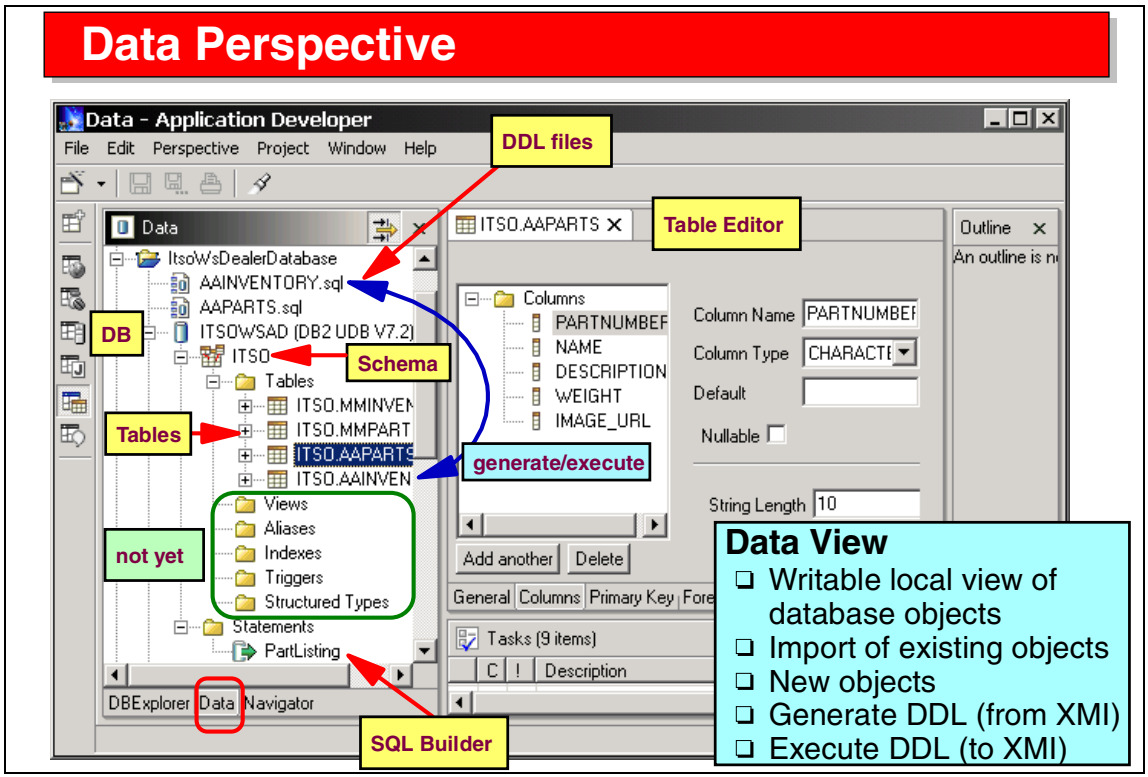
Visual 4-4 Files: XMI and DDL

Local descriptors are kept in XMI files that can be manipulated with tailored editors in the Data view. These editors provide a graphical view of the data; editing is not performed on the XMI source file.

Through a connection in the DB Explorer view, you retrieve existing table definitions and convert the definitions into local XMI files.

XMI files and DDL files are used to store the definitions, and each format can be converted into the other; that is, you can generate DDL from the XMI file, or generate the XMI file by executing the DDL file.

You can also generate an XML schema from a table definition.



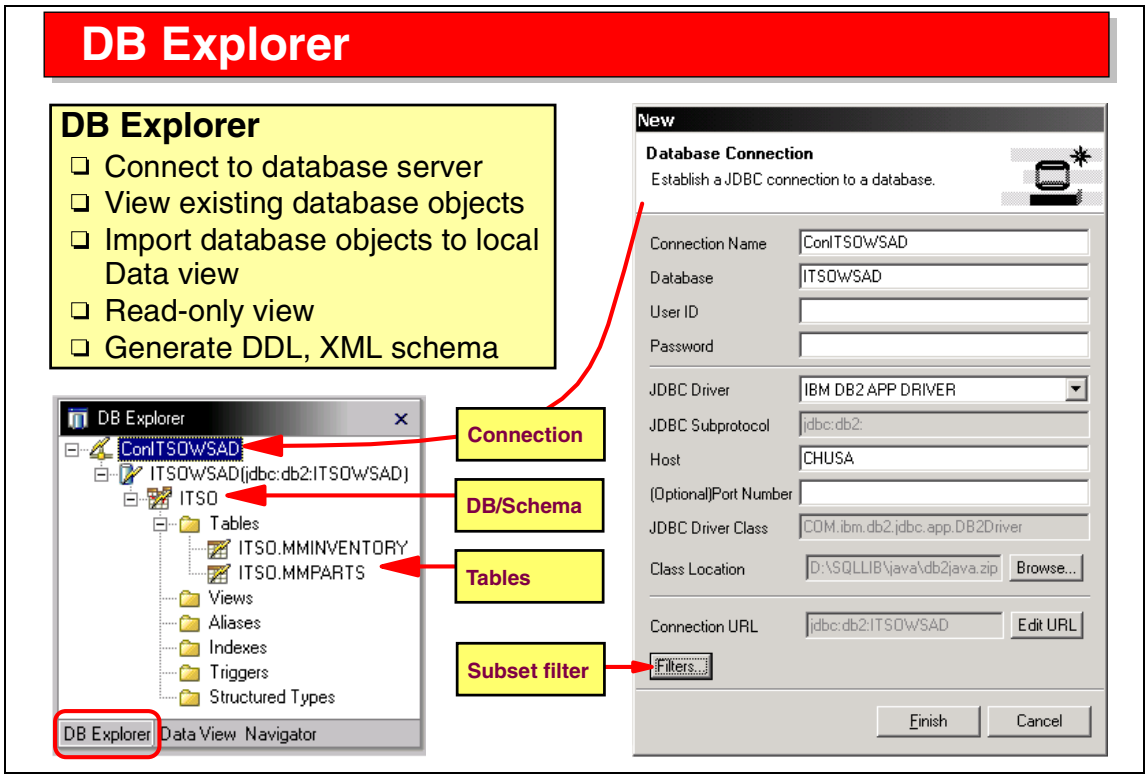
Visual 4-5 Data Perspective

The Data Perspective provides views and tools for definition and maintenance of descriptors for database, schema, and table definitions.

The Data view shows database descriptors (XMI files) in a hierarchical format. Editors are provided to define database, schema, and table descriptors. Tools are provided to generate SQL DDL files for such descriptors, or to create a descriptor from an existing DDL file.

The DB Explorer view enables connections to database definitions in real time and to import existing descriptors as local resource files (in XMI format).

SQL statements can be defined graphically using a wizard. Such statements can be executed for testing, and can be used by XML tools to convert database data into XML files. Statements can also be used by Web development tools to create skeleton Web applications (servlets and JSPs) that access databases.

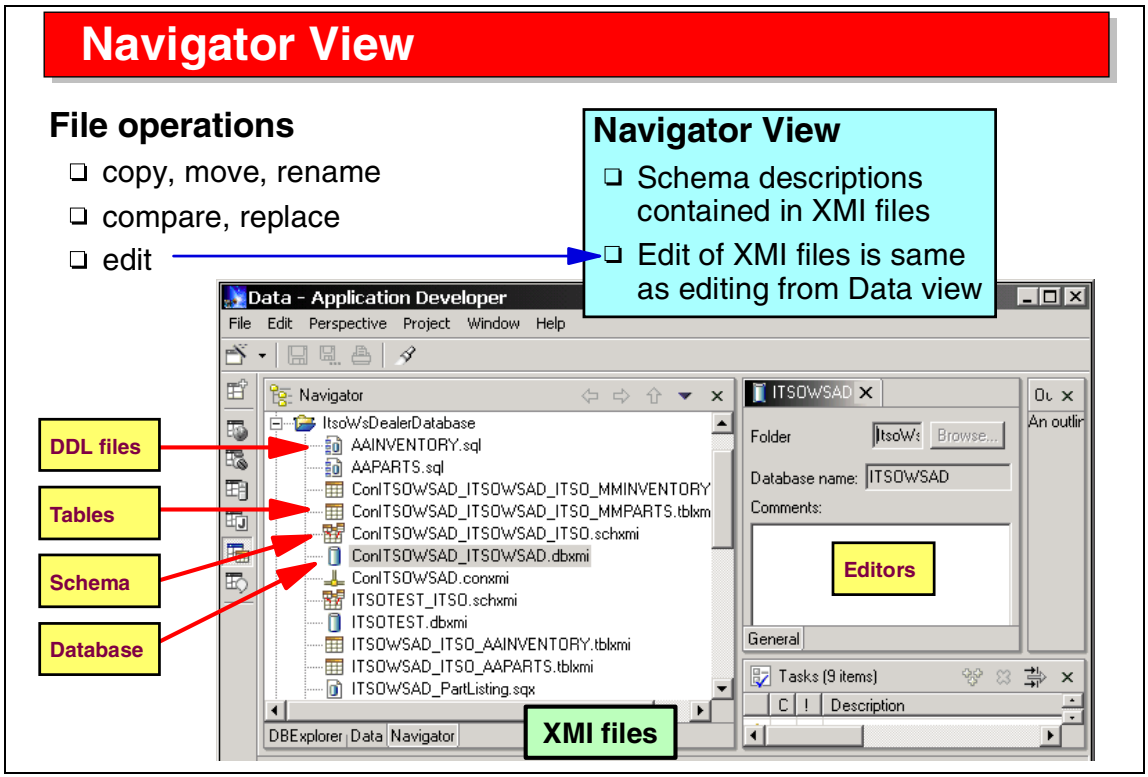


Visual 4-6 DB Explorer

Through the DB Explorer view, you can connect in real time to a database and retrieve existing descriptors.

A SmartGuide is provided to define the connection information, such as the JDBC driver. A filter can be set up to only retrieve tables that match a given naming convention.

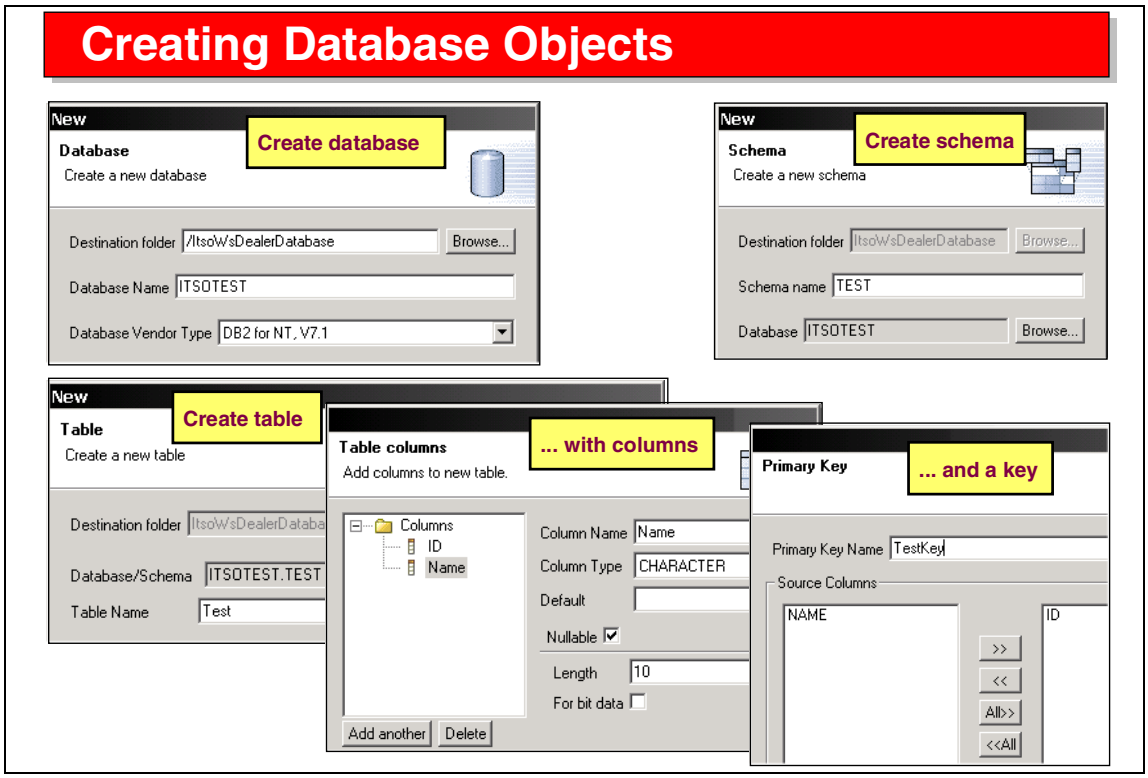
The DB Explorer is a read-only view. After viewing real-time information, the descriptors can be imported into a folder and then manipulated using the Data view. From the DB Explorer, you can generate DDL files and XML schemas.



Visual 4-7 Navigator View

The Data Perspective also provides a Navigator view that can be used for operations on the underlying XMI files.

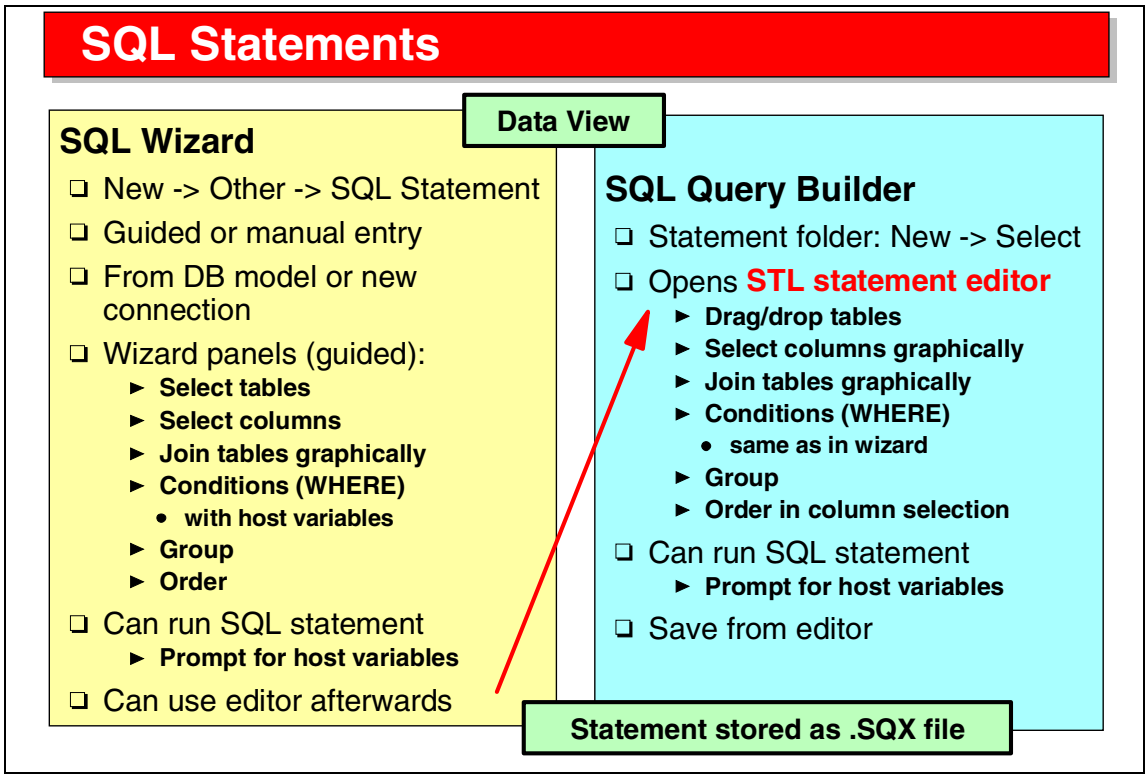
Invoking an editor on such an XMI file does, however, open the same graphical editor as from the Data view.



Visual 4-8 Creating Database Objects

SmartGuides are provided to create database, schema, and table descriptors.

For a table you can define the columns with their data types, as well as the primary key and foreign keys.



Visual 4-9 SQL Statements

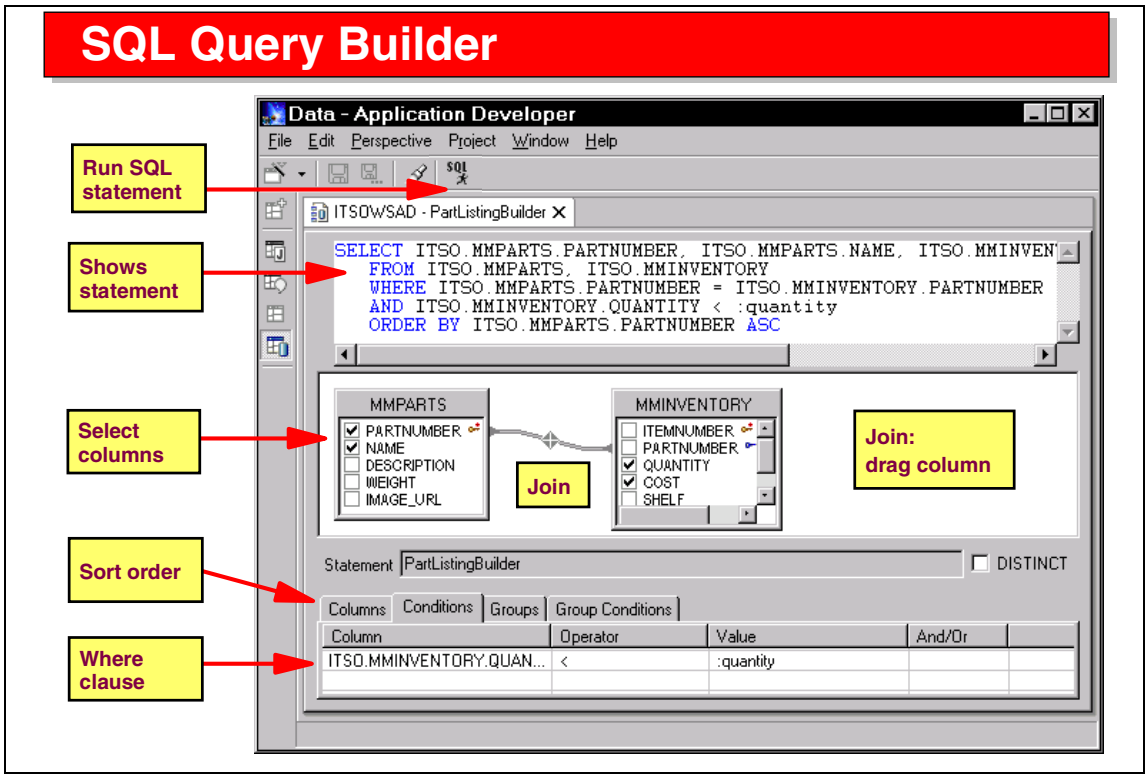
There are two ways to create SQL statements:

- SQL wizard—a guided dialog that goes through a number of panels to select the table(s), select the columns, and to provide join, where clause condition, grouping and order information. SQL statements built with the wizard can be edited afterwards with the query builder.
- SQL query builder—a graphical editor to specify the tables, columns, join, conditions, group, and order. Instead of a guided dialog, a set of panels accessible through tabs is provided.

SQL statements can be built from an imported database model or through an active connection. SQL statements can be run against a real database, and you are prompted for host variables that were defined in where conditions.

SQL statements are stored as .SQX files, which is an XML file.

Note that you can build select, insert, update, and delete statements.



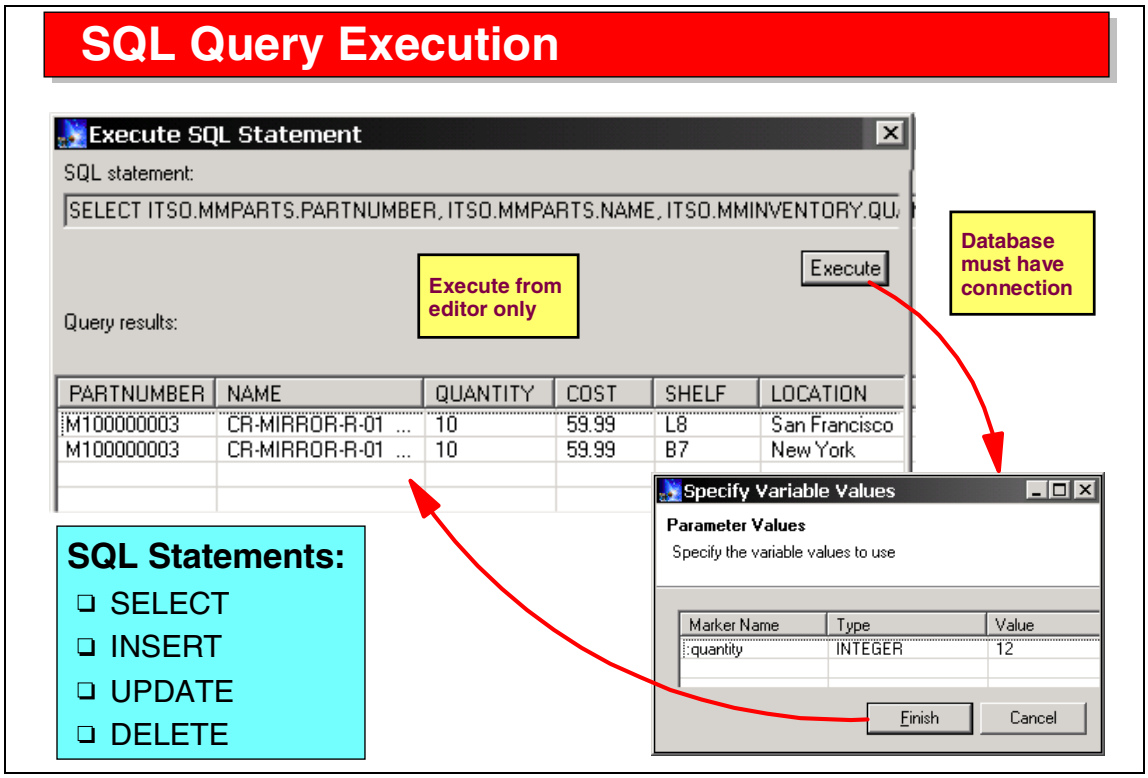
Visual 4-10 SQL Query Builder

The SQL query builder is really an editor with three panes:

- The top pane shows the actual SQL statement and you can edit the content.
- The middle pane shows the tables, selected columns, and joins. You can drag and drop table objects from the Data view into this pane (and also into the top pane).

  A join is performed by dragging a column from one table to the matching column in another table.
- The bottom pane contains a set of panels that are accessible through tabs and are used to specify sort information, where clause conditions, and grouping information.

An icon is provided to run the SQL statement.



Visual 4-11 SQL Query Execution

When an SQL query is executed, you are prompted for the values of any host variables used in where clauses.

Results are displayed in table format for select statements.

Summary

Relational Schema Center and Data Perspective provide

- Management of database objects
  - **View of real server tables**
  - **Versions supported in team environment**
- DDL generation
  - **Implement database objects in target environment**
- Schema design for EJBs
  - **Container-managed entity beans with associations/inheritance**
  - **Schema is required for EJB development**
- Base for SQL statements
  - **Run SQL statements to generate XML files**
  - **Use SQL statements for Web applications**

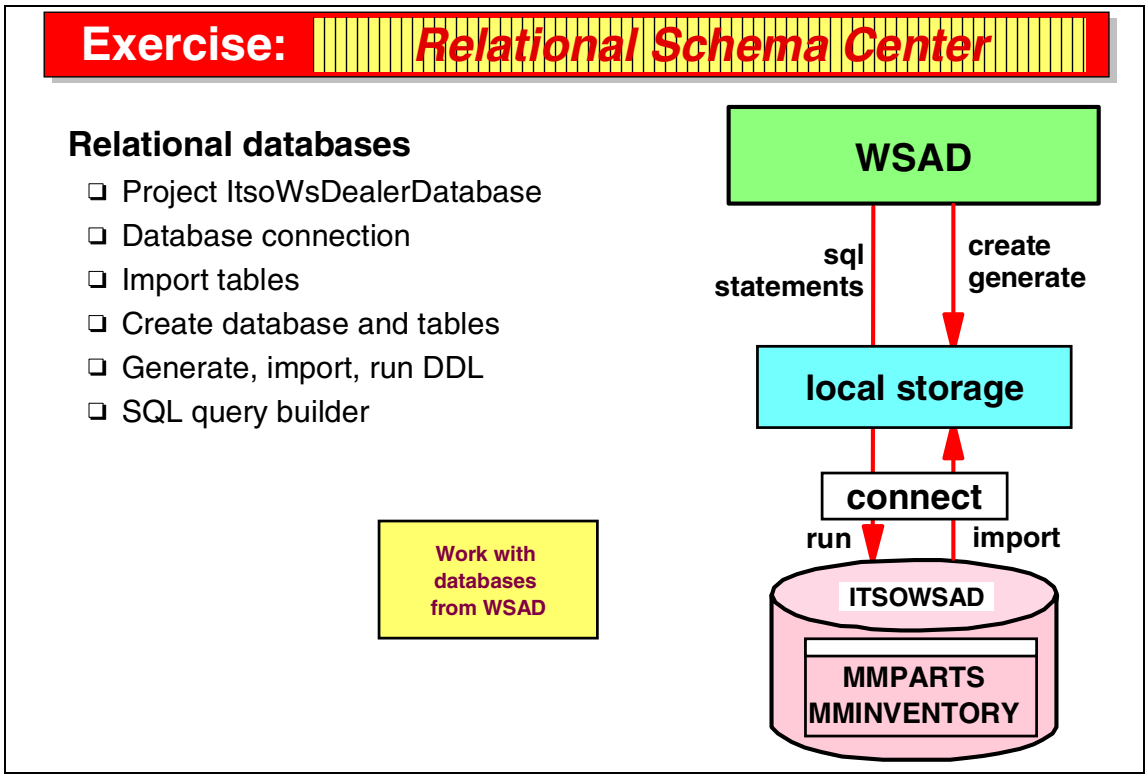
Visual 4-12 Summary

The relational schema center with the Data perspective and its DB Explorer and Data views can be used to perform database descriptor maintenance tasks that are required when developing Java applications that access such databases.

Maintaining database schemas is a requirement when developing entity EJBs that require a mapping between the EJB and an underlying table.

The SQL statement builder is used for SQL statements that are used in Web applications. In a Web project, we can generate servlets and JSPs based on such SQL statements.

SQL statements can also be run to generate result data in XML format.



Visual 4-13 Exercise: Relational Schema Center

The relational database exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with the ITSOWSAD database to:

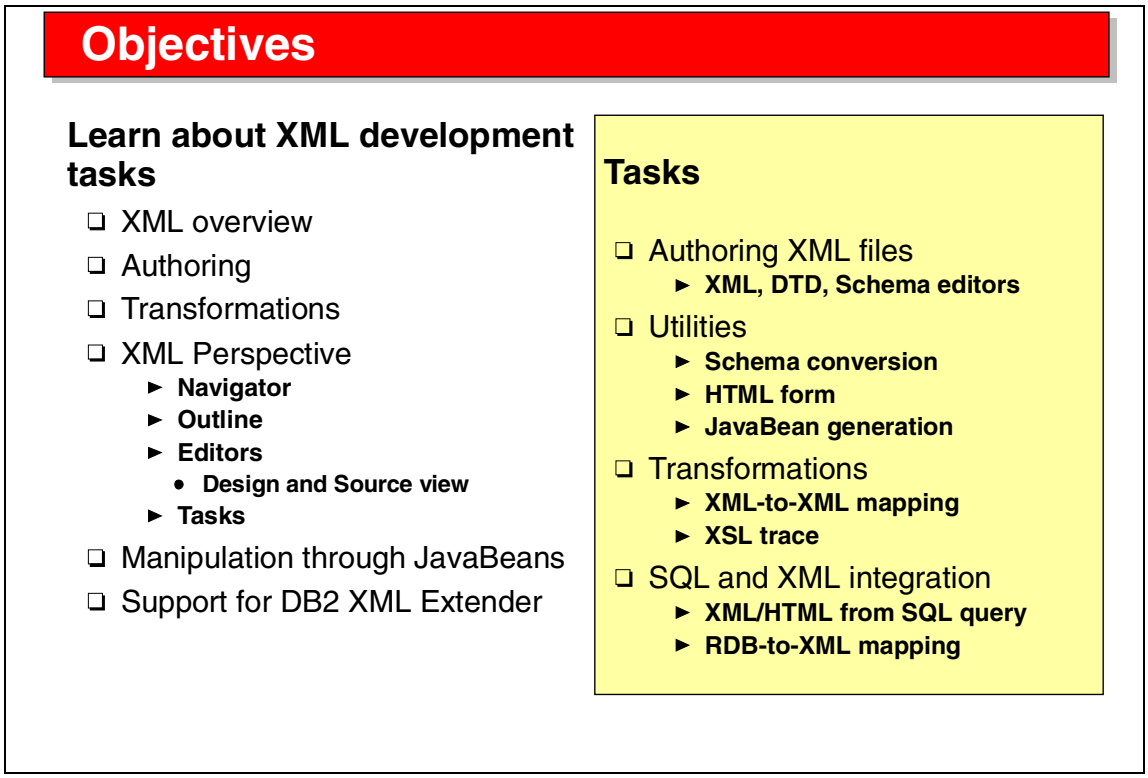
- Retrieve existing descriptors using a connection
- Create and manipulate descriptors using the SmartGuides and editors
- Work with DDL files
- Use the SQL query builder to create an SQL statement

See Exercise 2, “Relational Schema Center” on page 309 for the instructions for this exercise.

5

Application Developer: XML Development

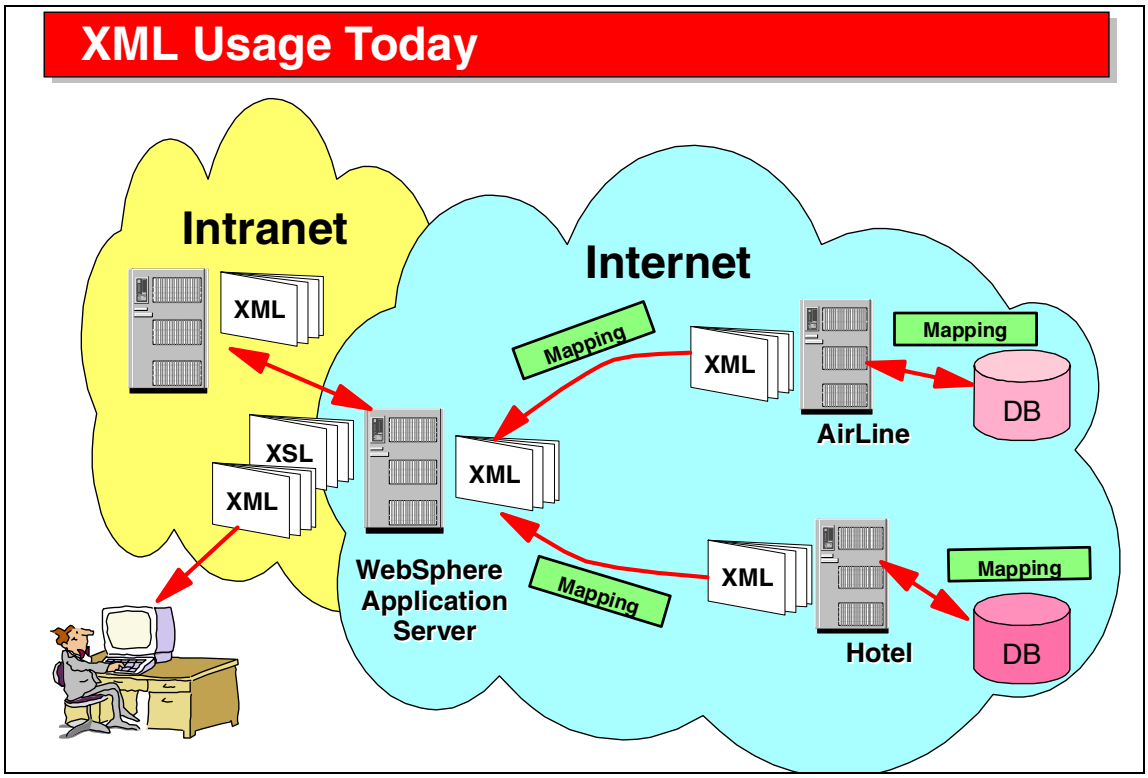
Visual 5-1 Title



Visual 5-2 Objectives

The objectives of this unit are to:

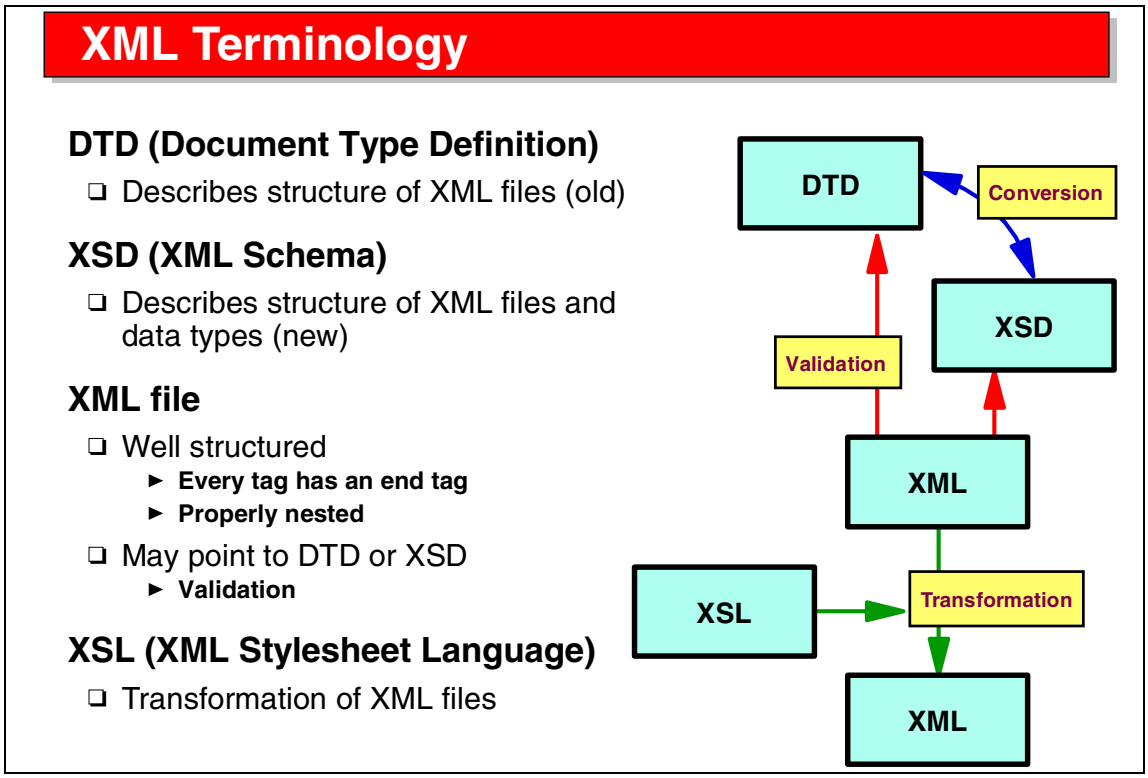
- Understand the tools (editors and utilities) provided to work with XML files, DTD files, and XML schemas
- Understand the XSL transformation tools
- Understand the integration between SQL and XML



Visual 5-3 XML Usage Today

XML is already used in many applications on the Internet and on the intranet to send data from one application or database server to another.

XML and XSL are also used to translate such data into formats supported by Web browsers.

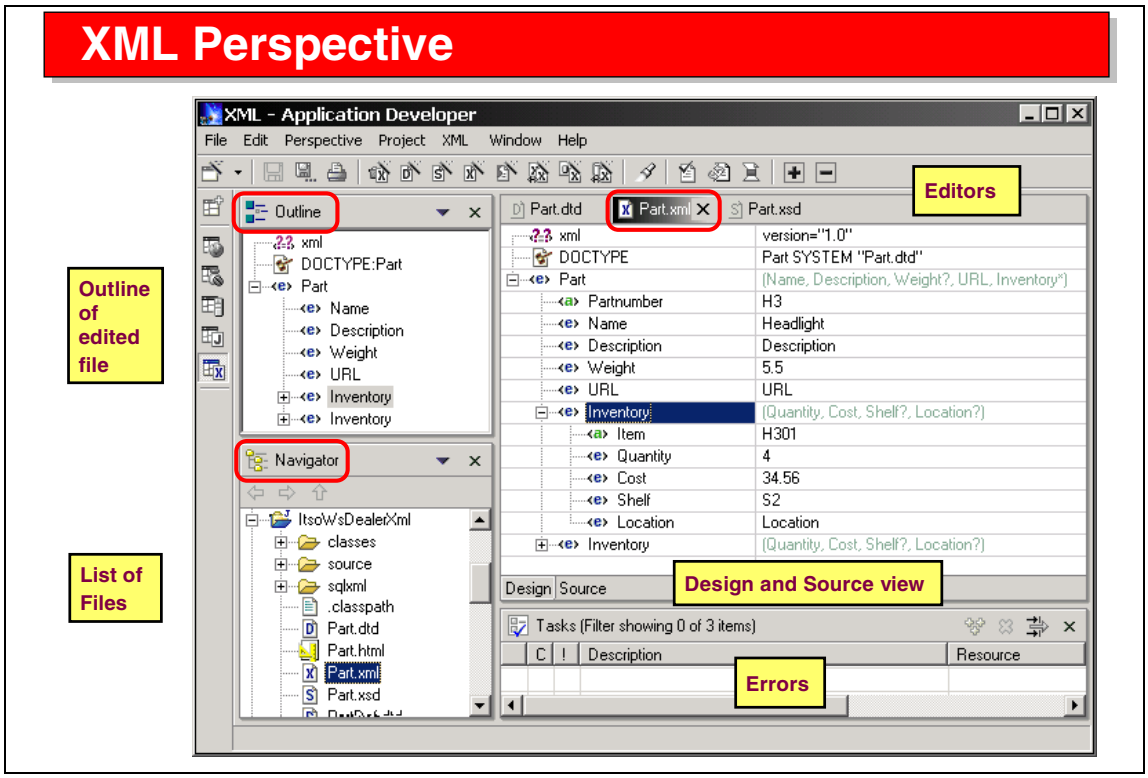


Visual 5-4 XML Terminology

The different types of files used with XML are:

- DTD—describes the structure of an XML file (old style descriptor)
- XSD (XML schema)—describes the structure and the data types of an XML file (new style descriptor, is itself an XML file)
- XML—a well-structured file that follows XML conventions:
  - Every tag must have an end tag
  - Tags must be properly nested

  An XML file may point to a DTD or XSD for validation of the structure and the data types.
- XSL—used by an XSL translator program to convert an XML file into another format (XML, HTML, other)



Visual 5-5 XML Perspective

The XML Perspective is provided for developing XML applications or adding XML functionality to Web applications.

XML editors for XML files, DTDs, and XML schemas are provided for manipulation of XML resource files.

XML tools are provided to convert XML descriptors (DTDs and schemas), generate files, generate XSL style sheets for XML manipulation or for converting database data into XML data.

Authoring Tools

Integrated set of visual editing tools

- DTD editor
- XML editor
- XML Schema (XSD) editor

Conversions

- DTD to XSD
- XSD to DTD

Views

- Outline view with structured content (add, remove)
- Design view for structural editing with choice list
- Source view with intelligent assist

Supports W3C standard

Validation of files

- DTD, XSD
- XML against DTD/XSD

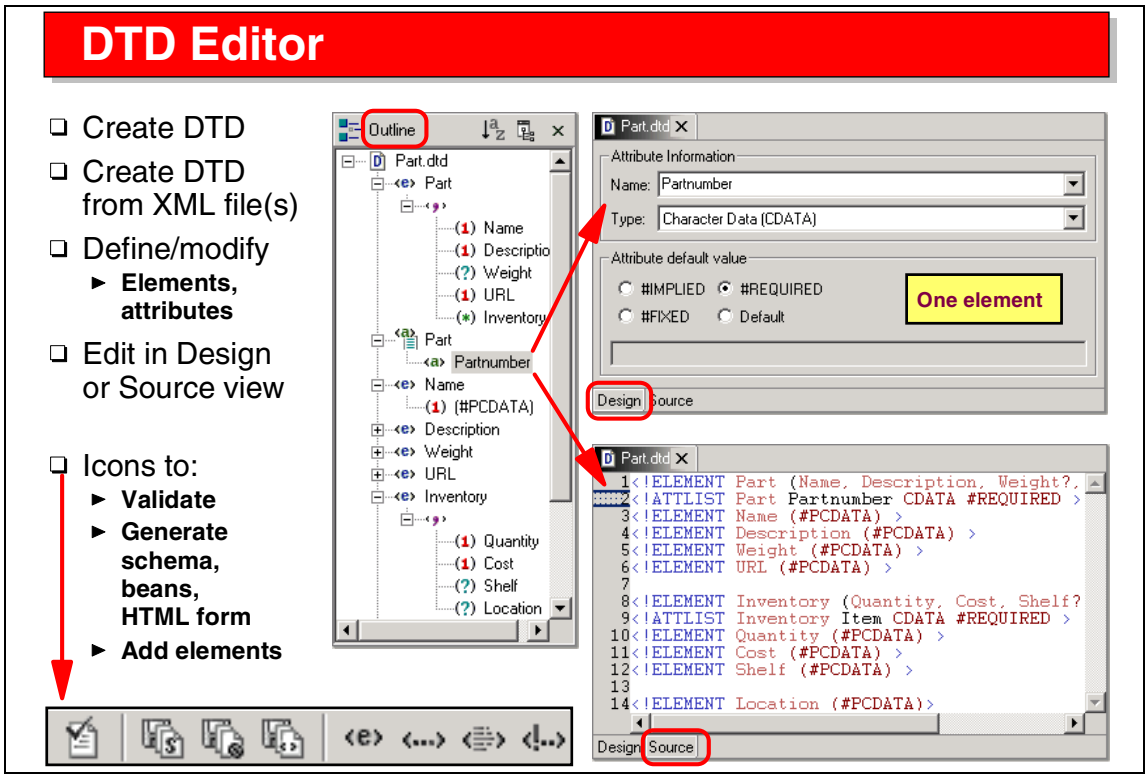
Visual 5-6 Authoring Tools

Graphical editors are provided to create and manipulate XML, DTD, and XML schema files.

These editors provide a Design view for structural editing and a Source view for direct editing of the source code (including code assist). Coupled with the editor is an Outline view with structured content for adding and removing elements.

Conversion utilities can generate an XSD from a DTD, or a DTD from an XSD.

All file formats can be validated; note that XML files can be validated against the DTD or XSD that they refer to.



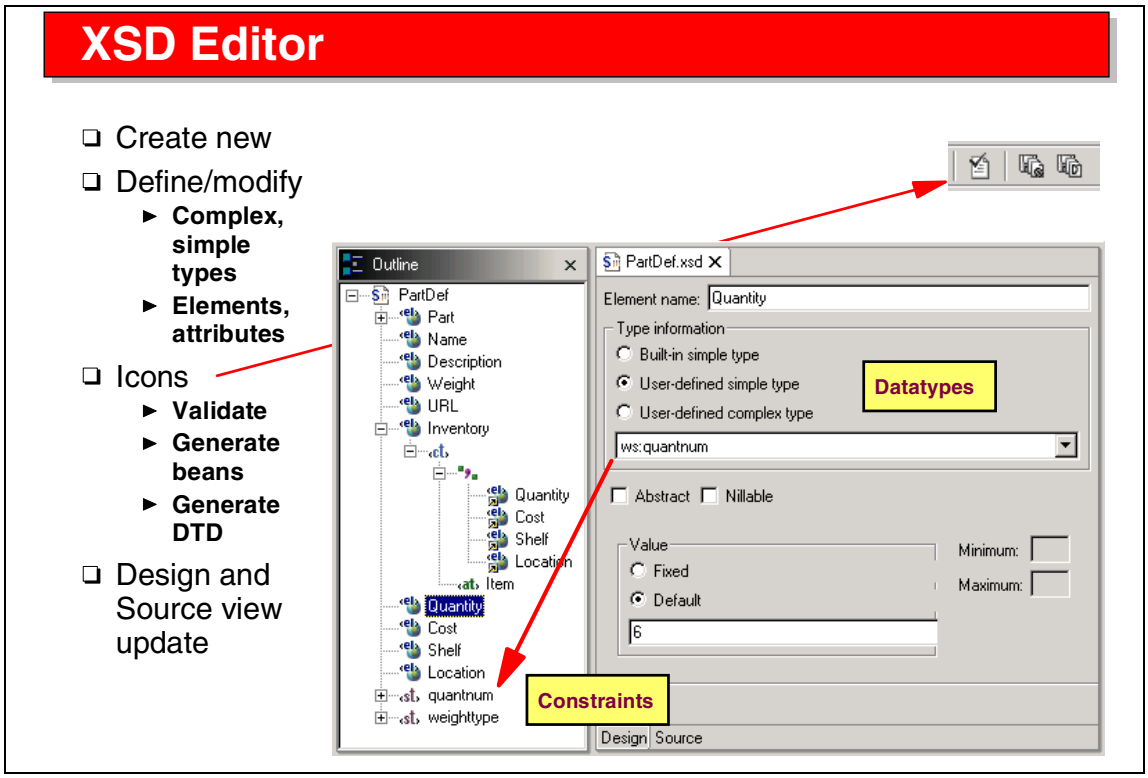
Visual 5-7 DTD Editor

The DTD editor enables you to create a new DTD or edit existing DTDs.

A Design view and a Source view is provided, and an Outline view is coupled to the editor.

Elements are added, removed, and selected in the outline, and edited in the Design or Source view.

Icons are provided to invoke the utilities to validate a file, generate an XML schema, generate JavaBeans, or generate an HTML form.



Visual 5-8 XSD Editor

The XSD editor enables you to create and edit an XML schema.

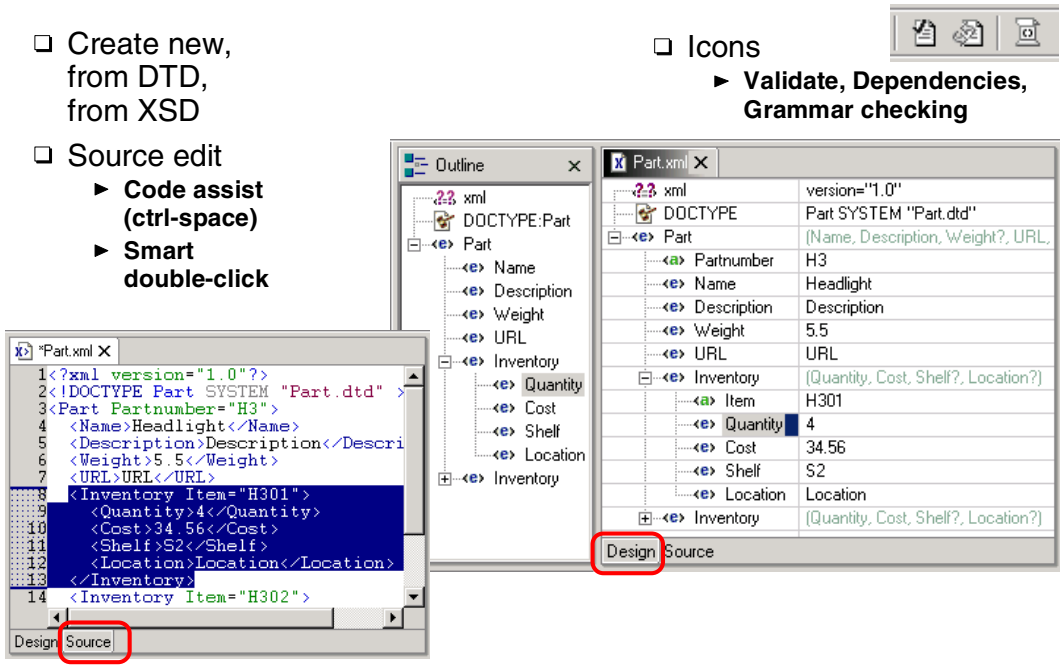
A Design view and a Source view is provided, and an Outline view is coupled to the editor.

Elements are added, removed, and selected in the outline, and edited in the Design or Source view.

Icons are provided to validate a file, generate JavaBeans, or generate a DTD.

XML Editor

- Create new, from DTD, from XSD
- Source edit
  - **Code assist (ctrl-space)**
  - **Smart double-click**
- Icons
  - **Validate, Dependencies, Grammar checking**



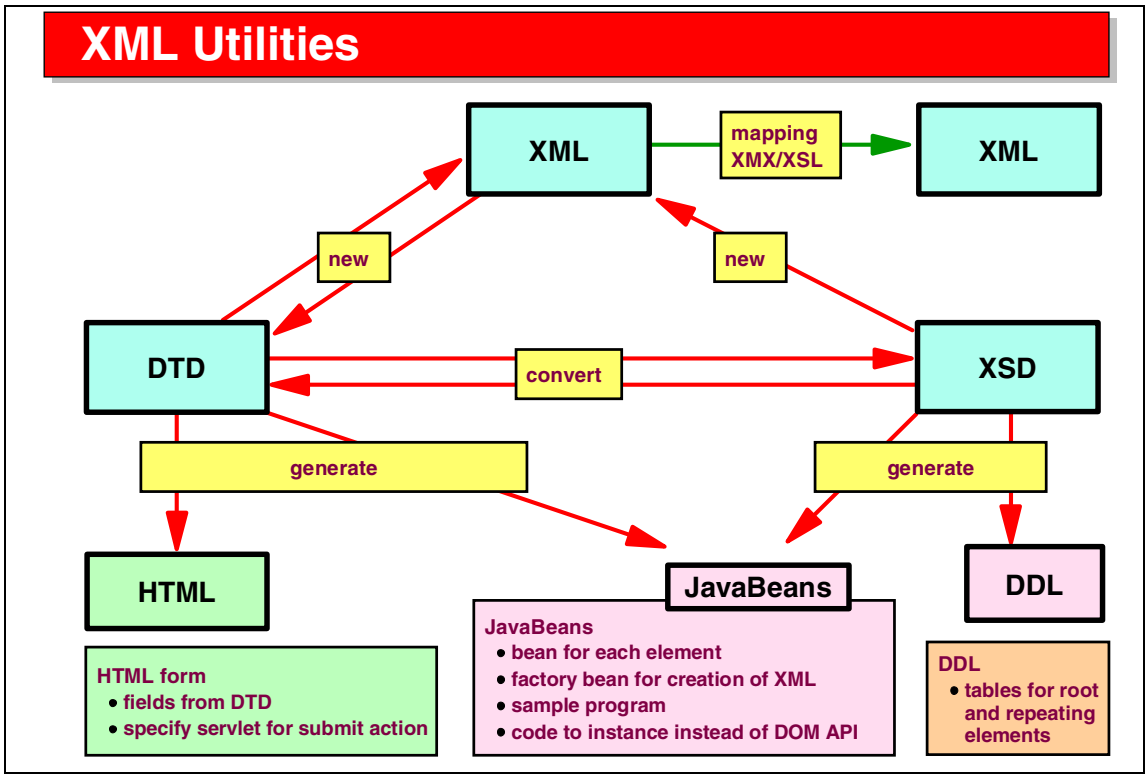
Visual 5-9 XML Editor

The XML editor enables you to create or edit XML files.

Skeleton XML files can be generated from a DTD or schema.

Code assist (ctrl-space) is available in the Source view.

Icons are provided for validation and grammar checking.



Visual 5-10 XML Utilities

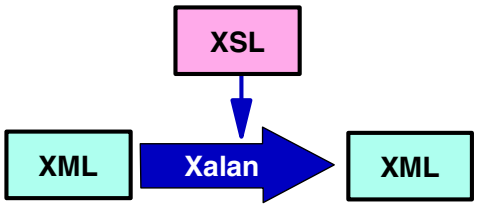
This diagram shows the conversion and generation possibilities:

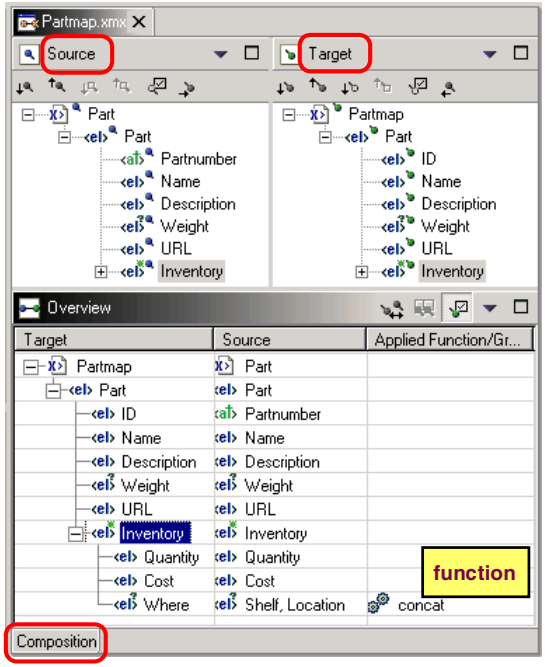
- Conversion between DTD and XSD files.
- Create new skeleton XML files from DTD or XSD.
- Generate a DTD from one or multiple XML files.
- Generate a skeleton HTML form from a DTD (the form includes a field for every element in the DTD, and you can specify the name of a servlet to be invoked by the Submit action).
- Generate JavaBeans from DTD or XSD (one bean for each element, a factory bean, and a sample program). This API can be used instead of the more complicated DOM API provided by XML parsers.
- Generate DDL for tables from an XSD.
- Generate a mapping with an associated XSL file to convert an XML file to another format.

XML-to-XML Mapping

Convert XML file from one definition to another

- Define mapping between DTDs, XSDs, or XML files
  - **By name, manual (XMX file)**
- Generates transformation XSL
  - **XSLT processor Xalan**
- Can define functions for conversions
  - **One element into multiple**
  - **Multiple elements into one**





Visual 5-11 XML-to-XML Mapping

The XML-to-XML mapping function enables to you specify a mapping between two DTDs (or two XSDs or two XML files).

The mapping is performed by drag and drop between the elements of the two files. Simple mapping from one element to another, and mapping by function (concatenation, substring...) between one and multiple elements is supported.

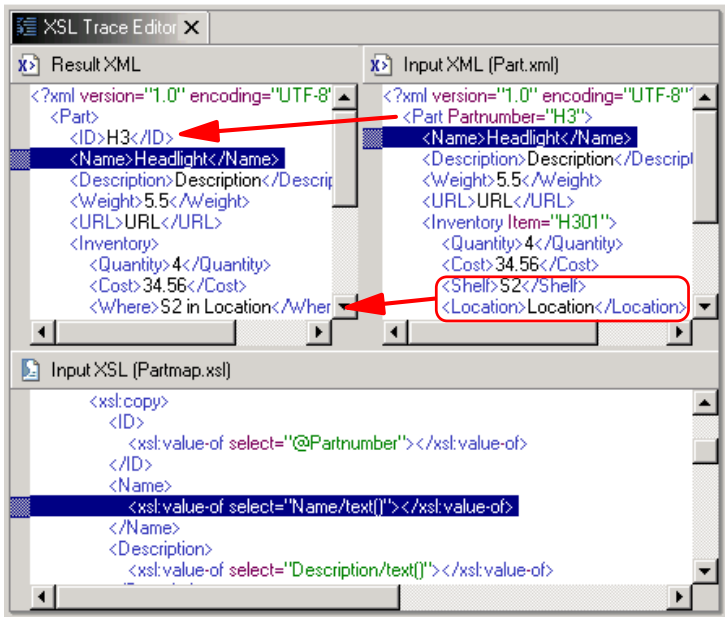
From the mapping, an XSL file is generated. This file can be used by the XALAN XSLT processor.

XSL Trace

Apply XSL to XML

- Select both XML and XSL file
- Apply to generate XML or HTML
- Step through code using icons

  - **forward, back, restart**
- Save trace as result file



Visual 5-12 XSL Trace

After a mapping has been created and the XSL file has been generated, the XSLT processor can be invoked for an XML file.

The XSL trace facility shows the original XML file, the translated XML file, and the XSL file that was used:

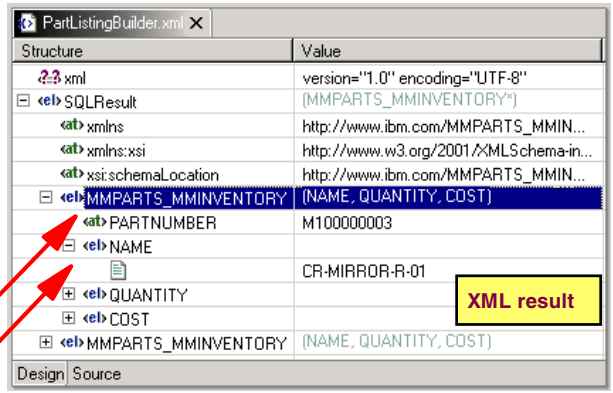
- Icons can be used to step through the execution to analyze the XSL rules that were used to convert the XML elements.

The trace is shown after the conversion has been done; it is not executed in real time.

XML from SQL Query

Convert SQL query result into XML and HTML

- Start with SQL statement defined with RDB tool
- Generate files for conversion
  - **XST - DB connection and SQL statement**
  - **XSD - (or DTD) schema for result XML**
  - **XSL - stylesheet for conversion to HTML**
- Simple mapping
  - **key -> element or attribute**
  - **column -> element or attribute**
  - **foreign key -> link**



HTML result

PARTNUMBER	NAME	QUANTITY	COST
M100000003	CR-MIRROR-R-01	10	59.99
M100000003	CR-MIRROR-R-01	12	59.99

Visual 5-13 XML from SQL Query

An SQL query created with the SQL query builder of the relational toolset can be executed and the result converted into XML and HTML files.

The mapping of key and other columns can be specified so that either attributes of a super element or individual sub-elements are created in the XML file.

A XSL file is generated to create the HTML output in table format.

RDB-to-XML Mapping

Map database persistent data to XML

- Define mapping between table columns and XML element/attribute
- Generates document access definition (DAD) file
  - **Used by DB2 XML Extender**
  - **Compose XML documents from DB2**
  - **Decompose XML document into DB2**
  - **Hides the complexity of creating DADs**
- Test harness for deploying the generated DAD to DB2 XML Extender

Visual 5-14 RDB-to-XML Mapping

For DB2 with the DB2 XML Extender product, XML data may be stored in DB2 tables, either as a BLOB column, or as individual columns for the XML elements.

The mapping is specified in a document access definition (DAD) file.

This feature will not be discussed in detail in this class.

JavaBean Generation

Can generate JavaBeans for each element of DTD or XSD

- Create a package (under source) for the JavaBeans
- Select a DTD (or XSD) and the target package
- Generate the beans

JAR files are added to the build path

- XSDBEANS variable
- XERCES variable

Also generates a sample program that uses the JavaBeans to create an XML file

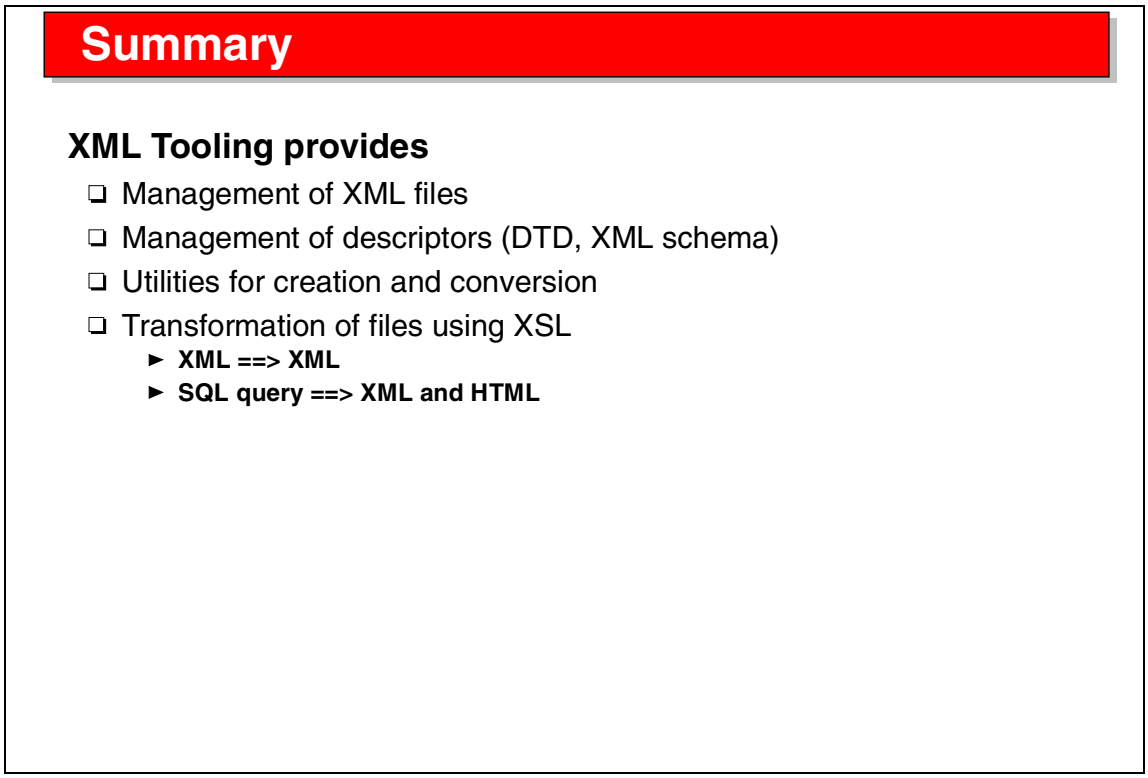
- Sample data is very simple
- Best to update with real data before running
- Result in project folder by default (location is set in sample code)

Visual 5-15 JavaBean Generation

From a DTD or XSD file, JavaBeans can be generated to provide a simple API to programmatically access XML files:

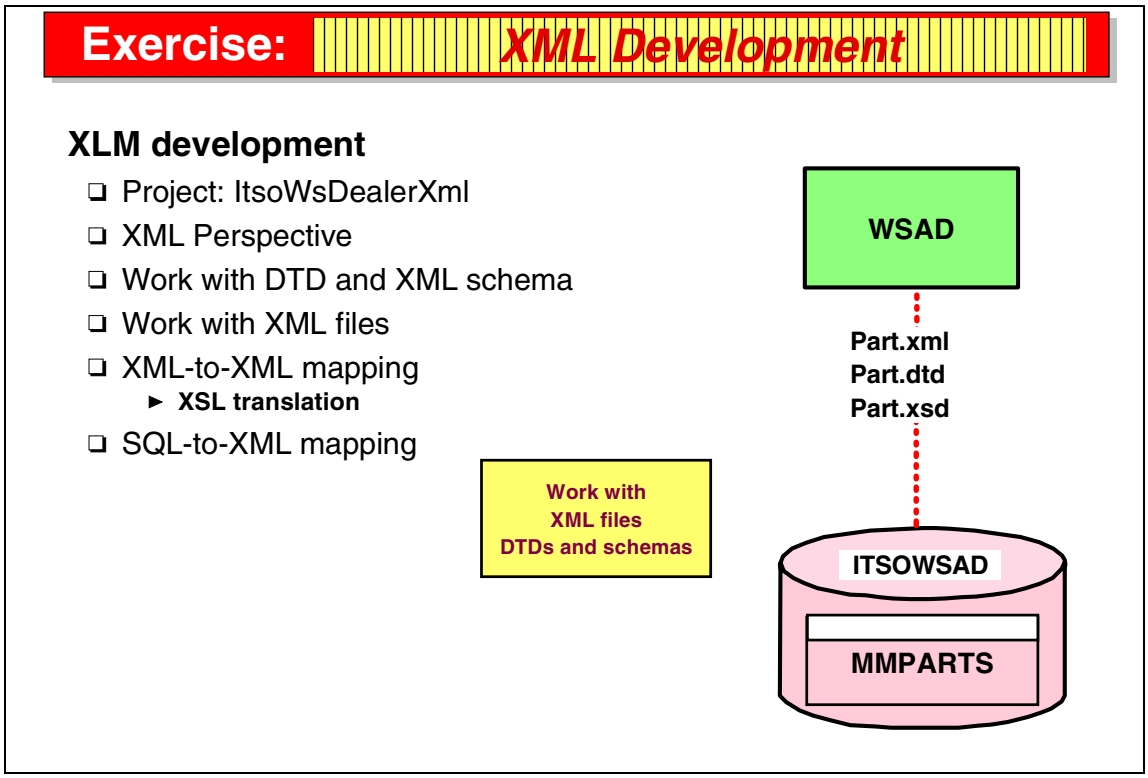
- For each element, a JavaBean is generated. The JavaBean contains sample test data, which should be modified before running the sample program.
- A sample program is generated as well. This program creates a sample XML file by using the generated JavaBeans.

These JavaBeans could be used in own programs to manipulate XML files. This would be an alternative to the DOM and SAX APIs provided by IBM Java XML tools.



Visual 5-16 Summary

The XML tooling of the Application Developer provides a rich set of tools for manipulating XML files and the associated DTD and XSD descriptors.



Visual 5-17 Exercise: XML Development

The XML development exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with XML files, DTDs, and XSDs that describe the parts data stored in the ITSOWSAD database:

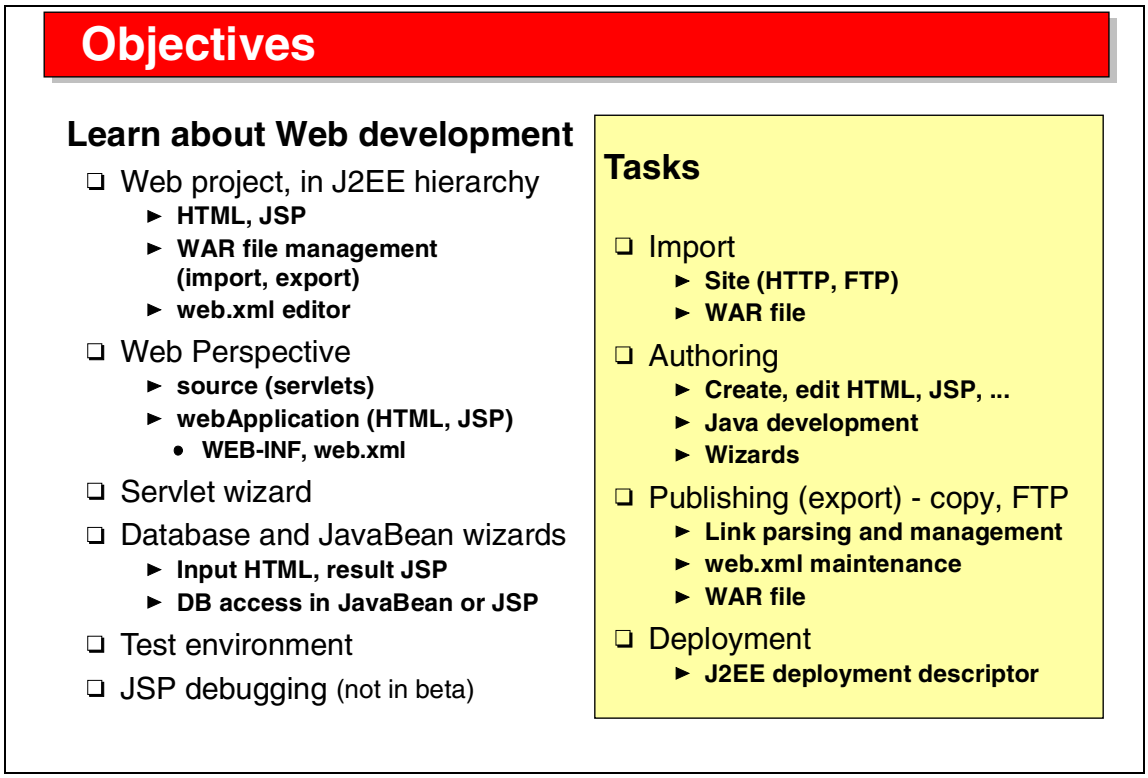
- Use DTD and XSD editors and convert between the formats.
- Work with XML files.
- Create an XML-to-XML mapping between two DTDs that describe the same XML part files. Use the XSLT processor to convert an XML file.
- Use the SQL-to-XML mapping to execute an SQL statement and convert the result into XML and HTML files.

See Exercise 3, “XML development” on page 315 for the instructions for this exercise.

6

Application Developer: Web Development

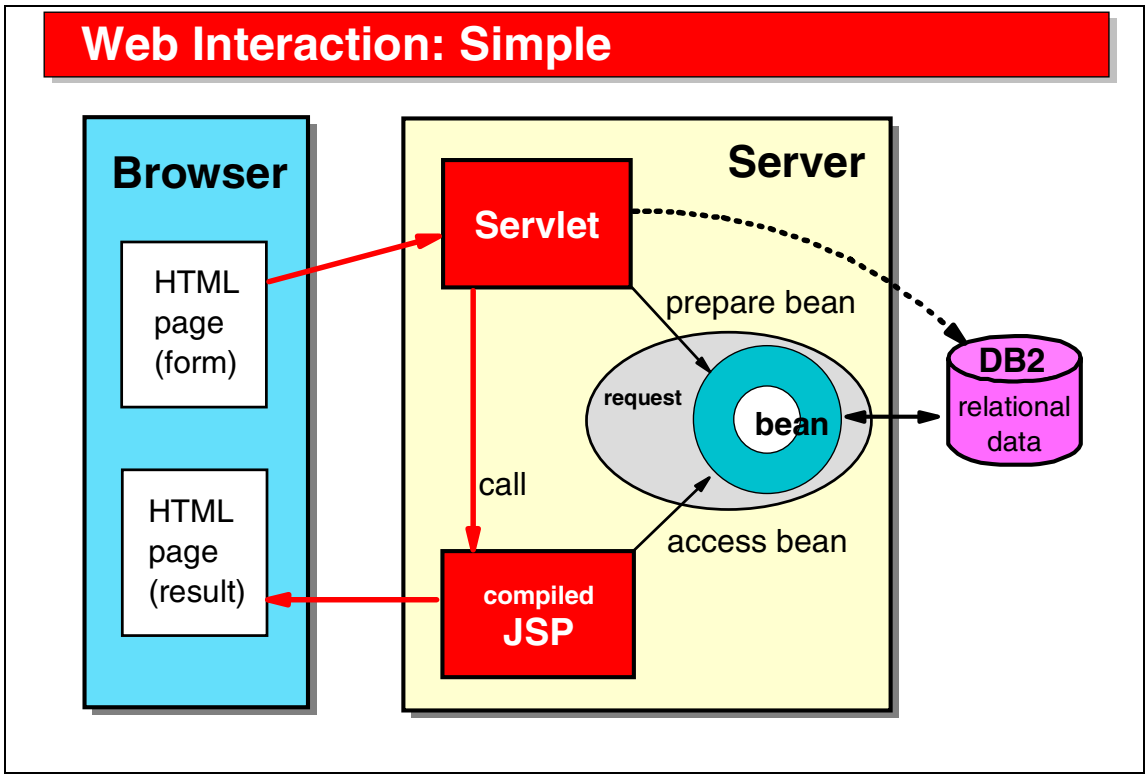
Figure 6-1 Title



Visual 6-2 Objectives

The objectives of this unit are to:

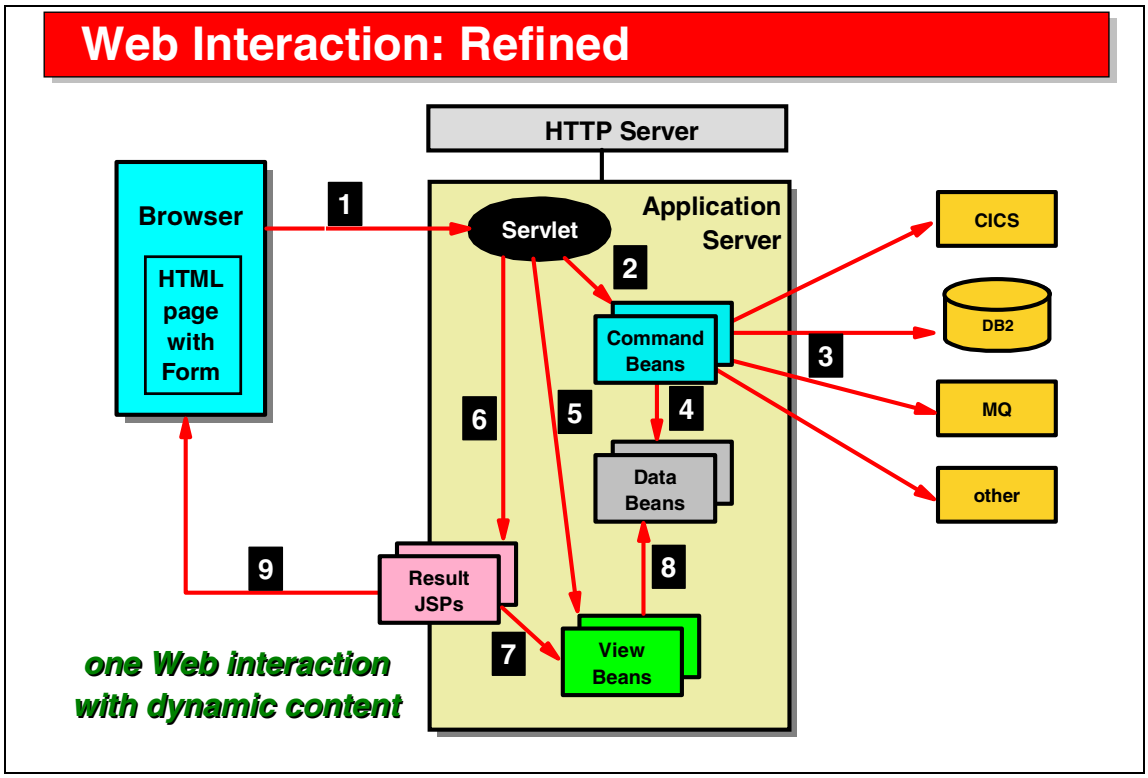
- Understand Web projects and their fit in the J2EE hierarchy
- Understand the Application Developer tools provided for Web development
- Understand the SmartGuides and wizards that are provided to create servlets and simple Web applications based on SQL statements or JavaBeans
- Understand the built-in test environment for running and debugging Web applications
- Understand deployment of Web applications to WebSphere Application Server



Visual 6-3 Web Interaction: Simple

Let us review how a simple Web interaction works:

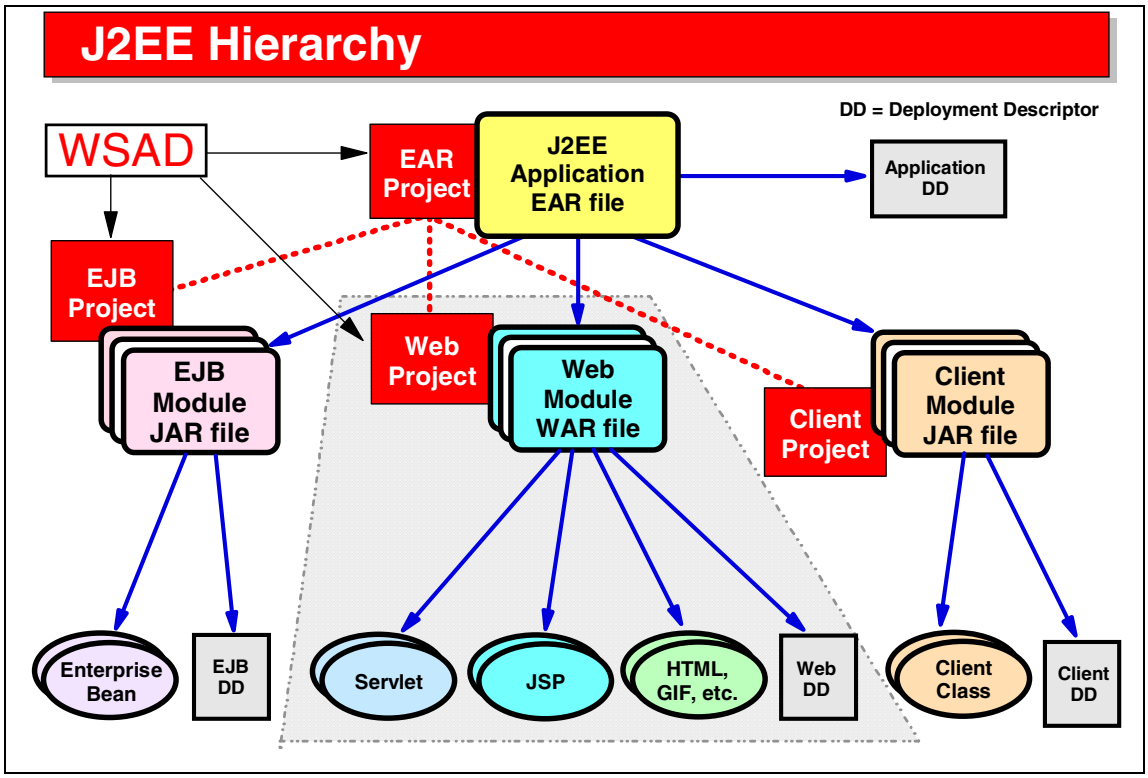
- An HTML page is displayed in a browser. The HTML page contains a form where the user can enter data and submit the form for processing.
- The Web server passes the request to an application server that schedules a servlet to process the form.
- In the model-view-controller (MVC) design pattern, the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



Visual 6-4 Web Interaction: Refined

In many real Web applications, processing is more complex:

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

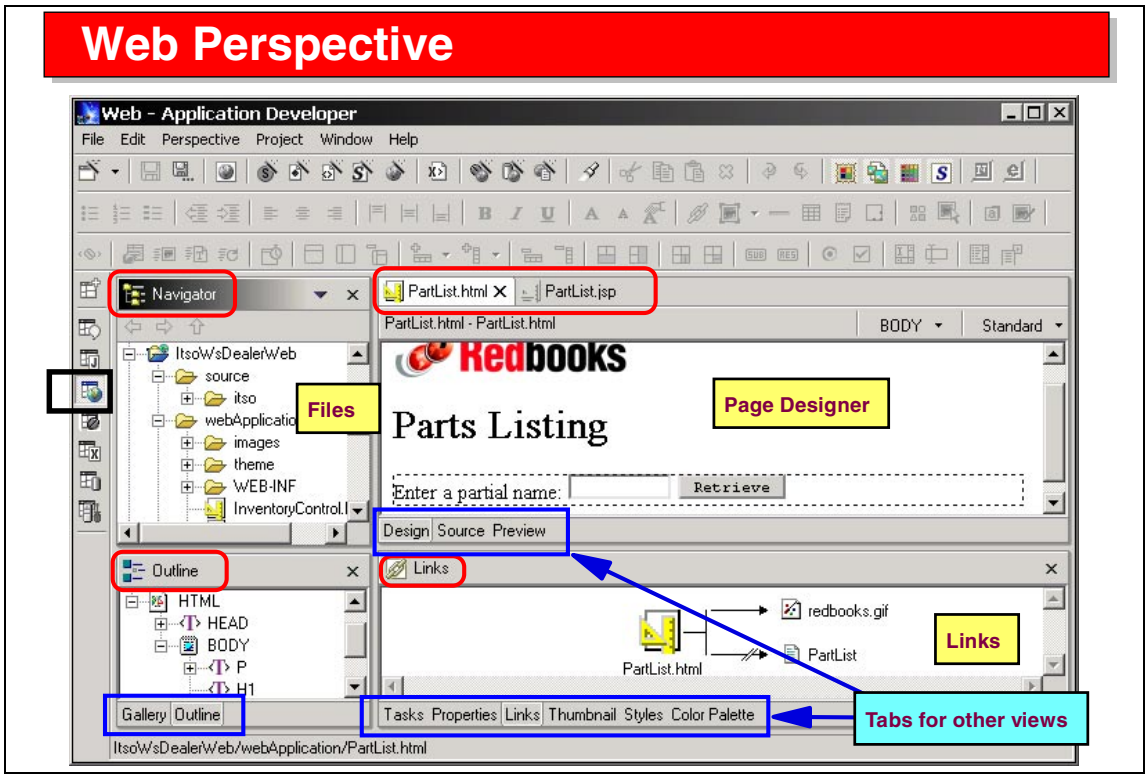


Visual 6-5 J2EE Hierarchy

This diagram shows the J2EE hierarchy and the matching support in the Application Developer:

- A J2EE application is stored in an enterprise archive (EAR) file which contains EJB modules (stored in an EJB JAR file), Web modules (stored in Web archives (WAR) files), and client modules (stored in a JAR file).
- Each of the modules contains a deployment descriptor; for example, a WAR file contains a web.xml file.
- A WAR file contains all the components of a Web application, that is, servlets, JSPs, HTML files, images, and so forth.
- The J2EE hierarchy is matched by projects in the Application Developer. An EAR project contains references to EJB, Web, and client projects. A Web project contains all the resources (servlet, JSP, HTML, images) and the deployment file (web.xml).

This setup makes deployment to a J2EE-based application server very easy.

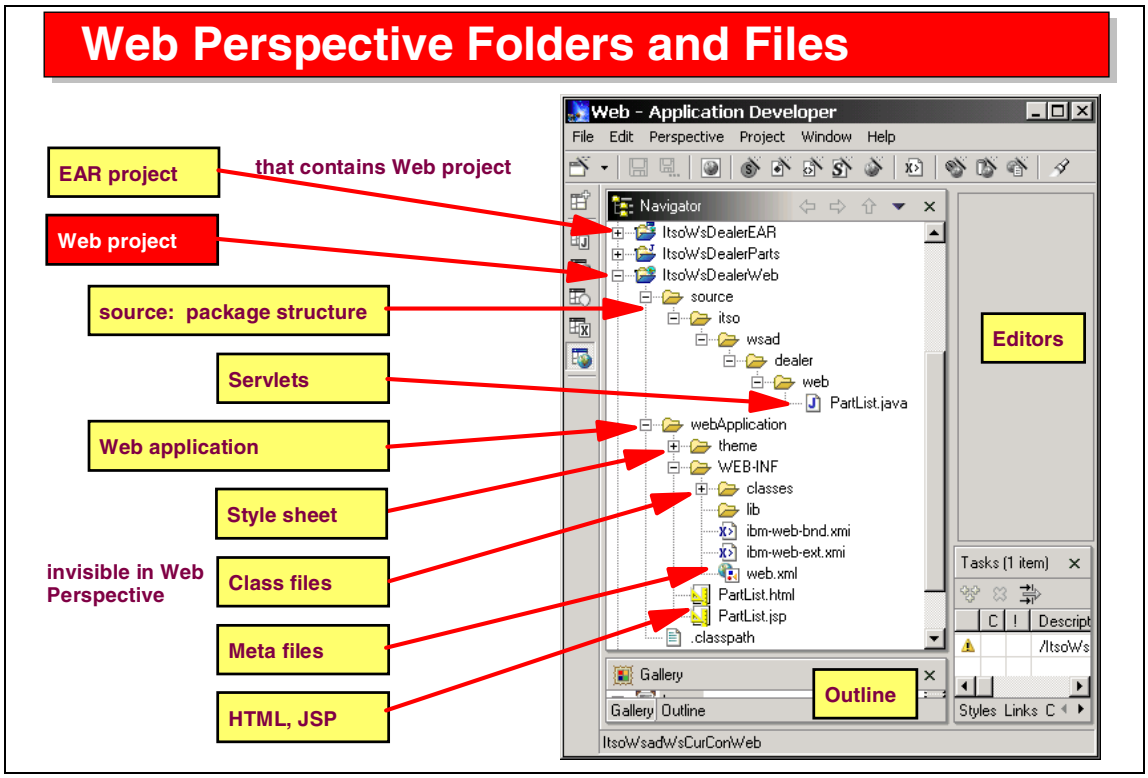


Visual 6-6 Web Perspective

The Web Perspective contains four panes:

- Top left—Navigator view (displays the folders and files of the projects)
- Top right—reserved for editors
- Bottom left—Outline view (of current editor) or Gallery (for HTML/JSP files)
- Bottom right—Tasks (errors), Properties (of selected resource), Links (of Web resources), Thumbnail, Styles, Color, Palette (Web resources)

One of the supported editors is the Page Designer for HTML and JSP files. The Page Designer itself has three tabs to display the Design (WYSIWYG), Source (HTML source code) or Preview (browser) view.



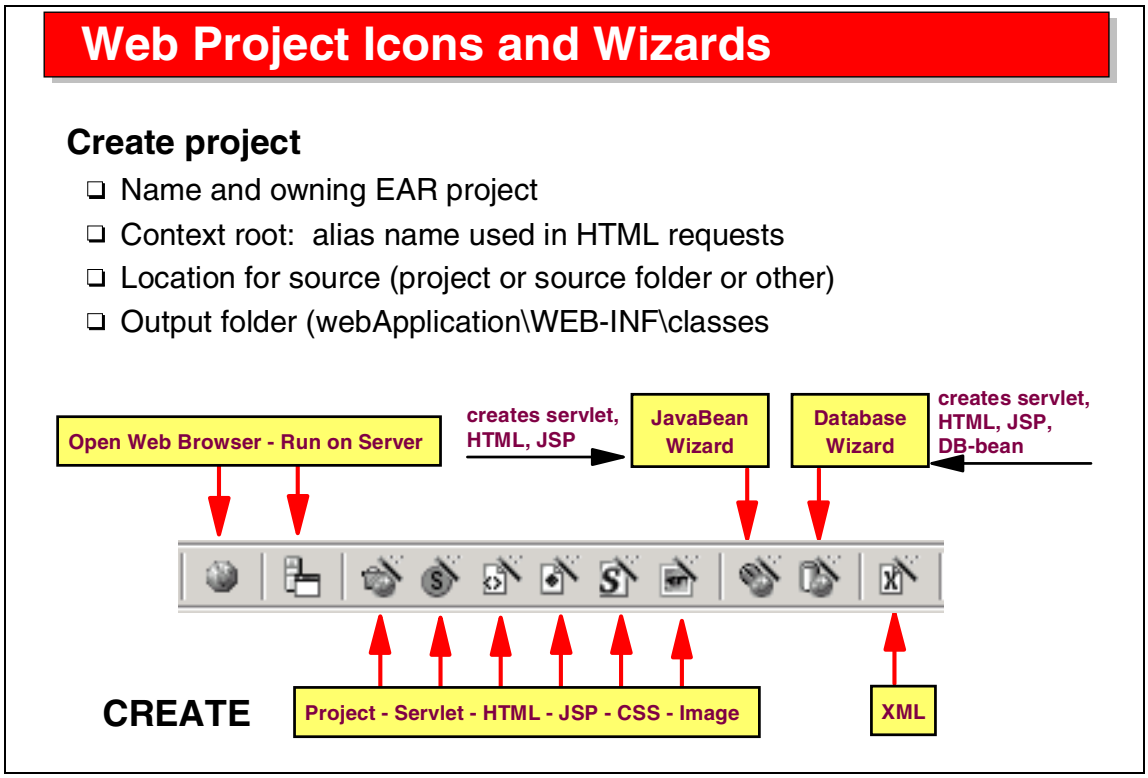
Visual 6-7 Web Perspective Folders and Files

The Navigator view of the Web Perspective shows the setup of a Web project:

- The source folder contains the Java source code of servlets, organized in subfolders that form the package structure.
- The webApplication folder contains all the other Web resources:
  - The theme folder is HTML style sheets.
  - The WEB-INF folder with the compiled servlet classes, the lib folder with additional JAR files, the deployment descriptor (web.xml) and IBM extension files.
  - The HTML, JSP, and image files (these files are usually organized into many subfolders).

Notice our naming standard:

- EAR project names carry the suffix EAR
- Web project names carry the suffix Web
- EJB project names carry the suffix EJB



Visual 6-8 Web Project Icons and Wizards

When creating a Web project, you must also supply an existing or define a new owning EAR project:

- In this class we always name Web projects with a Web suffix and EAR projects with an EAR suffix.

You specify the location of the source (Java servlets), the compiled class files, and the build path (this is the same as for a Java project).

The Web Perspective provides a number of icons to quickly access some of the tasks:

- Open a Web browser (for testing)
- Run on Server (open the welcome page of the Web application)
- Create icons for a Web project and the different type of files
- Icons to invoke the JavaBean wizard and the Database wizard

Editing of Web Resources

Page Designer

- HTML and JSP files, same as in Studio Classic

Animated GIF Designer, WebArt Designer

- GIF files

Stylesheet Editor

- CSS files

Web Application Deployment Data

- web.xml
  - **Define servlets and JSPs**

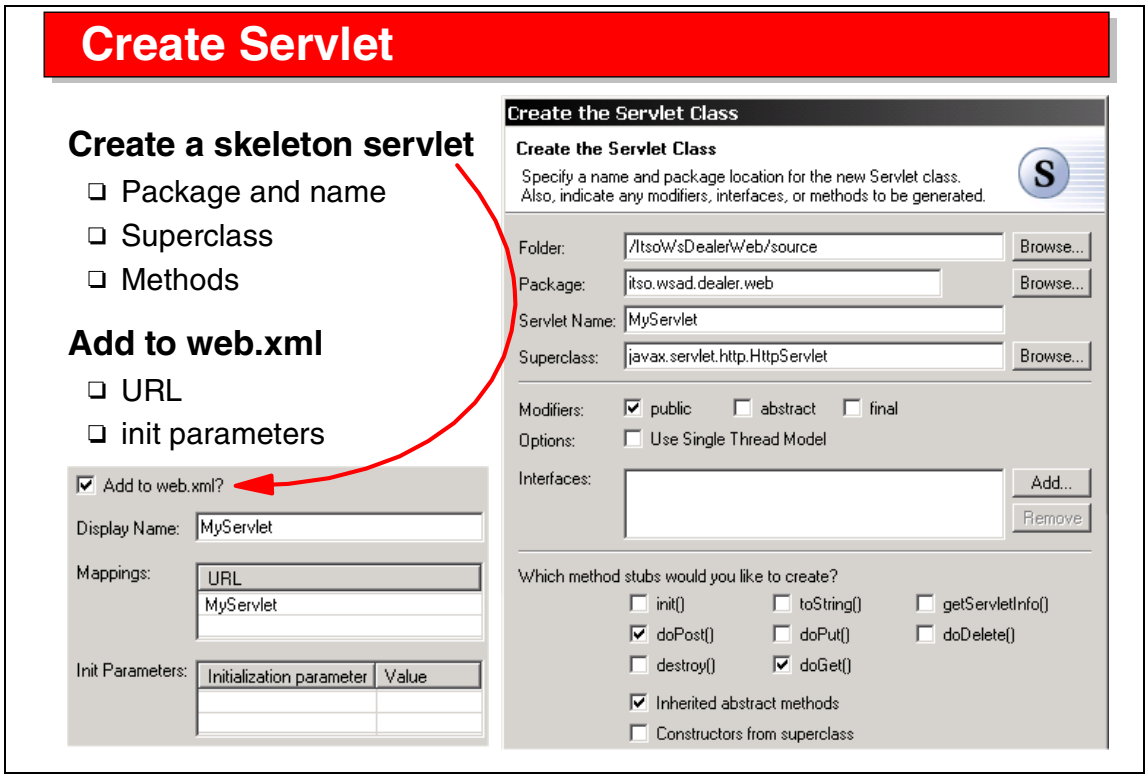
Visual 6-9 Editing of Web Resources

The Page Designer (that was available in WebSphere Studio classic) is the editor for HTML and JSP files:

- The Design view is for WYSIWYG construction.
- The Source view shows the HTML source.
- The Preview view shows the page as it appears in a browser.

Additional tools are the Animated GIF Designer, the Web Art Designer, and the Stylesheet editor. We will not discuss these tools in this class.

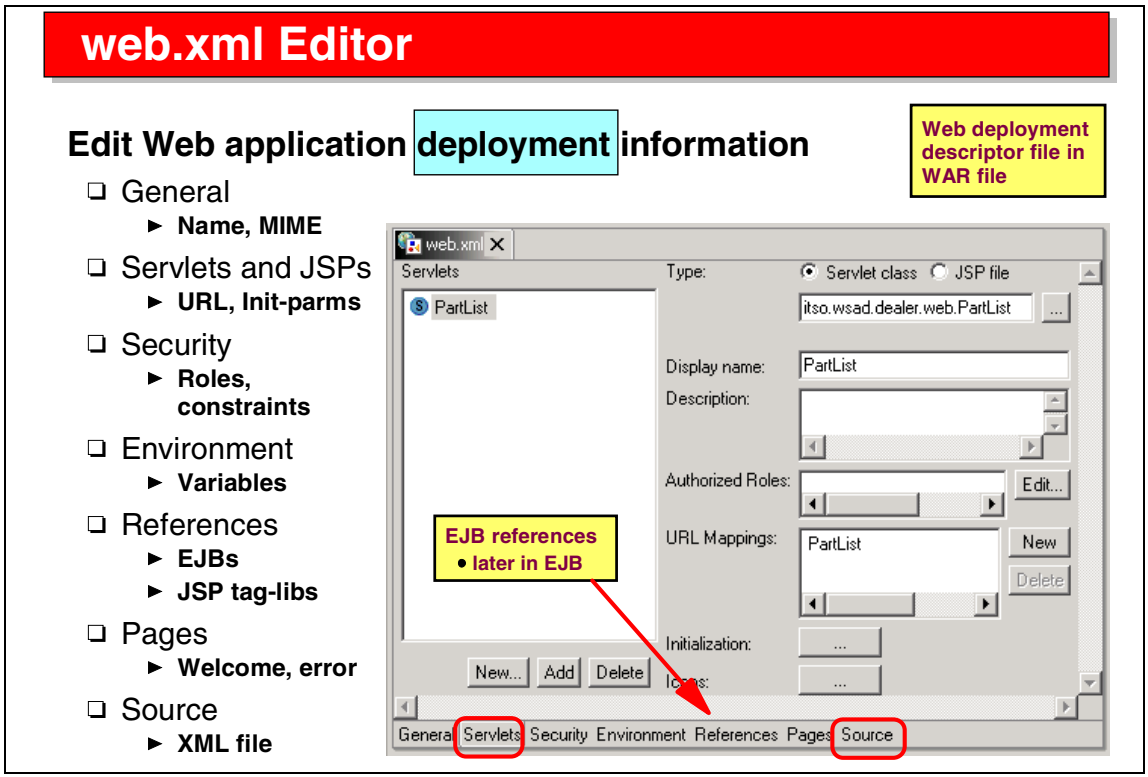
The Web application deployment descriptor is the web.xml file, where servlets and JSPs are defined together with additional deployment information.



Visual 6-10 Create Servlet

A SmartGuide is provided to define a servlet:

- You specify the folder, the package name, the name of the servlet, its superclass, modifiers, interfaces that must be implemented, and the method skeletons that should be generated.
- On the second page, you can add the servlet to the web.xml file and specify its alias name (for invocation) and initialization parameters and their values.

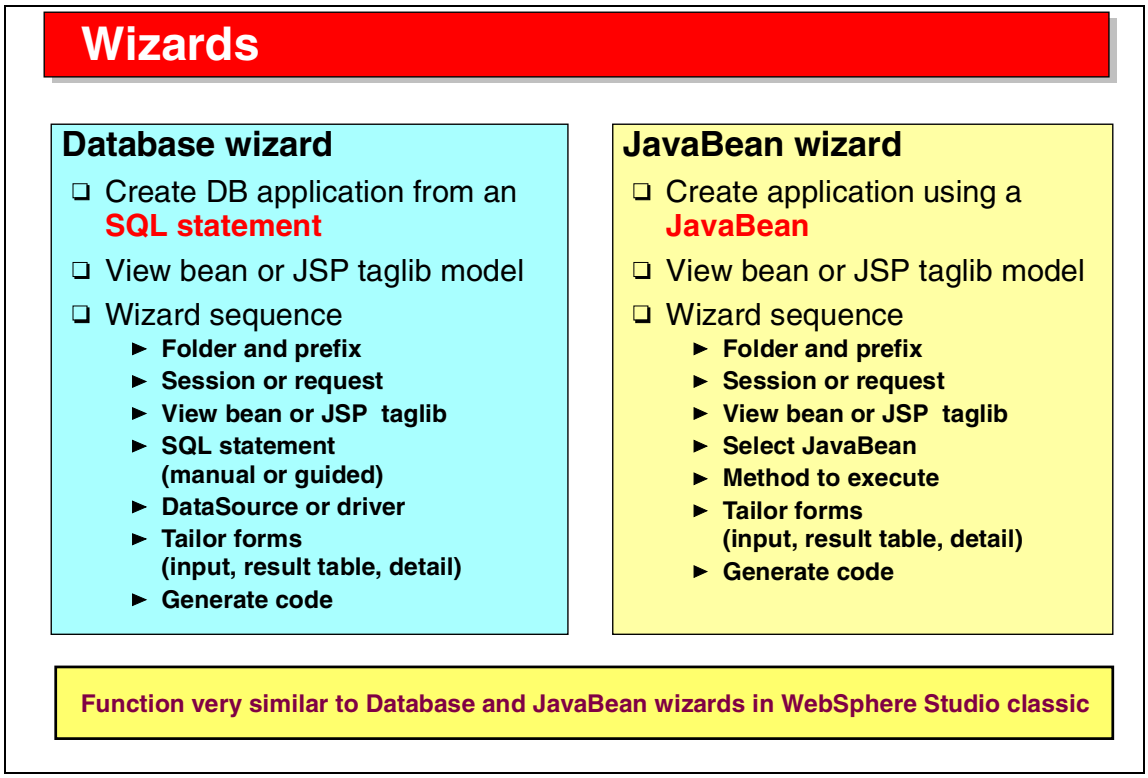


Visual 6-11 web.xml Editor

The web.xml editor is provided to maintain the deployment descriptor.

A number of panels (accessible by tabs) are provided. For example:

- On the General page, you can provide a description and mime type mappings.
- On the Servlets page, servlet and JSP information is maintained.
- On the References page, you can define references to EJBs that are used in the Web application.
- On the Pages page, you can define the default welcome page.
- On the Source page, you can edit the actual XML source (but this is not suggested).



Visual 6-12 Wizards

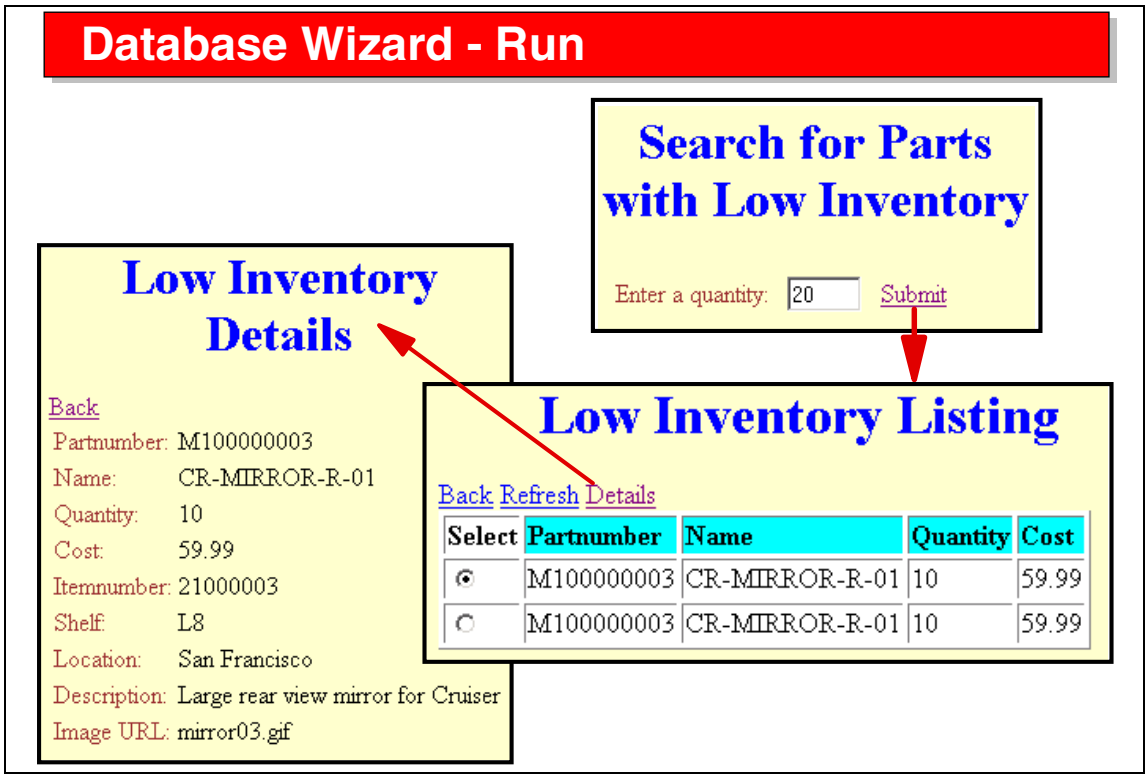
Two wizards are provided to generate skeleton Web applications:

- Database wizard—generates a Web application based on an SQL statement
- JavaBean wizard—generates a Web application based on a JavaBean

For both wizards, two models are supported:

- View bean model—generates a controller servlet, a JavaBean for processing, and JSPs for output
- JSP taglib model—generates a controller servlet and JSPs for processing and output

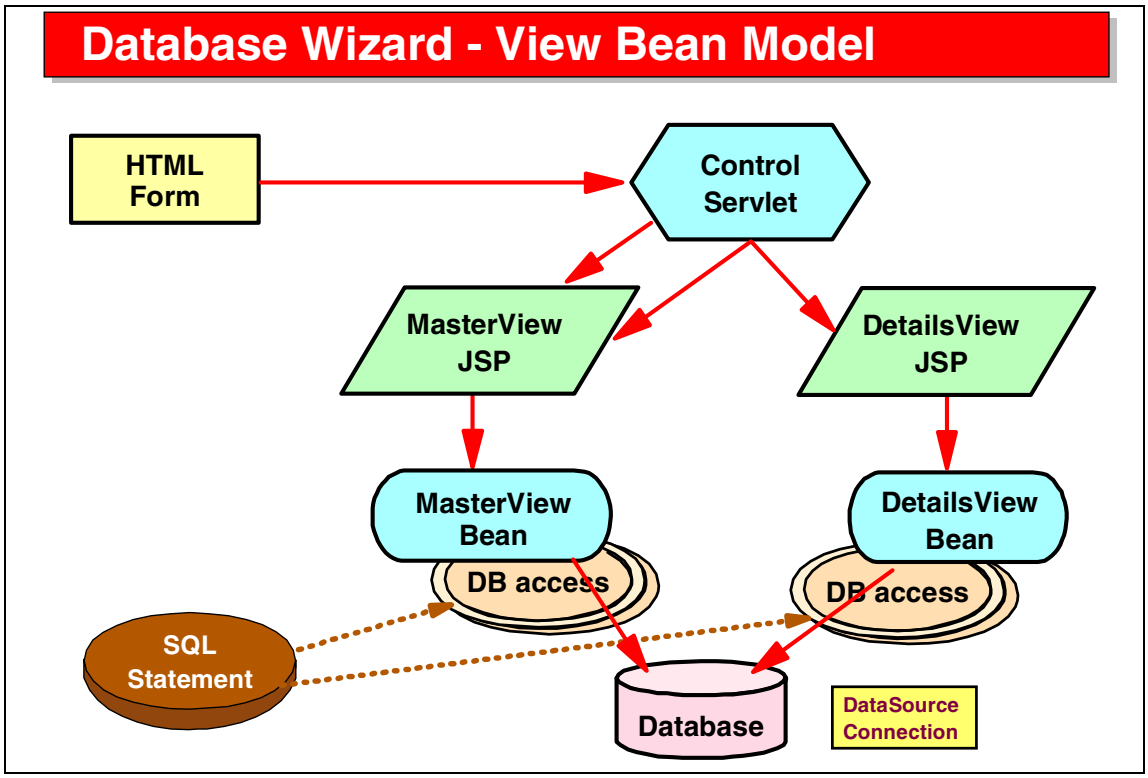
Each wizard guides the user through a series of panels where you specify the output folder, if a session should be used, the model, the SQL statement and data source (or JavaBean and method to invoke), the forms (input HTML, output JSPs). Then you generate the code.



Visual 6-13 Database Wizard - Run

Here we see a sample execution of a Web application generated with the database wizard:

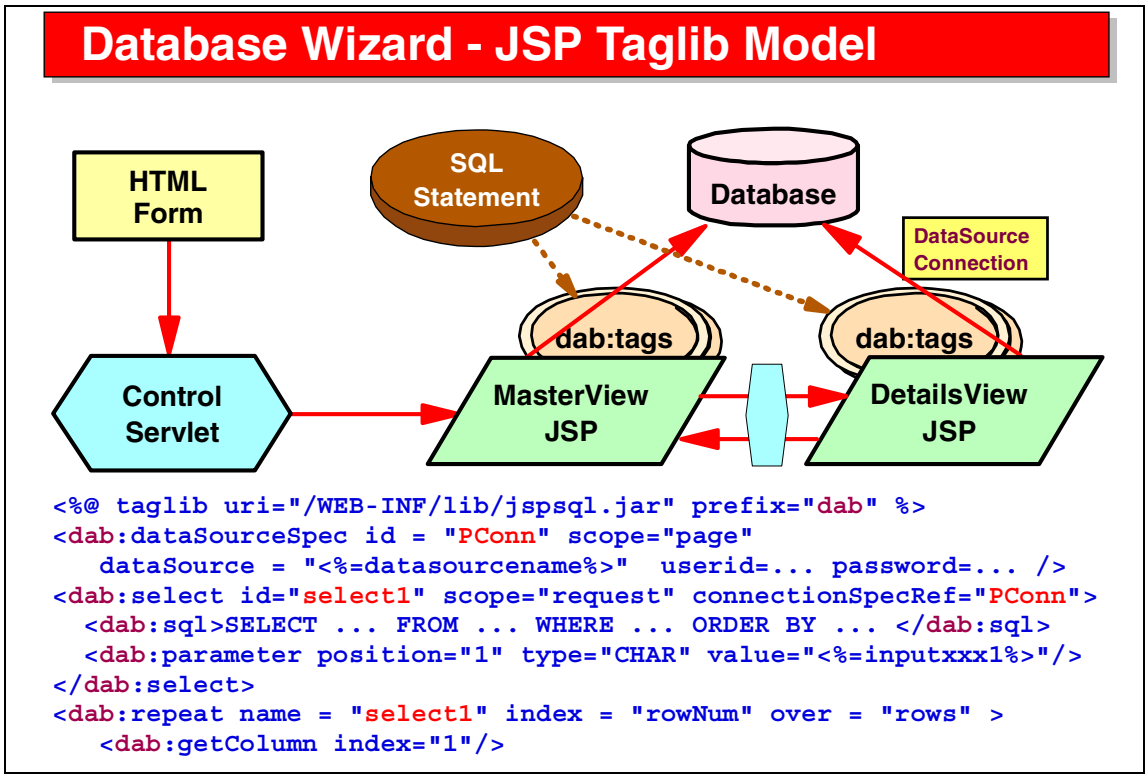
- The HTML input form is displayed and the user enters values for the input field(s).
- The list of matching database records is displayed in a table, usually with a subset of the columns retrieved by the SQL statement.
- One record can be selected and its details are displayed in a detail form that usually shows all the columns retrieved by the SQL statement.



Visual 6-14 Database Wizard - View Bean Model

When using the view bean model of the database wizard, the generated components include:

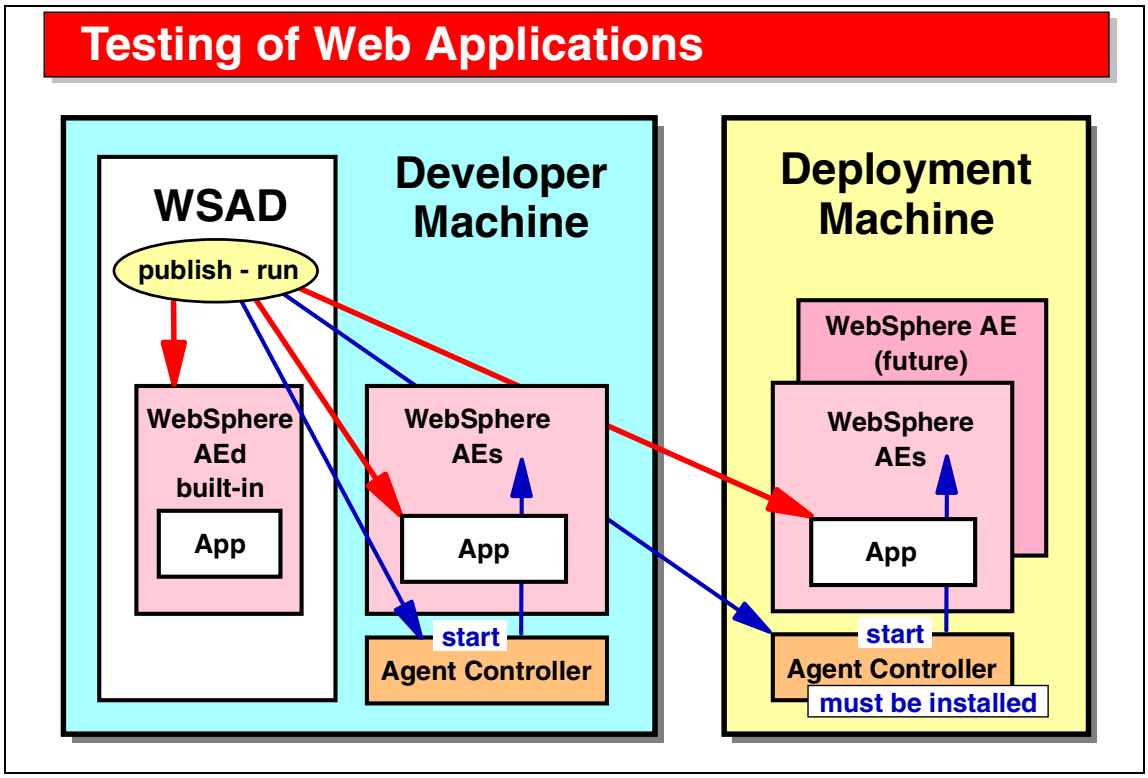
- HTML input form.
- Controller servlet, which invokes the master view or the details view JSP.
- Master view and details view JSPs, which invoke the master view or details view JavaBean for database access.
- The SQL statement is used to generate the database access code into the master and details view JavaBeans, using the data access bean technology from VisualAge for Java.
- The database is accessed using a data source that provides connection pooling.



Visual 6-15 Database Wizard - JSP Taglib Model

When using the JSP taglib model of the database wizard, the generated components include:

- HTML input form.
- Controller servlet, which invokes the master view or the details view JSP.
- Master view and details view JSPs, which invoke the master view or details view JavaBean for database access.
- The SQL statement is used to generate the database access code into the JSPs using special JSP tags to define the data source, run the SQL statement, repeat through the result set, and retrieve the column values.

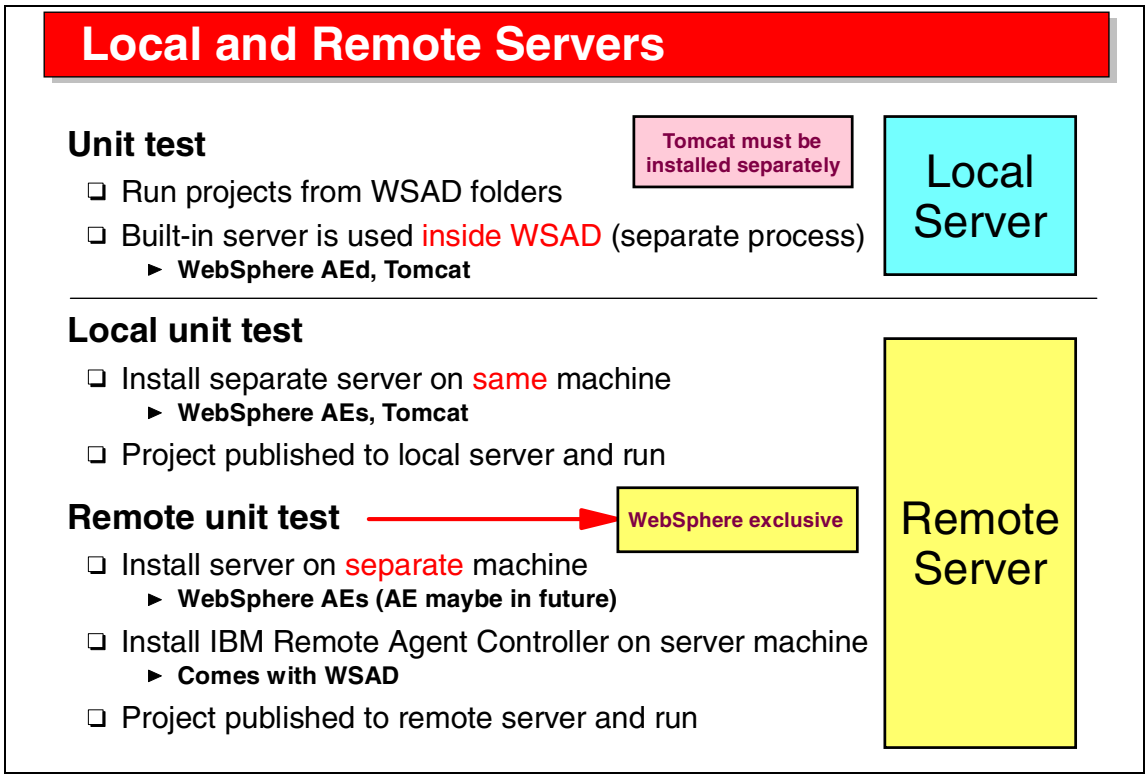


Visual 6-16 Testing of Web Applications

The Application Developer provides a local and remote test environment for testing of Web applications:

- WebSphere Application Server AEd (developer edition, same code as AEs, but free) is built into the Application Developer.
- WebSphere Application Server AEs (single server edition) can be installed on the same machine, or on a remote machine. A remote server is started through the IBM Agent Controller, which must be installed on the machine where the server runs.

For testing, a Web application is published to the selected server by installing the owning EAR project file into the application server. Then the server is started and the Web application can be tested in a Web browser.



Visual 6-17 Local and Remote Servers

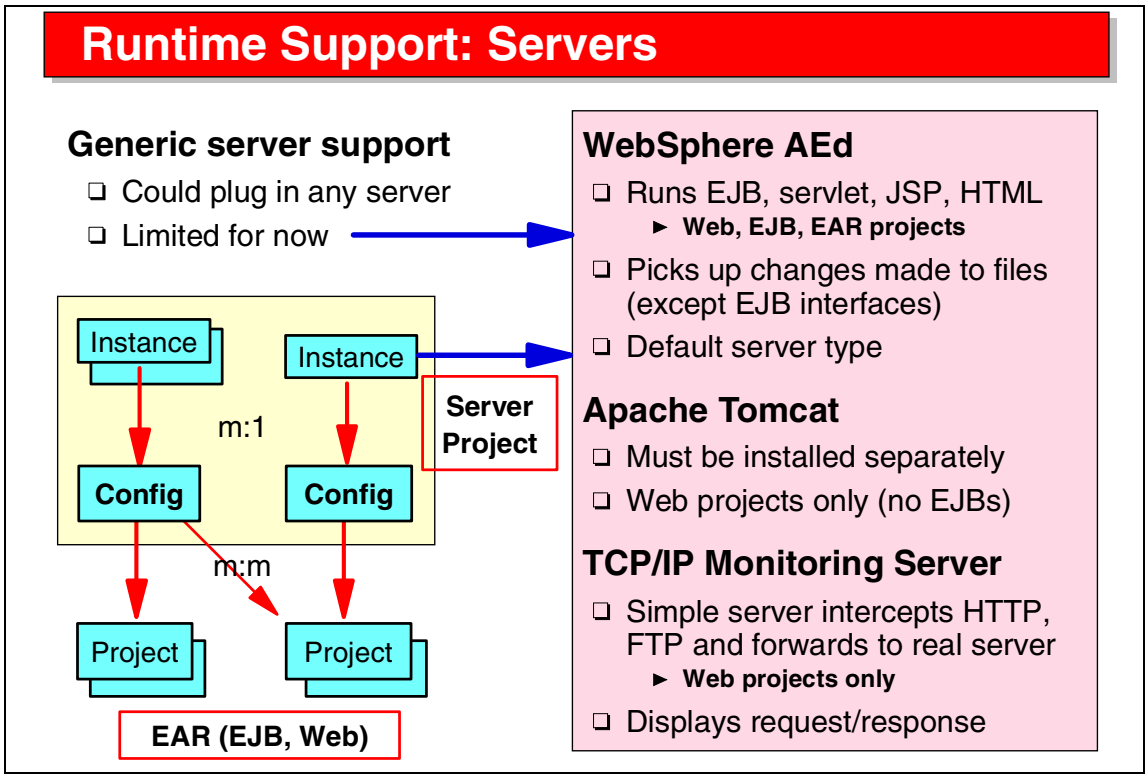
WebSphere AEs and Tomcat are supported for testing of Web applications.

If the server runs inside the Application Developer, we call it a local server. If the server runs outside, on the same or another machine, we call it a remote serer. A remote server is started through the Agent Controller.

Only WebSphere AEs can be used as a remote server on another machine.

Tomcat must be installed separately in all cases.

WebSphere AEs must be installed separately when used as a remote server on the same or another machine.



Visual 6-18 Runtime Support: Servers

Servers are defined in the Application Developer. To run a server you must have an instance of the server, and a server configuration.

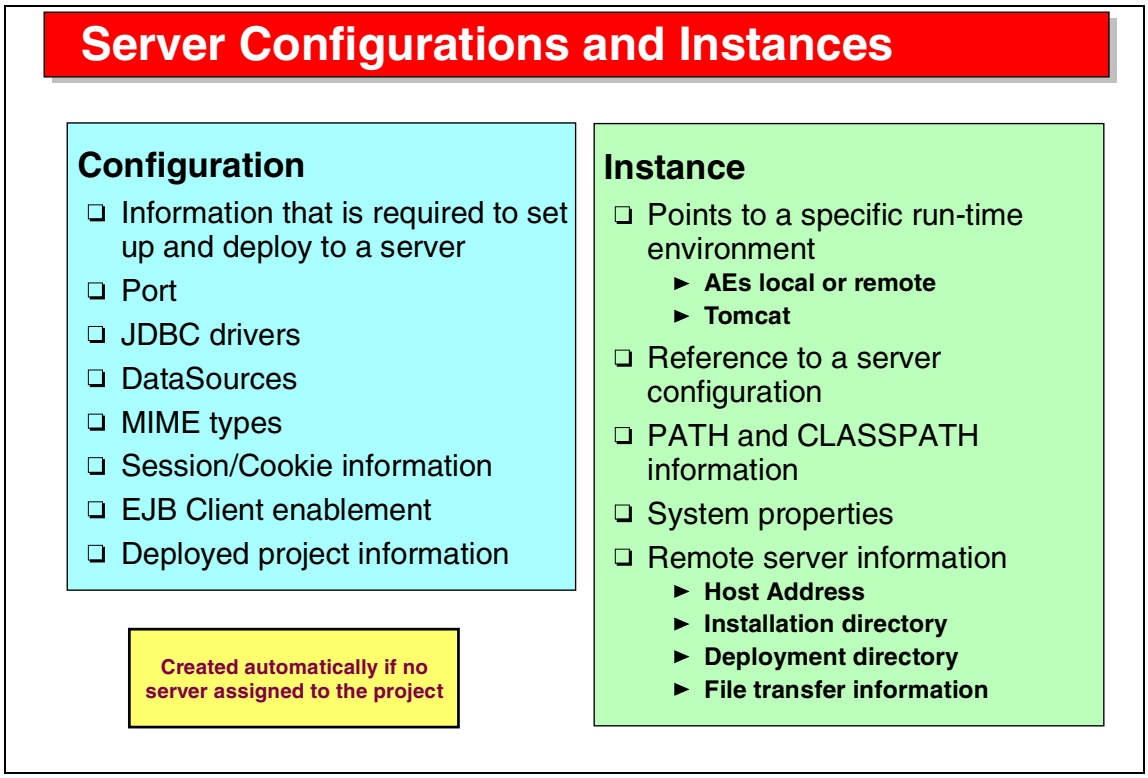
Server instances and configurations are defined in Server projects, which can be shared between developers.

Multiple instances can point to the same configuration.

EAR projects (with contained EJB and Web projects) are associated with server configurations. A configuration can be associated with multiple projects, and a project can be associated with multiple configurations, one of which will be the preferred configuration.

When a server is started, all associated projects are loaded. When a project is run on a server, the preferred server is used (or started).

The TCP/IP Monitoring Server can be used to intercept and display the actual messages between browser and server.



Visual 6-19 Server Configurations and Instances

A server configuration specifies:

- Information about the server facilities, such as ports that are used, JDBC drivers to be loaded, data sources to be defined, mime types, if sessions or cookies are used, and if the universal test client should be loaded.

A server instance specifies:

- The runtime environment (WebSphere AEs or Tomcat), PATH and CLASSPATH information, system properties (for example, JIT compiler), and how projects are deployed to a remote server.
- A remote server requires information about the host address, the installation and deployment directories, and how files are transferred (either through LAN copy or through FTP).

Runtime and Test Configurations

Server project

- For team sharing of configurations

Server consists of configuration and instance

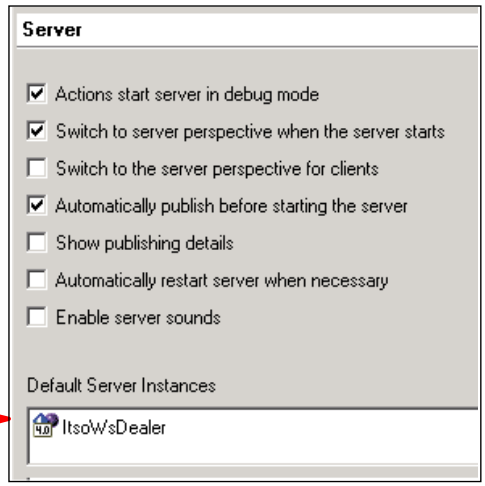
- Automatic for simple scenarios
- Manual in most real scenarios
  - **Can define and reuse templates**

Templates

- Save a configuration as a template
  - **Window -> Preferences -> Server -> Templates**
- Saves time to create multiple similar configurations

Server preferences

- **Window -> Preferences -> Servers**

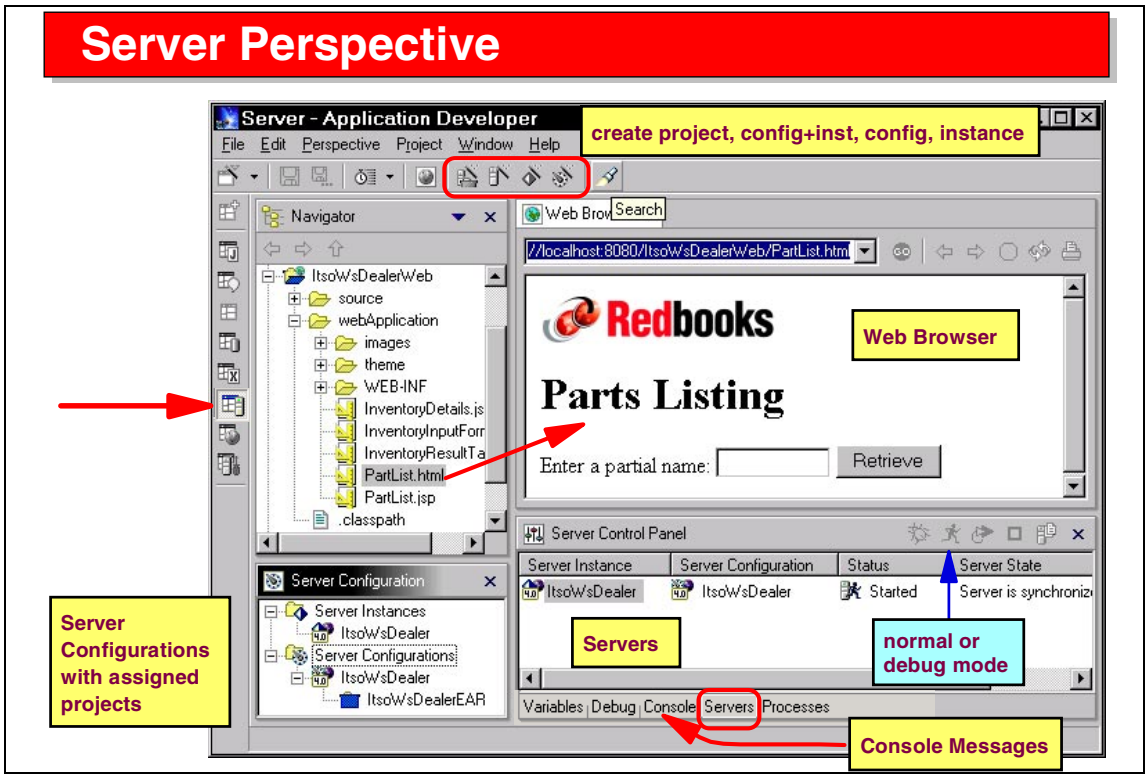


Visual 6-20 Runtime and Test Configurations

A Server project is used to keep server definitions. Such a project can be shared and versioned in a team environment.

Servers can be defined automatically for simple projects, but in most cases tailored servers are defined for a set of projects that run on the same server.

A server definition can be stored as a template for easy definition of additional servers with the same, or similar, characteristics.



Visual 6-21 Server Perspective

In the Server Perspective we maintain definitions of application servers for testing of Web applications, EJBs, and Web Services.

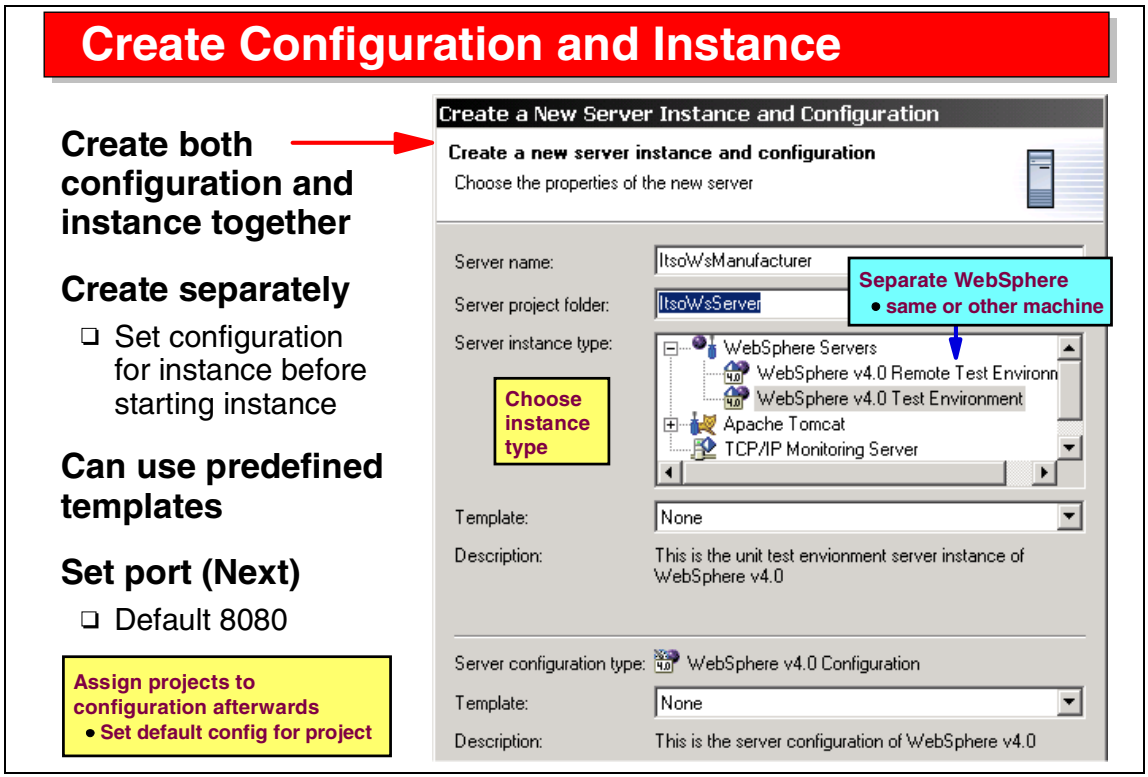
Server configurations define the type of server (WebSphere or Tomcat), and are configured with JDBC drivers and data sources.

Projects are associated with servers. When a server is started, the associated projects are loaded and their code can be executed.

Servers can be started in normal or debug mode. In debug mode, breakpoints can be placed into servlets and JSPs for debugging purposes.

Icons are provided to create a server project or server instances and configurations.

You can use the Server Perspective to edit resources and run or debug projects.



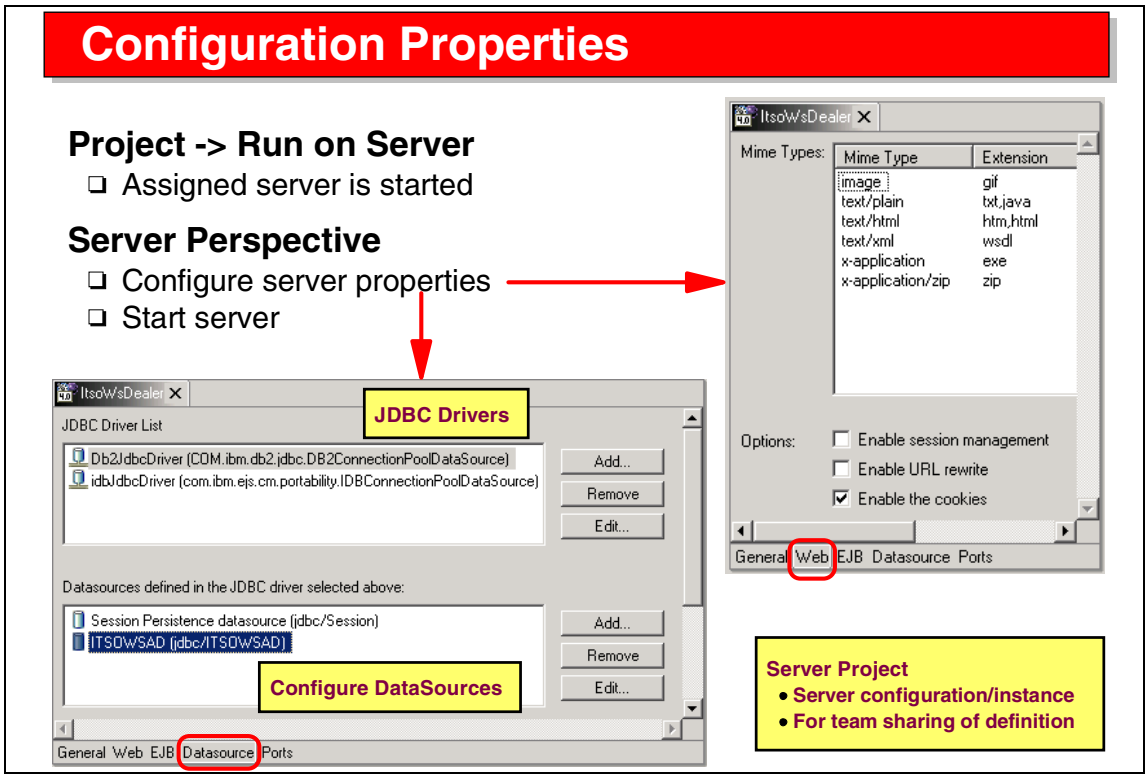
Visual 6-22 Create Configuration and Instance

A SmartGuide is provided to define a server instance and configuration. You have to specify:

- Name of the server
- Server project
- Instance type (WebSphere AEs or Tomcat)
- Port (on the next panel)

For a remote server, additional panels are required. (This will be discussed later in the class).

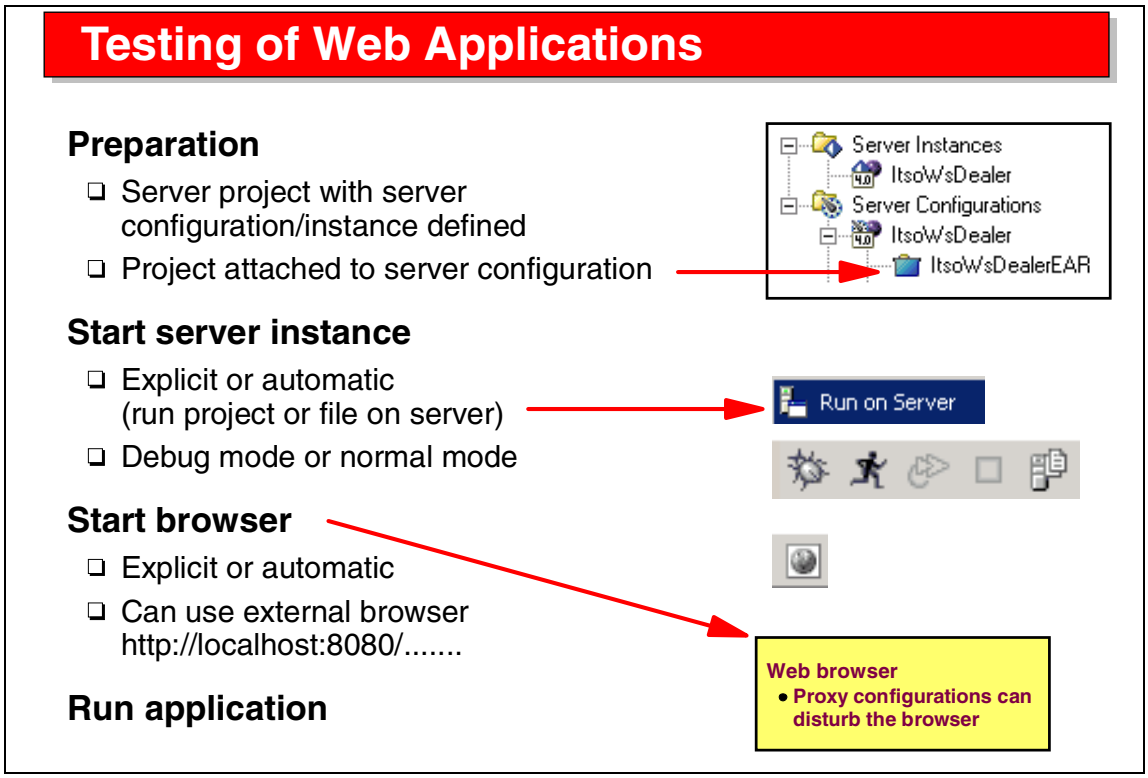
After defining a server you can associate EAR projects with the server configuration, and you can assign a preferred (default) server for each project.



Visual 6-23 Configuration Properties

The server configuration properties are maintained in a special editor:

- Mime types
- Enable session management
- Enable URL rewrite
- Enable cookies
- Define JDBC drivers to be loaded (click *Add* and specify the class and location of the ZIP/JAR file)
- Define data sources for JDBC drivers (click *Add* and define the JNDI name, database name, and connection pooling information)



Visual 6-24 Testing of Web Applications

Testing a Web application involves these steps:

- Define a server and associate the EAR project that contains the Web project with the server
- Start the server in normal or debug mode. You can simply select the project and *Run on Server* to start the preferred server in debug mode, or you can start the server manually.
- Start a Web browser by selecting an HTML file and *Run on Server*, or start a Web browser manually (inside the Application Server or outside) and enter a URL.

Debugging of Web Applications

Start server in Debug mode

Set breakpoint in servlet or JSP

- JSP breakpoints at JSP tags only

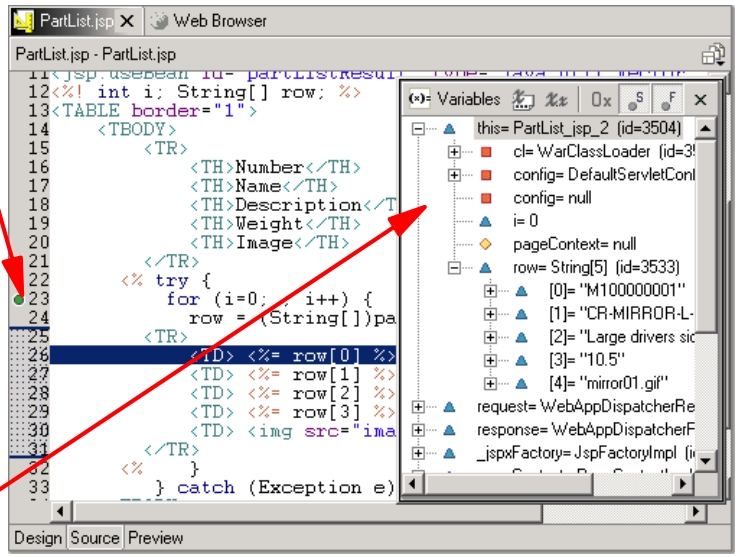
Run Web app

Debug servlet

- Same as Java

Debug JSP

- JSP source code
- Java source is hidden
- Variables visible



Visual 6-25 Debugging of Web Applications

To debug a Web application, you start the server in debug mode.

In debug mode you can set breakpoints in servlets and JSPs, step through the code using the standard debug icons (step into, step over, step return), and monitor variables.

Debugging Java code is the same as for a Java project.

Debugging a JSP is performed at the source code level. You can set breakpoints only at JSP tag lines, not in HTML code. The variables of the JSP servlet are visible in the Variables view.

Summary

Web projects and Perspective provide

- Web development environment
- J2EE-conforming deployment
- Page Designer for HTML and JSP
- Wizards for code generation

Server project and Server Perspective provide

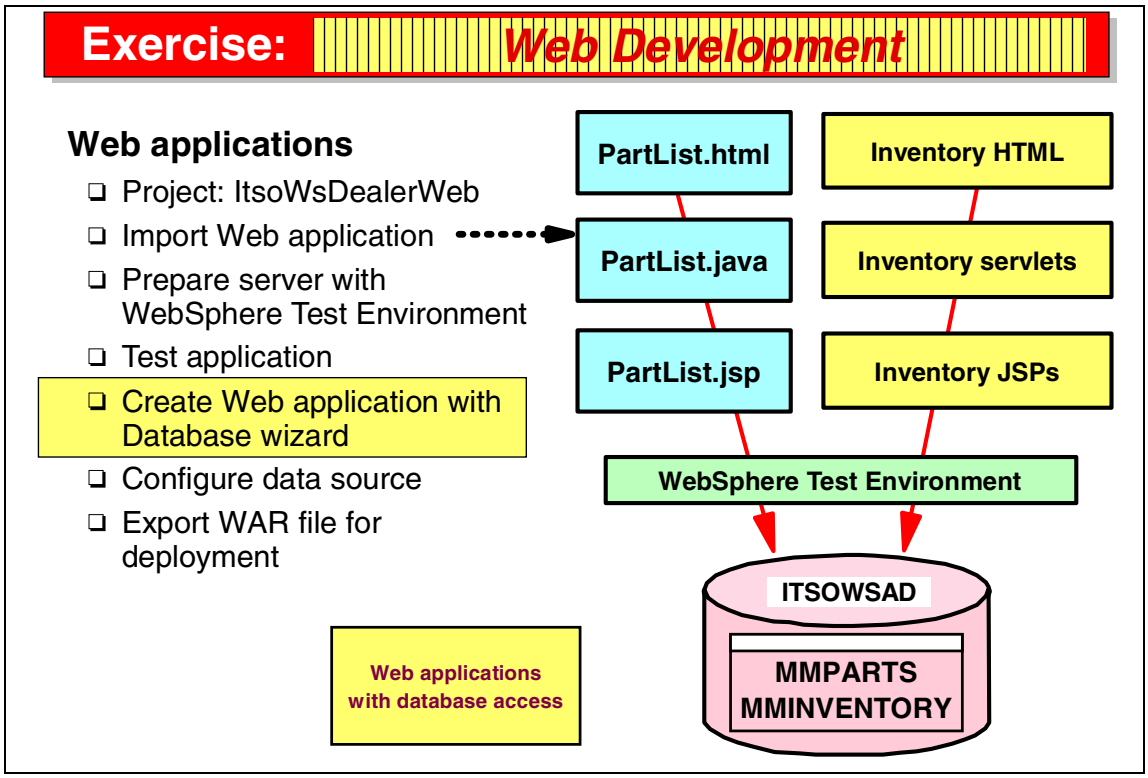
- Test environment for Web applications
- Server configurations and instances
  - **JDBC drivers and DataSource**

Visual 6-26 Summary

The Application Developer provides very good tooling for Web application development and testing.

The setup is very effective for J2EE-enabled application servers because the project setup in the Application Developer mirrors the deployment information required for J2EE.

The built-in test environment makes testing and debugging of Web applications very easy.



Visual 6-27 Exercise: Web Development

The Web development exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with two applications:

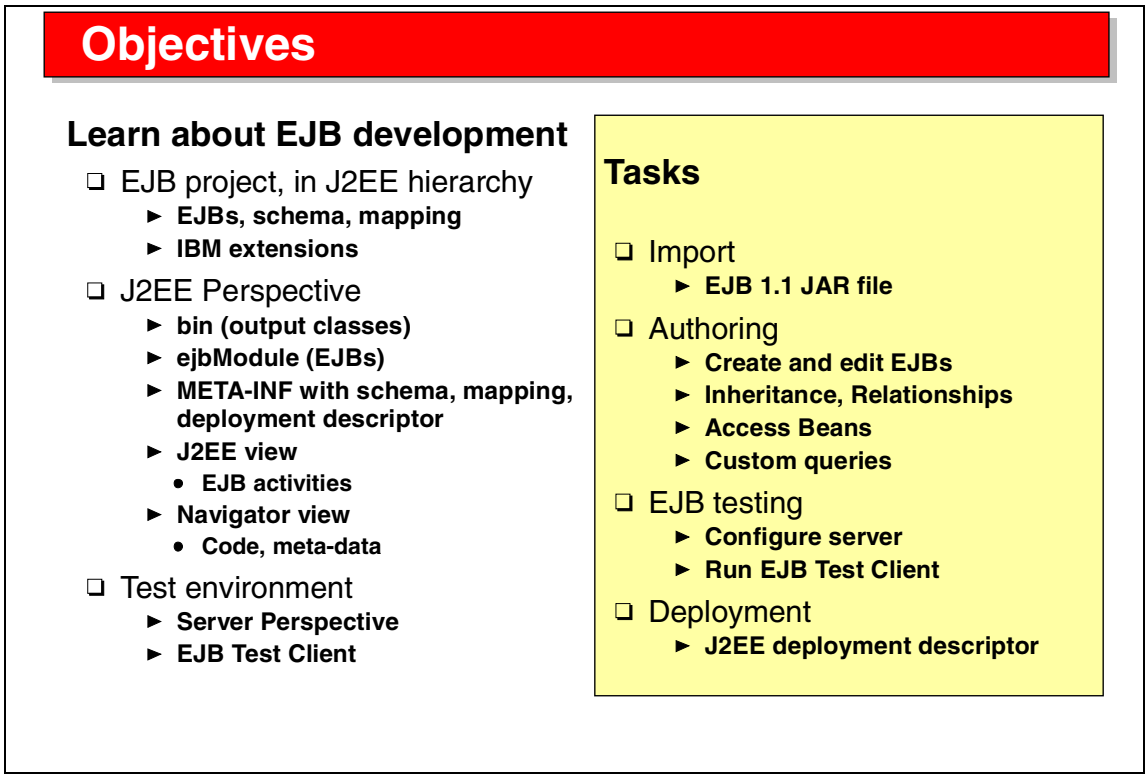
- PartList—Web application with a servlet and a JSP with JDBC access to the ITSOWSAD database
- Inventory—Web application generated with the database wizard

See Exercise 6, “Application Developer: Web Development” on page 117 for the instructions for this exercise.

7

Application Developer: EJB Development

Visual 7-1 Title



Visual 7-2 Objectives

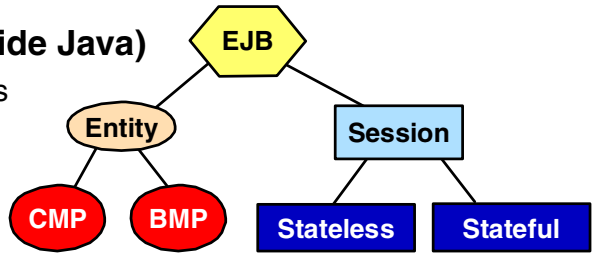
The objectives of this unit are to:

- Understand the EJB development environment provided by the Application Developer
- Understand the EJB project and J2EE Perspective
- Understand the test environment, including the EJB test client

EJB Review

EJB specification (server-side Java)

- Defined by Sun Microsystems
- Compatible with CORBA



There are 2 types of EJB

- **Entity** Bean
  - **This EJB represents persistent business data, such as customer, account, ...**
- There are 2 types of Entity EJBs

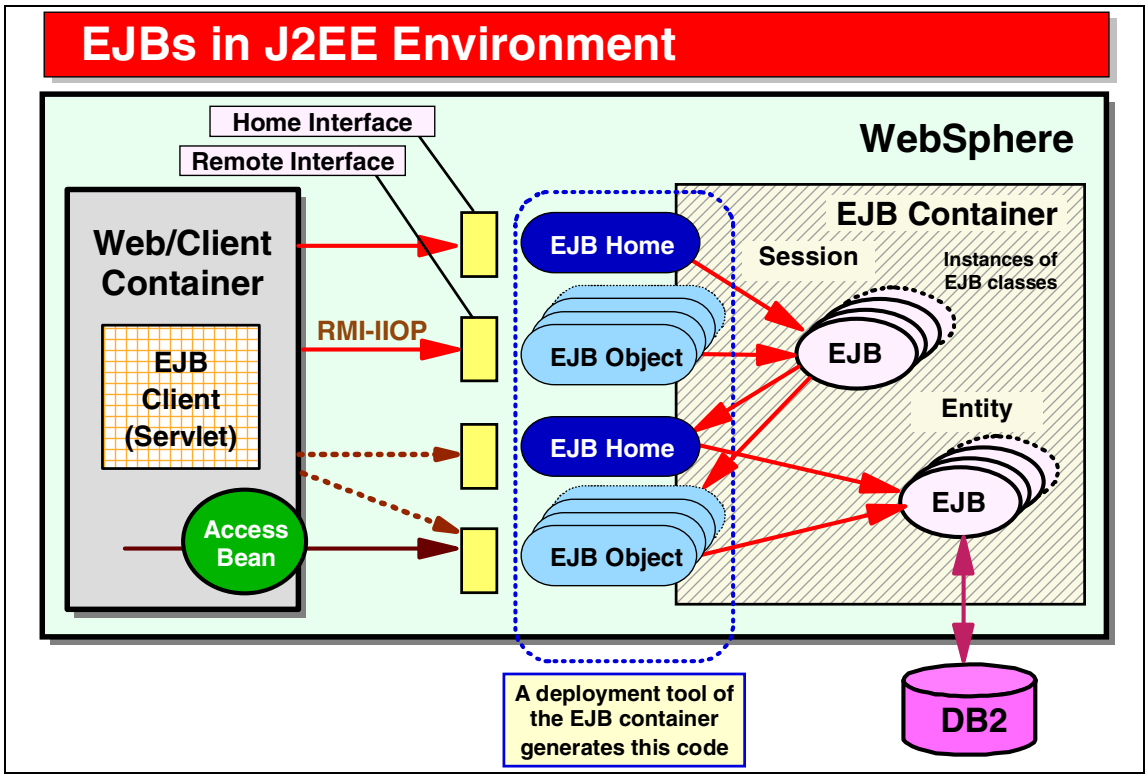
  `CMP (Container-Managed Persistence) entity bean`
  - **EJB developer specifies mapping to relational database**
  - **An EJB container, such as WebSphere, provides persistence**

  `BMP (Bean-Managed Persistence) entity bean`
  - **EJB developer must develop persistence layer**
- **Session** Bean
  - **This EJB executes business logic on behalf of a single client in a server**
  - **For example, you can implement a set of transactions in a session bean**
  - **stateless (shared) or stateful (for one user)**

Visual 7-3 EJB Review

The EJB specification provides for entity and session EJBs.

In this class we only deal with container-managed persistence entity EJBs that are mapped to relational tables, and stateless session EJBs that are used for business logic and transaction management.



Visual 7-4 EJBs in J2EE Environment

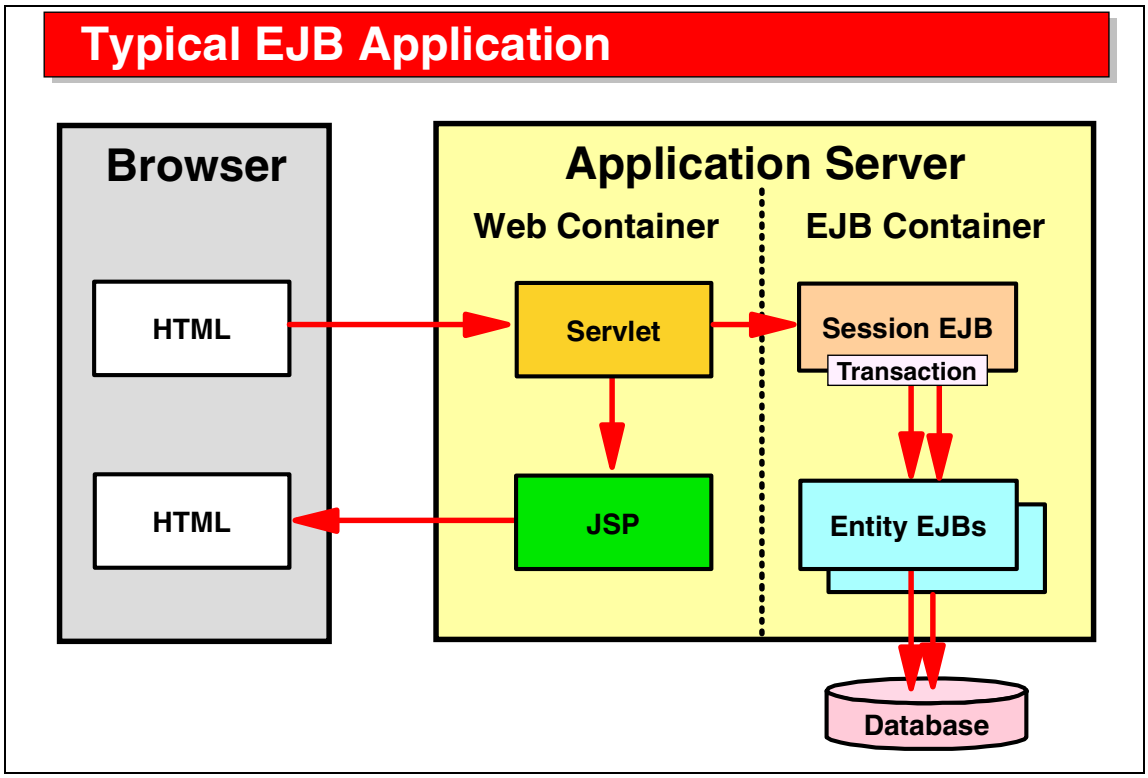
The EJB specification dictates that EJBs run in an EJB container, which is usually provided by an application server.

Clients connect to EJBs through the RMI-IIOP protocol using two interfaces:

- The home interface that gives access to EJB instances through create (for new beans) and find (for existing beans) methods.
- The remote interface that is used to access the entity instances created or found through the home interface.

Each EJB has a JNDI (Java Naming and Directory Interface) name, which is used to acquire the home interface.

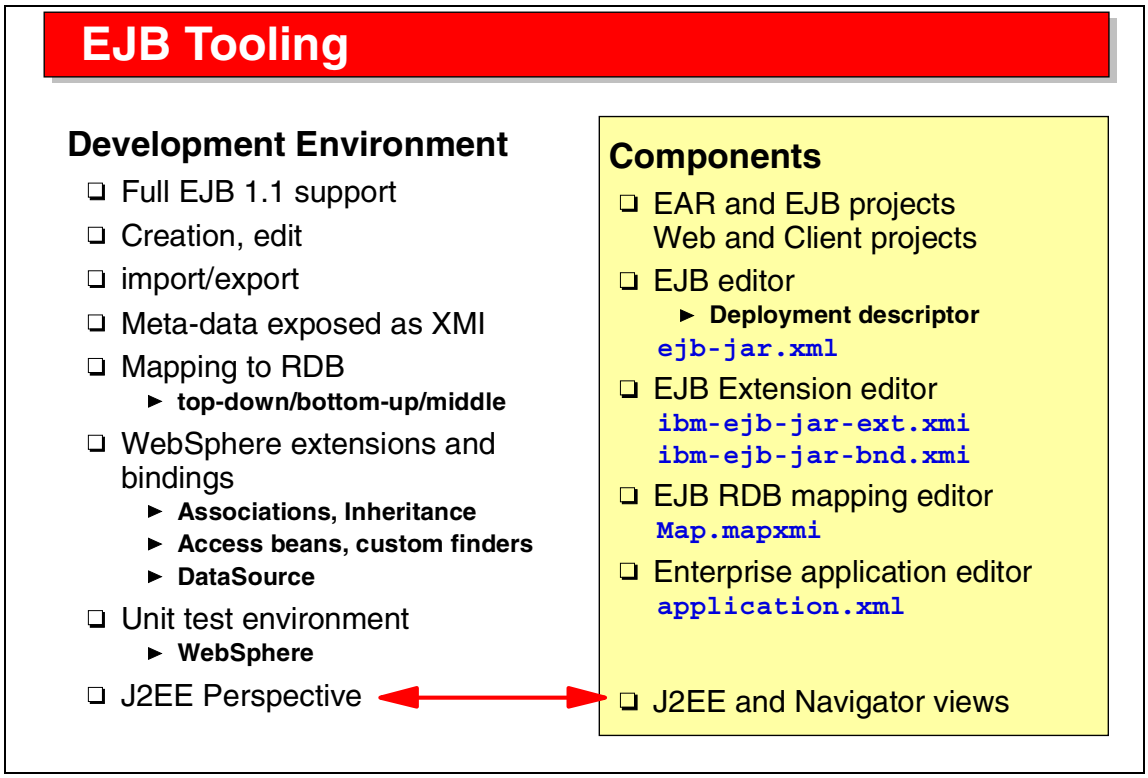
Notice that session EJBs access entity EJBs through the home and remote interfaces as well.



Visual 7-5 Typical EJB Application

This diagram shows the structure of a typical (simplified) EJB-based Web application:

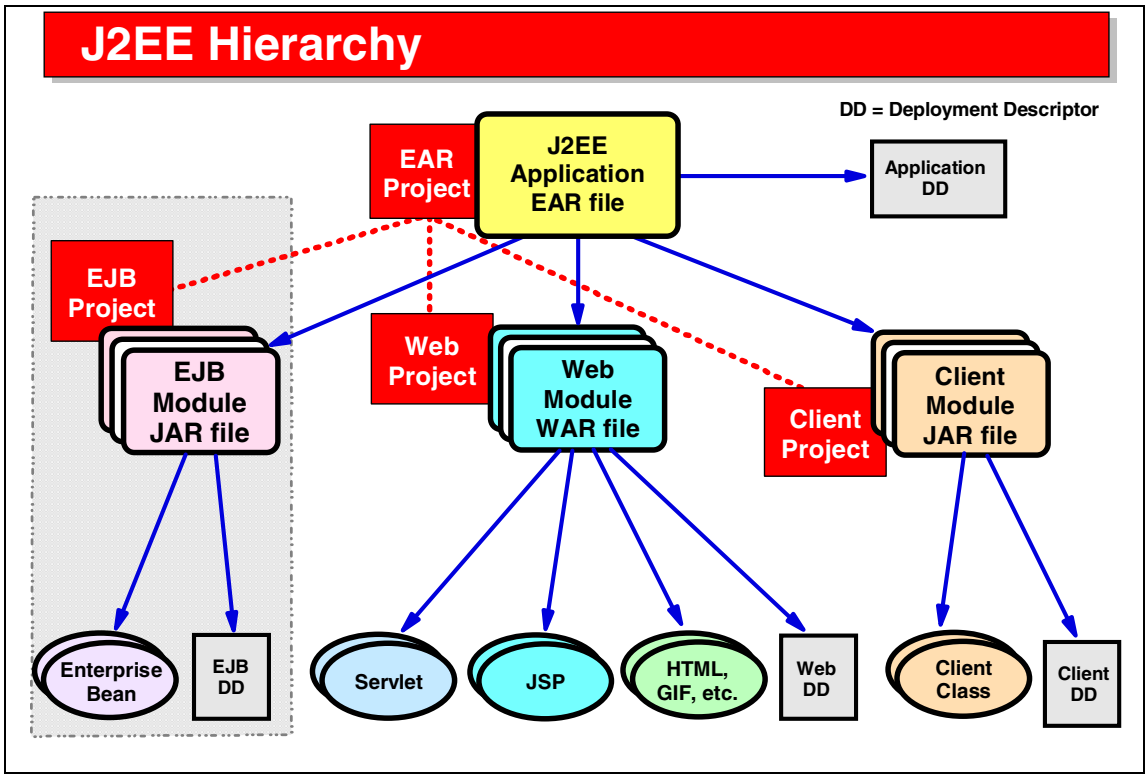
- A servlet is invoked from an HTML form.
- The servlet invokes a method in a session EJB (first the home of the session EJB is acquired and a session bean instance is created). This starts a transaction.
- The session EJB accesses multiple entity EJBs for database retrieval and update. All business logic is in the session bean.
- When the session bean method ends, the transaction is committed and the database updates are made permanent (this may require two-phase commit if multiple database managers are involved).
- The servlet invokes a JSP to produce HTML output (not shown are additional beans need to pass the data to the JSP.



Visual 7-6 EJB Tooling

The EJB tooling support provided by the Application Developer includes full support of the EJB 1.1 specification. In addition IBM extensions for associations, inheritance, access beans, and custom finders are supported.

All the control information is kept in EJB deployment descriptors: the ejb-jar.xml file, and IBM extension files (ibm-ejb-jar-ext.xmi, for example). Editors are provided to maintain the deployment descriptors.

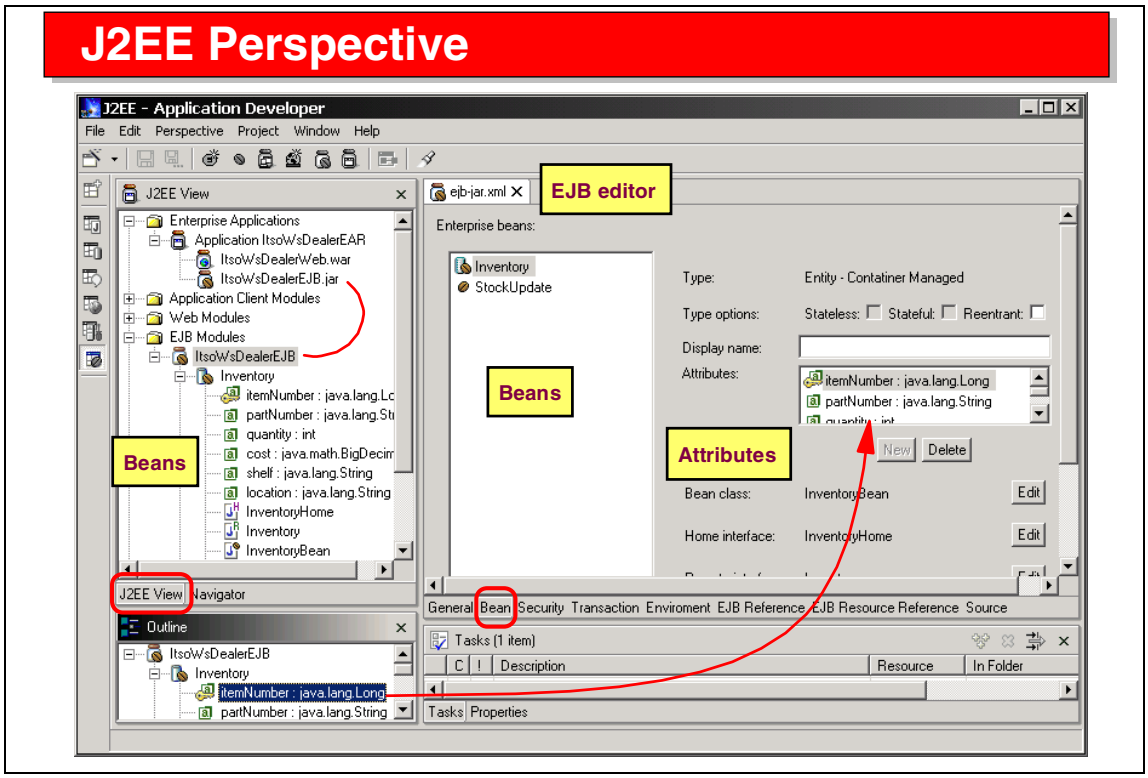


Visual 7-7 J2EE Hierarchy

This diagram shows the J2EE hierarchy and the matching support in the Application Developer:

- A J2EE application is stored in an enterprise archive (EAR) file that contains EJB modules (stored in an EJB JAR file), Web modules (stored in Web archives (WAR) files), and client modules (stored in a JAR file).
- Each of the modules contains a deployment descriptor; for example, an EJB JAR file contains the ejb-jar.xml file.
- The EJB JAR file contains all the interfaces and deployed code that make up entity and session EJBs.
- The J2EE hierarchy is matched by projects in the Application Developer. An EAR project contains references to EJB, Web, and client projects. An EJB project contains all definitions of the EJBs, including the mapping to relational tables.

This setup makes deployment to a J2EE-based application server very easy.



Visual 7-8 J2EE Perspective

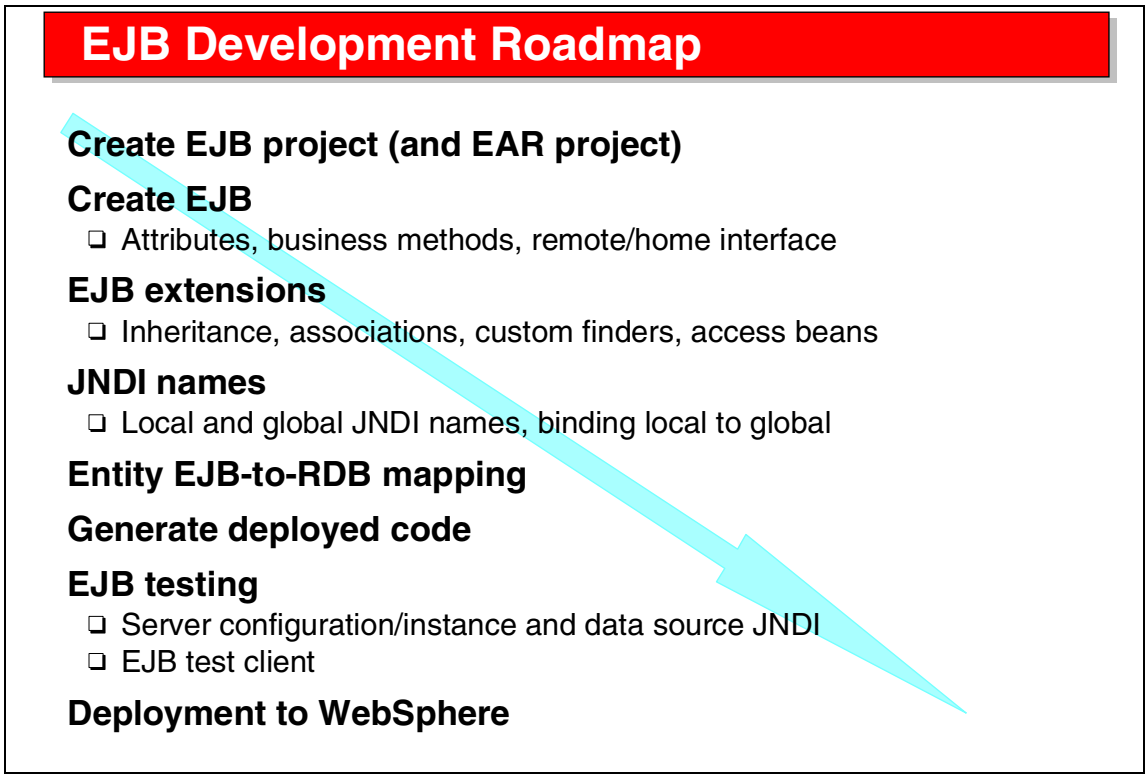
The J2EE Perspective is used for management of J2EE deployment descriptors (EAR, enterprise archives), and for development of EJBs.

The J2EE view is the only view where entity and session EJBs can be developed. This view displays a logical view of the EJBs with their fields, key, and main underlying Java files (bean class, home and remote interface, key class).

The Navigator view displays all the project resources, including the control files (XMI files) that are used to store the EJB design (meta) information.

An EJB Editor is provided to define and manipulate EJB deployment information, such as JNDI names, transaction attributes, read-only methods, and security information.

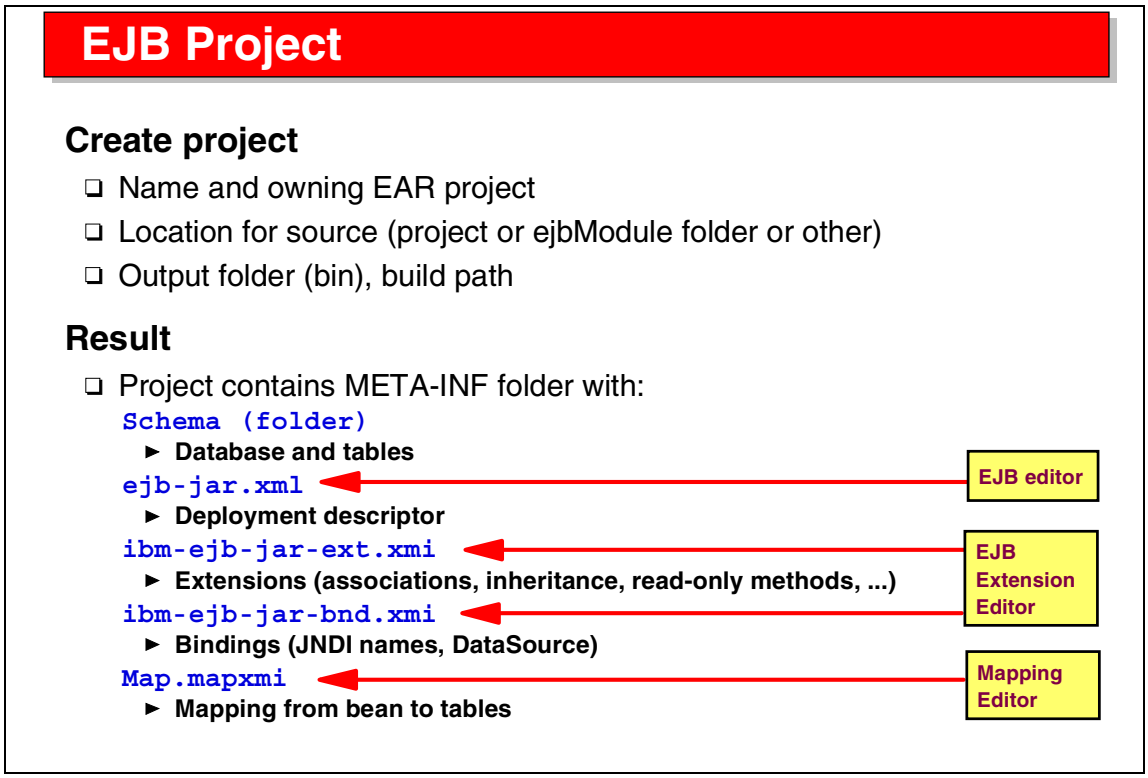
An EJB Extension Editor is provided to define IBM extension of the EJB specification, such as associations and custom finders.



Visual 7-9 EJB Development Roadmap

The steps involved to create the EJB-based application are:

- Create an EJB project to hold all the definitions and deployed code.
- Create entity and session EJBs with attributes, business methods, home and remote interface.
- Define IBM-supported EJB extensions.
- Define the JNDI names used to acquires EJB home interfaces. In EJB 1.1, applications should use local JNDI names that are then mapped to global (unique) names in an application server.
- Map entity EJBs to relational tables.
- Generate the deployed code from the definitions and the mapping.
- Define a server for testing, and configure data sources for EJB access in the database. Add the project to the server, start the server, and use the test client to instantiate EJBs and run methods.
- Deploy EJB applications to an application server.



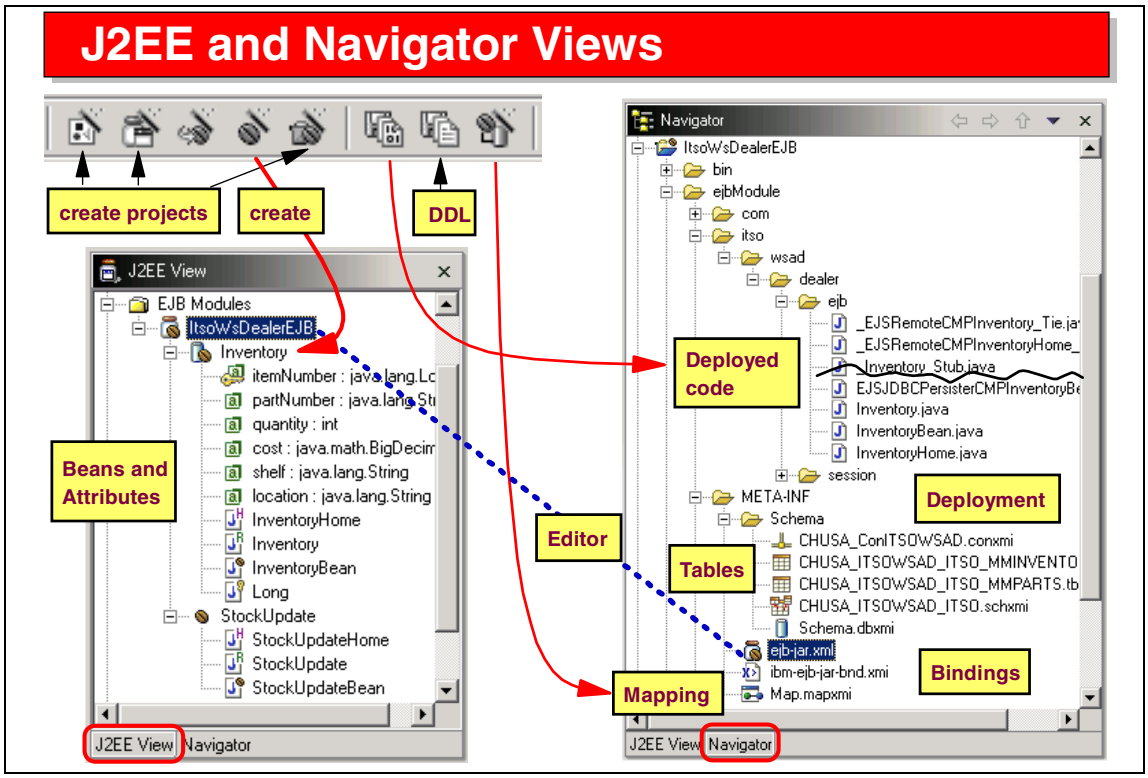
Visual 7-10 EJB Project

An EJB project must be attached to an EAR project.

As for a Java project, you have to define the source and output folders and the build path that is used to compile the Java source code.

The EJB project contains a META-INF folder for the control information:

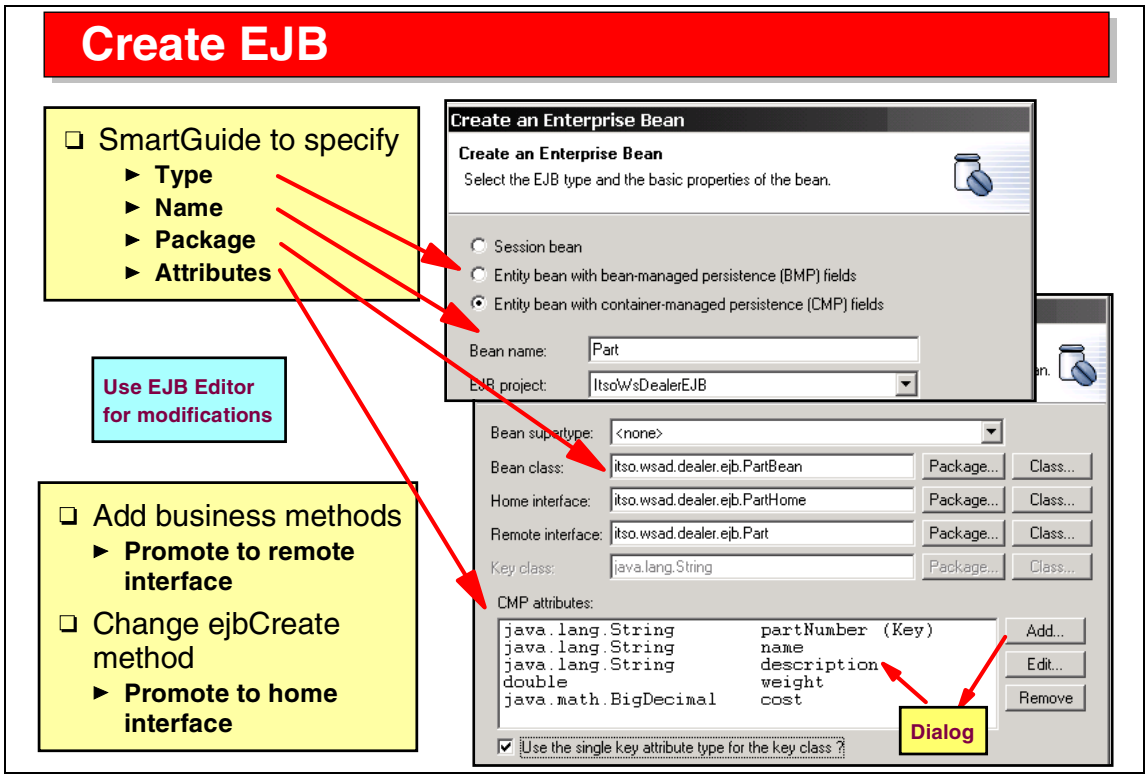
- Schema (tables) used in the mapping
- EJB deployment descriptor
- IBM extension deployment information (XMI files)
- Mapping file for entity-to-table mapping



Visual 7-11 J2EE and Navigator View

The J2EE Perspective comes with two main views:

- The J2EE view for design and specification of EJBs. This view shows the logical structure of the EJB modules and only the main files:
  - The home interface
  - The remote interface
  - The bean itself
  - The key class that holds the key attribute of an entity EJB
- The Navigator view that shows all the folders and files:
  - The bin folder with the class files
  - The ejbModule folder with the Java source code
  - The META-INF folder with the schema, deployment descriptors, and mapping



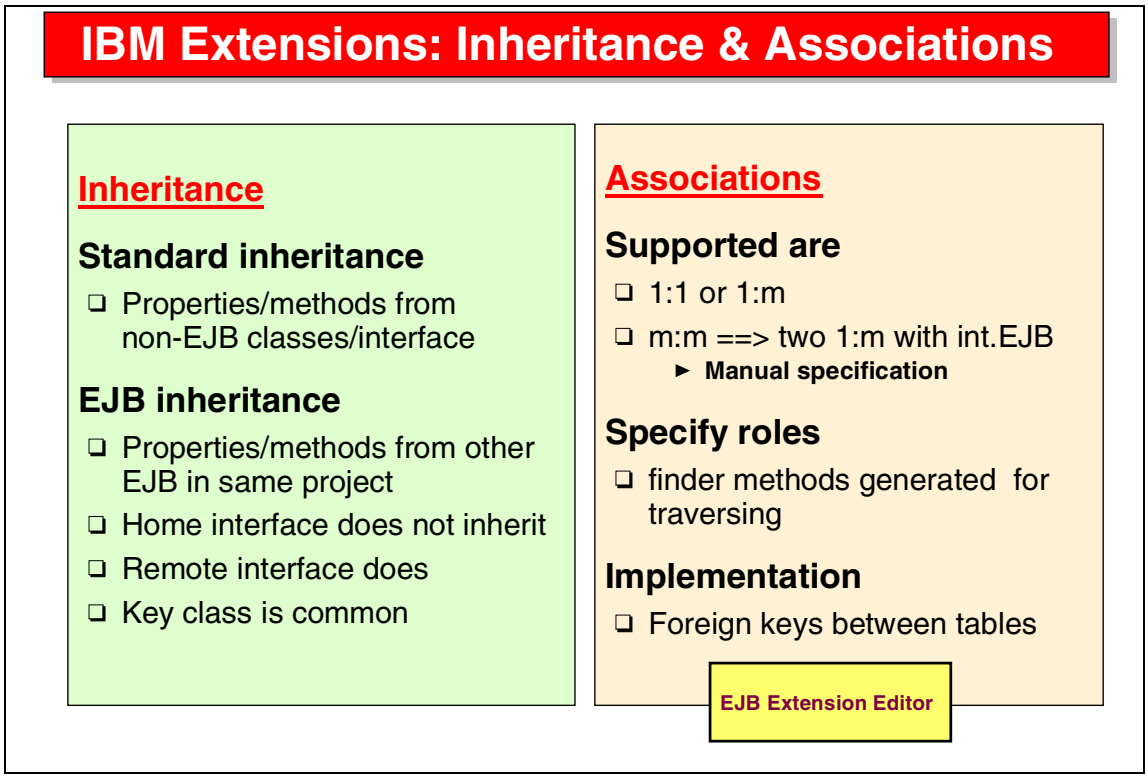
Visual 7-12 Create EJB

A SmartGuide is provided to define an EJB. This visual shows the definition of an entity bean (container-managed). The user specifies:

- The name and type
- The project and package
- The bean name
- The superclass for beans with inheritance
- The home and remote interface names are derived automatically
- Persistence fields (click *Add* to define fields)
- The key field can be embedded in a key class, or used directly

At the end of the SmartGuide, the Java code for bean, home interface, remote interface, and key are generated.

Typical follow-on tasks include defining business methods (for the remote interface) and adding tailored create methods (for the home interface).



Visual 7-13 IBM Extensions: Inheritance and Associations

The EJB 1.1 specification does not support inheritance and associations.

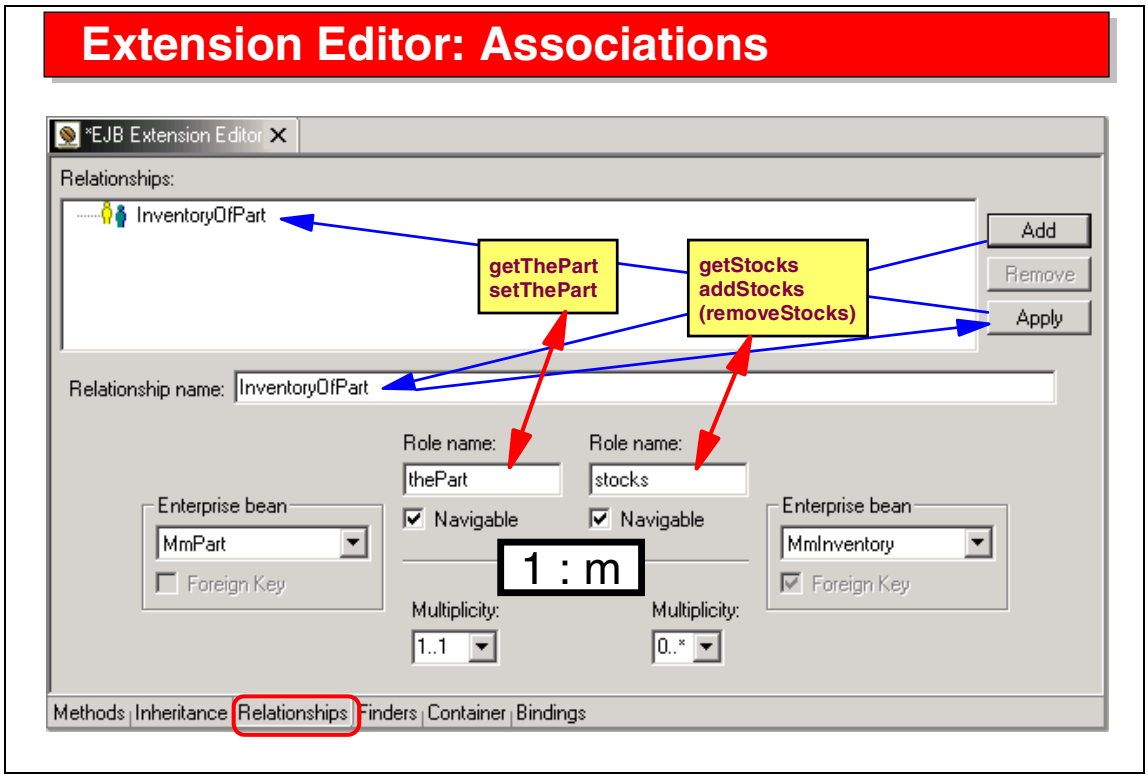
IBM extensions are provided in the Application Developer for these functions.

Inheritance:

- An EJB can inherit attributes and methods from another EJB. Note that the home interface is separate and does not use inheritance.

Associations:

- 1:m and 1:1 associations are supported directly. For m:m, an intermediate EJB with two 1:m associations must be defined.
- Associations are implemented by foreign keys in the underlying tables.



Visual 7-14 Extension Editor: Associations

An association is defined in the Extension editor.

The association is defined between two entity beans, MmPart and MmInventory. The cardinality (1:m) is defined through the two multiplicity values; 1..1 specifies one and only one part (for an inventory item), while 0..m specifies multiple but optional inventory items (for a part).

A role name is defined for each direction; for example, the role of the MmInventory bean as seen from the MmParts bean is *stocks*.

- This specification generates a method named *getStocks* into the part bean.
- Because the association is 1:m, a method named *addStocks* is generated to add an inventory item to the part (*removeStocks* is not generated because 1..1 forces an inventory item to be associated with a part).

The role name of the part is defined as *thePart* (there is only one part for an inventory item):

- The generated methods are *getThePart* and *setThePart*.

IBM Extensions: Access Beans

Easier access to EJBs from client programs

- Lookup of home, create or findByPrimaryKey

Optimized access through caching of attributes

Access bean types

- JavaBean wrapper → deprecated
- Copy helper → deprecated
- Data class → for new applications
  - **Replaces copy helper**
  - **Caches and synchronizes attributes**
- Factory → for new applications
  - **Generated with any other**
  - **Provides access to home**
- ~~Rowset~~
  - **Not supported**

Visual 7-15 IBM Extension: Access Beans

Another IBM extension is access beans.

Access beans make client access to EJBs easier. The access bean contains the code to look up a home by its JNDI name and provide direct access to the remote interface.

Access beans can cache the attributes of entity beans for faster access, and then synchronize (commit) values at the end of business logic methods.

JavaBean wrapper and copy helper access beans were used in VisualAge for Java, but are now deprecated (they can still be used). The copy helper has been replaced by the data class access bean that provides the caching function. Rowset access beans, which were collections of copy helpers, are not supported any more.

Custom Finder Methods

Compatibility with previous

- You can continue to define SQL query strings or method declarations in the finder helper interface

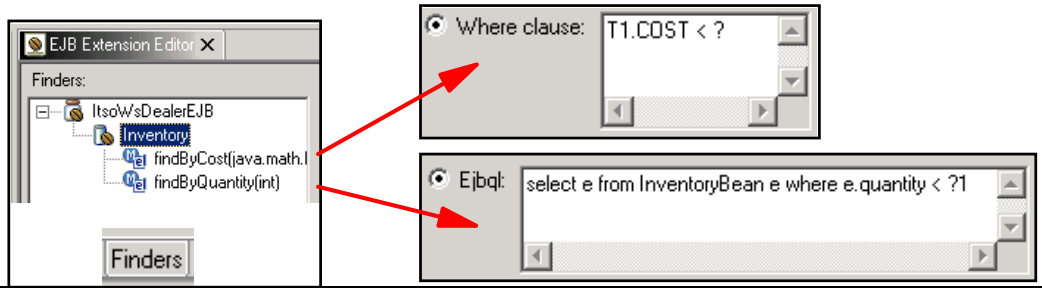
New development

- Define methods in home interface

```
public Enumeration findByQuantity(int quantx)   throws ...;
public Collection  findByCost(BigDecimal costx) throws ...;
```

- Extension editor
  - **EJB query language and SQL language**

ibm-ejb-jar-ext.xmi



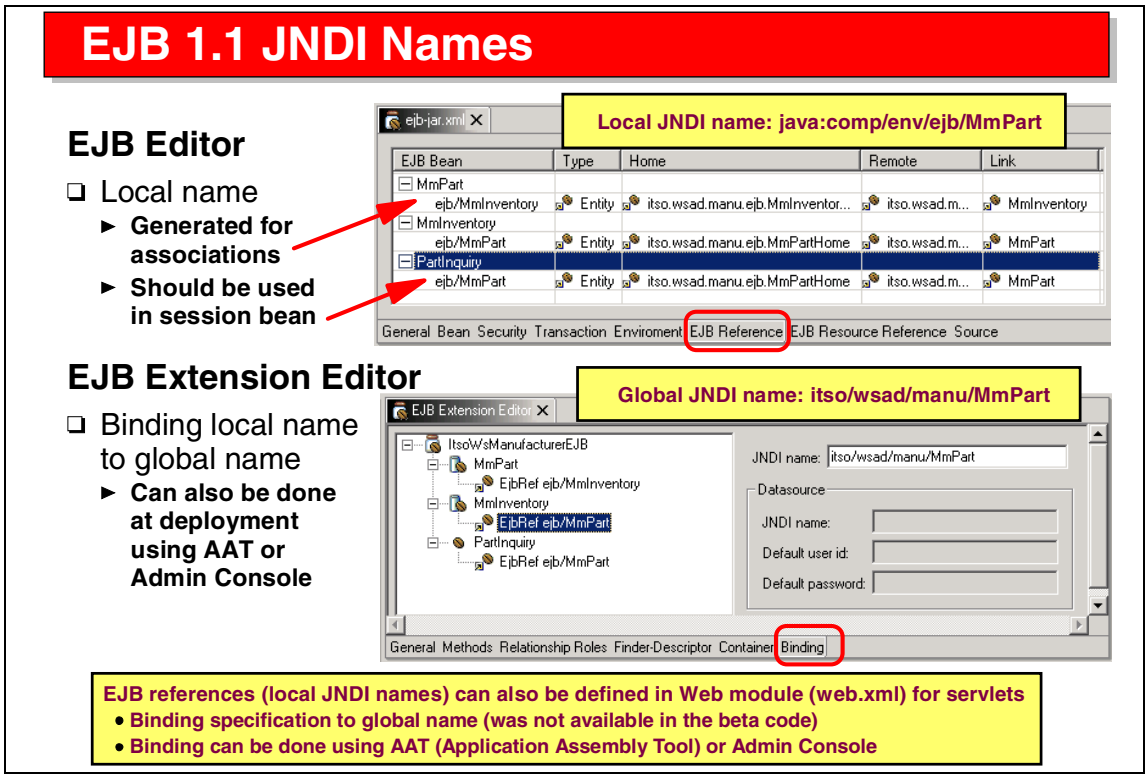
Visual 7-16 Customer Finder Methods

By default, entity EJBs are accessed by their primary key.

For many applications, access by other attributes or partial values is required. This can be implemented through custom finder methods that can return multiple EJBs that qualify the search criteria.

In VisualAge for Java, custom finders were defined in the finder helper interface. This is still supported, but the Application Developer provides better function:

- First define the custom finder methods in the home interface. Note that with EJB 1.1, the return value can be an Enumeration (as before) or a Collection (new in EJB 1.1).
- Use the extension editor to specify the underlying SQL statement where clause that is used to retrieve the matching EJBs. In addition to the SQL language, a new EJB query language (Ejbql) can also be used.

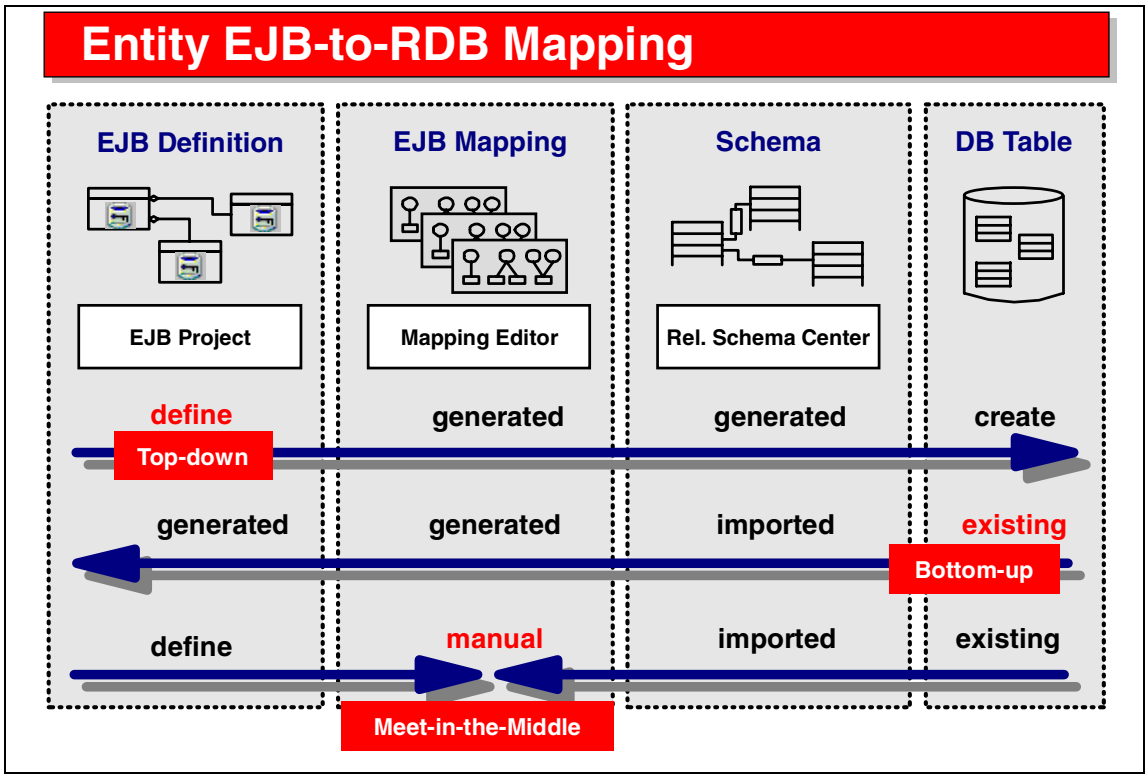


Visual 7-17 EJB 1.1 JNDI Names

In the EJB 1.1 specification, it is suggested that applications use local JNDI names to access EJBs. A local JNDI name is specified as **ejb/Beanname** (in the Java code, the home is found using **java:comp/env/ejb/Beanname**). In an application server, global JNDI names are used and they must be unique.

Local JNDI names are used in the generated code for associations. They should also be used in hand-written code of session beans (that access entity beans) and in Web applications (servlets).

Local names and their mapping to EJBs are defined in the EJB editor. The mapping of local names to global names (called the binding) can be done in the extension editor, or it can be done as a deployment activity in the Application Assembly Tool (AAT) of WebSphere.

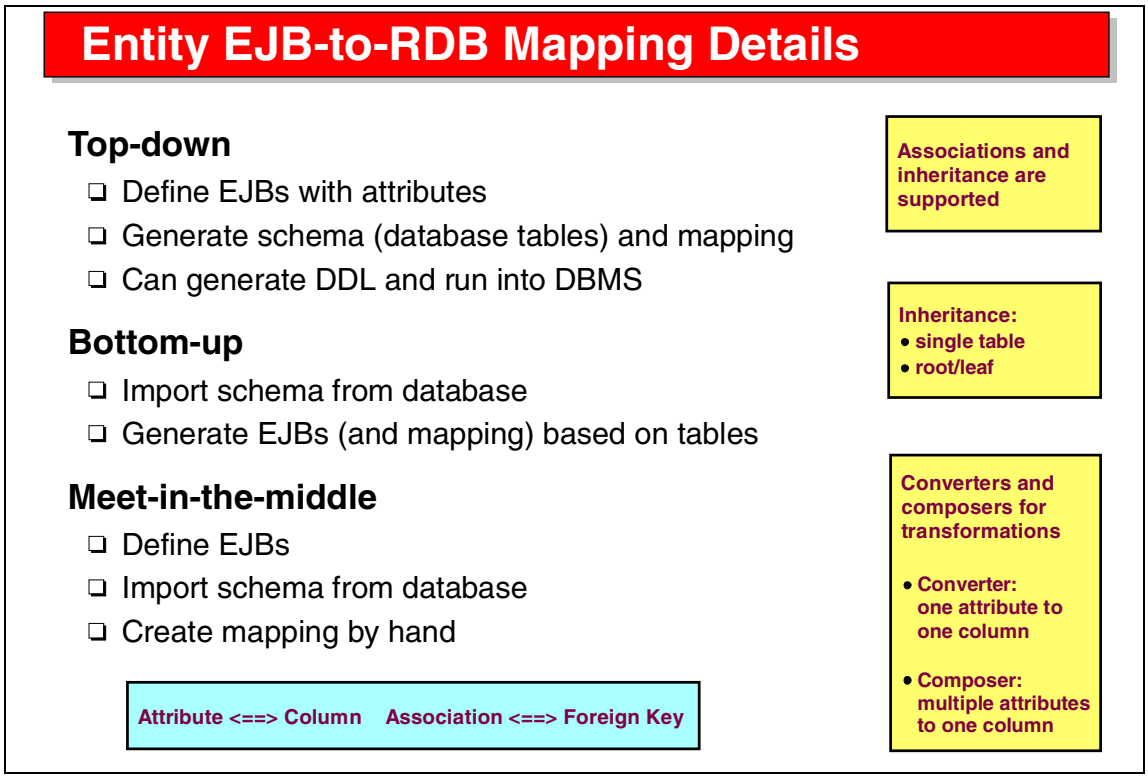


Visual 7-18 Entity EJB-to-RDB Mapping

Entity beans (container-managed) must be mapped to relational tables. This can be done in three ways:

- Top-down—Define the entity bean and have the matching tables generated (one column for each property).
- Bottom-up—Import existing tables using the relational schema center and have matching entity beans generated (one property per column).
- Meet-in-the-middle—Define the entity bean and import an existing table. Perform the mapping by hand (using the mapping editor). The entity bean should correspond to the table from the beginning, but this approach gives more freedom in regard to data type conversions.

The disadvantages of top-down and bottom-up are that the user has little control over the names and data types that are generated.



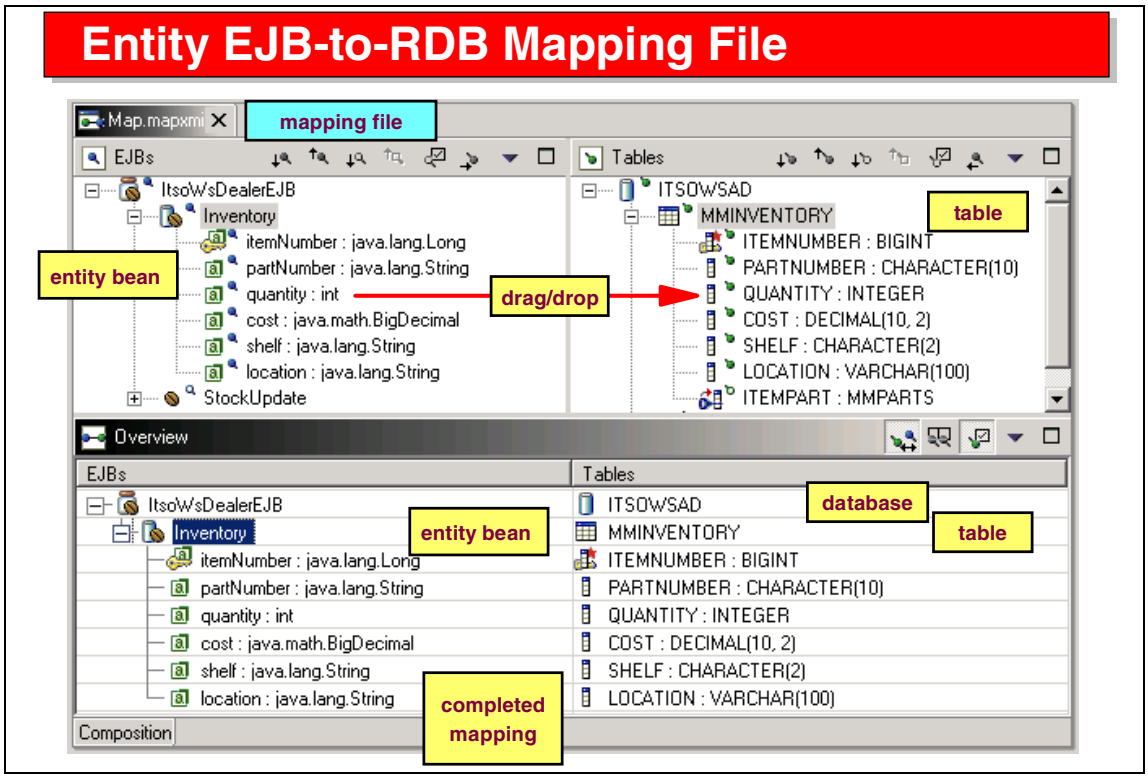
Visual 7-19 Entity EJB-to-RDB Mapping Details

Associations and inheritance are supported by the mapping tool as well.

An inheritance structure can be mapped into a single table or into multiple tables (one per entity bean).

Associations map to foreign keys implemented in the tables.

Special mapping function called converters and composers enable more complex transformations of data types, or to compose entity attribute types from multiple table columns.

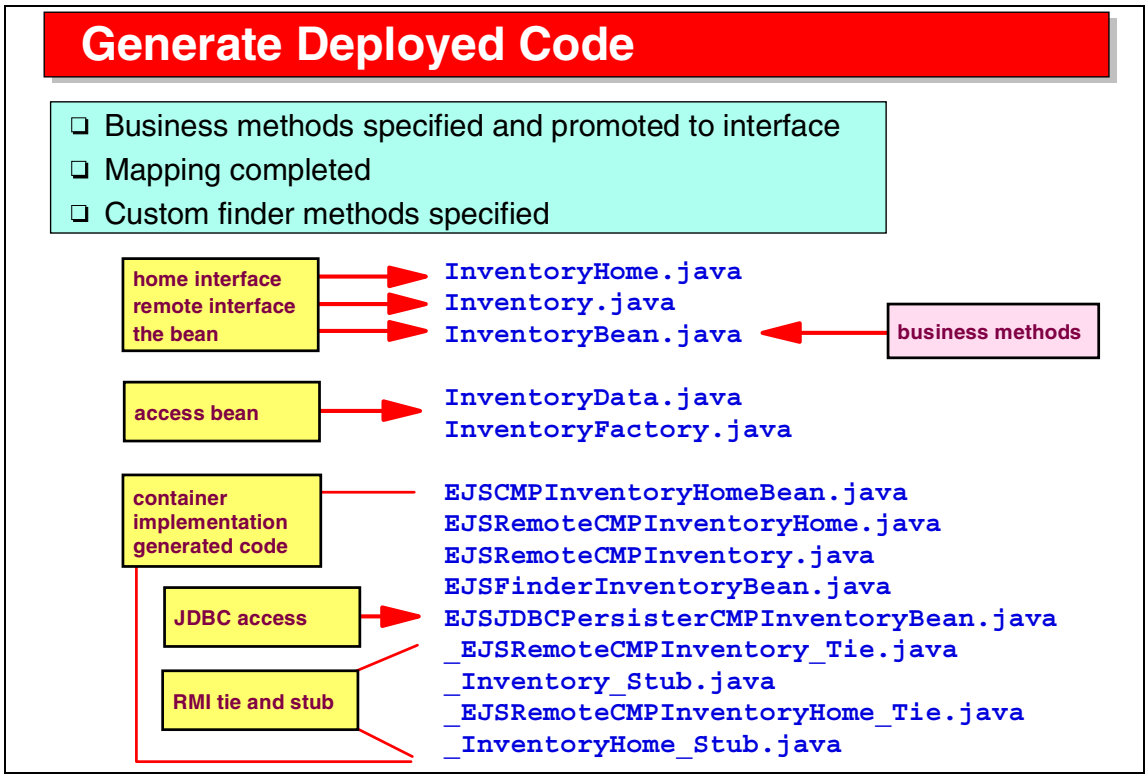


Visual 7-20 Entity EJB-to-RDB Mapping File

The mapping tool provides the support for the mapping of entity beans to relational tables:

- In the top pane, you drag entities to tables, entity attributes to columns, associations to foreign keys (also in the opposite direction).
- The bottom pane shows the mapping that has been completed.

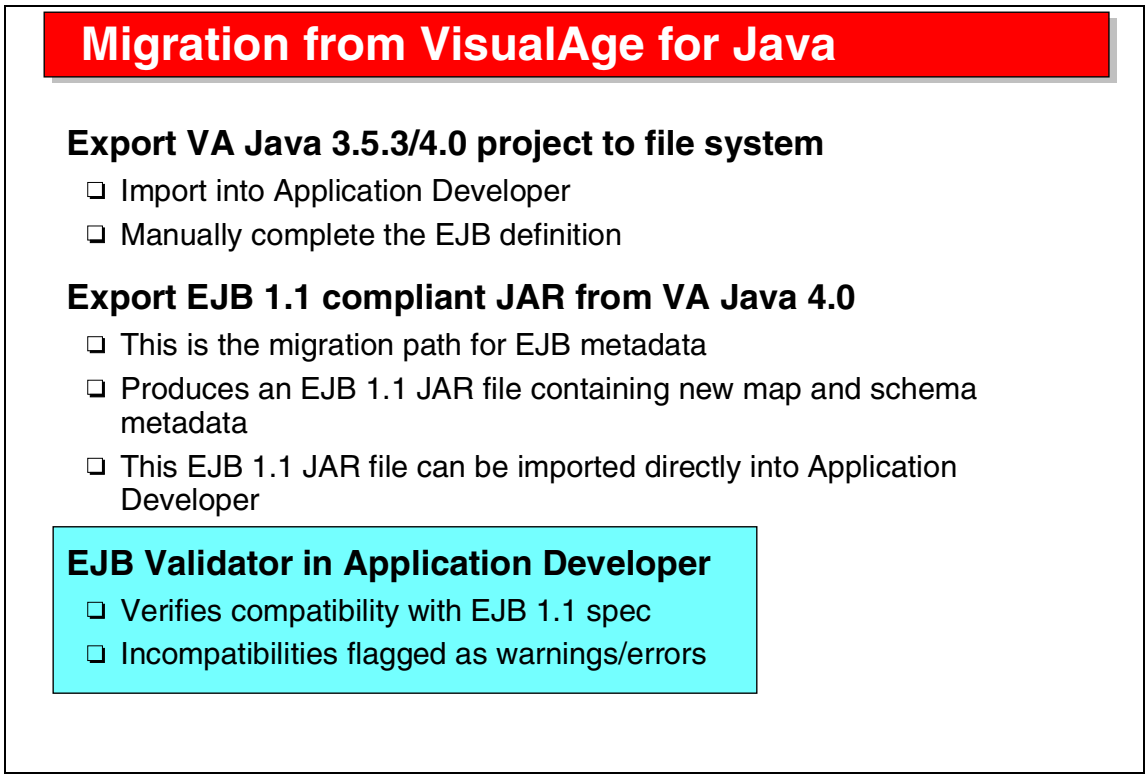
The mapping information is stored in the Map.mapxmi file of the EJB project.



Visual 7-21 Generate Deployed Code

When all the specifications are complete, you generate the deployed code.

One class named EJSJDBCPersisterCMPxxxxx contains all the SQL statements used to create, retrieve, update, and delete the entity bean.

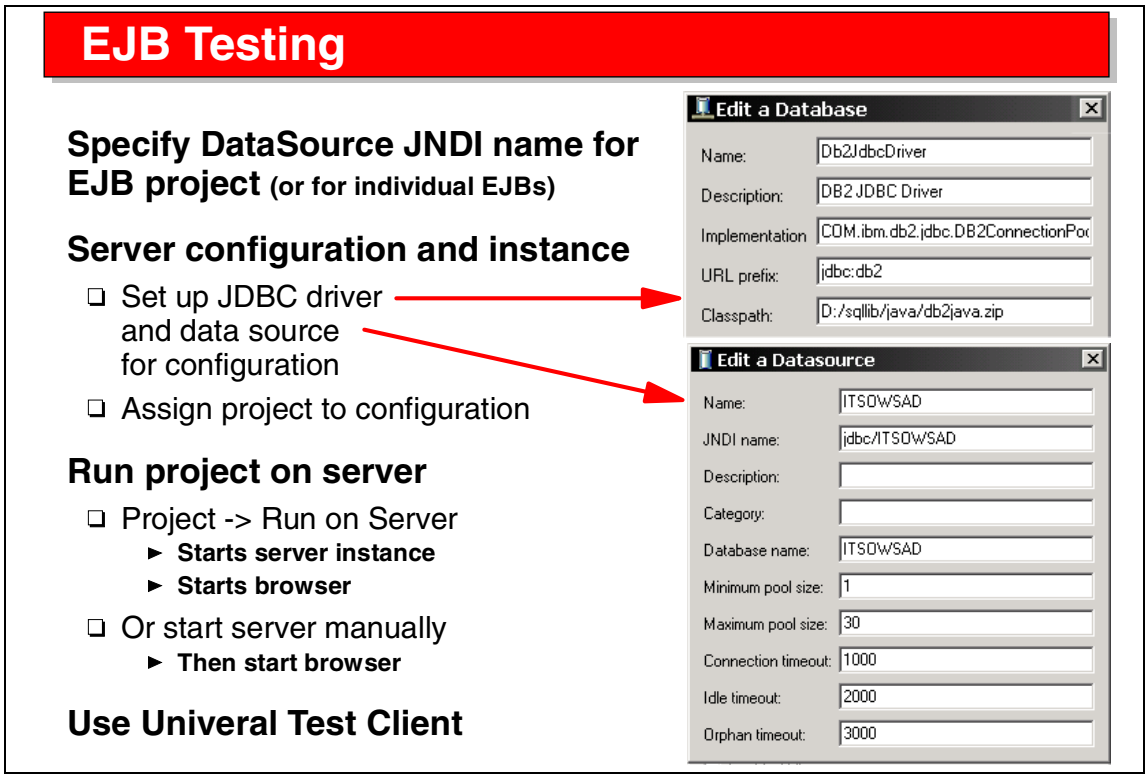


Visual 7-22 Migration from VisualAge for Java

Migration of EJB definitions from VisualAge for Java is easy when an EJB 1.1 compliant JAR file can be generated. This is only supported in VisualAge for Java Version 4. Such a JAR file contains all the deployment information including the mapping to the tables.

From earlier versions of VisualAge for Java you can export the Java source files into an EJB project, but manual effort is required to complete the definitions and to redo the mapping to tables.

The Application Developer provides an EJB validator that checks if the EJB 1.1 specifications have been followed.



Visual 7-23 EJB Testing

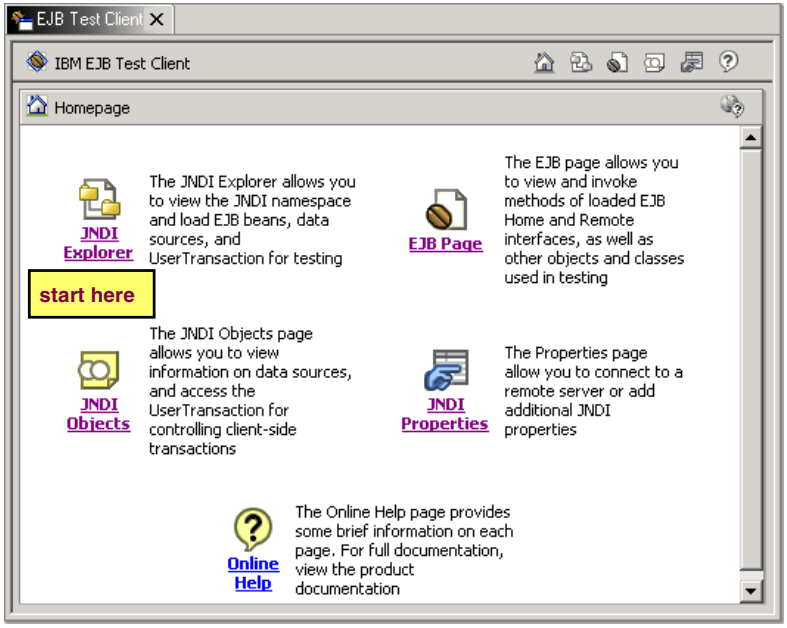
EJBs can be tested right in the Application Developer in the embedded WebSphere AEs server. The steps are:

- Define the data source JNDI name in the EJB editor.
- Set up a WebSphere AEs server with JDBC driver and data source information that matches the JNDI name and access the correct database.
- Assign the project to the server.
- Start the server (manually or by selecting *Run on Server* for the project).
- Start a browser with the EJB test client.

Universal Test Client

New test client

- Browser-based
- Similar function as VA Java EJB Test Client
- Can test EJBs and Web Services

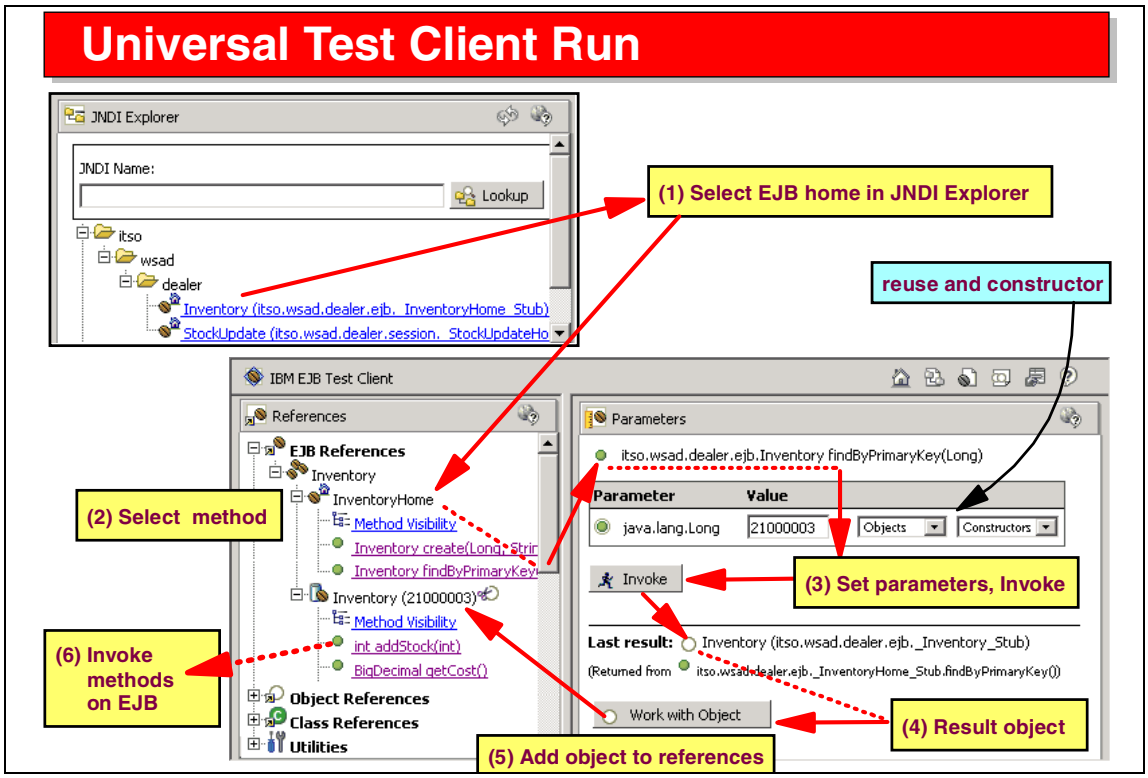


Visual 7-24 Universal Test Client

The universal test client (UTC) can be used to test EJBs very effectively.

This test client provides functions similar to the EJB test client of VisualAge for Java, but it is browser-based and also provides functions to test Web Services.

The starting point to test EJBs is the JNDI Explorer, where we can find the homes of the EJBs that run in the EJB container in the server.



Visual 7-25 Universal Test Client Run

A typical test run progresses as follows:

- Select an EJB in the JNDI Explorer.
- The home interface is displayed in the References pane.
- Select a method of the home interface (create or findByPrimaryKey).
- Enter parameter values in the right pane. For simple types, you can just enter the value, while for objects you can reuse an existing object or invoke a constructor.
- Click *Invoke* to run the method. The result EJB is shown in the bottom pane. Click *Work with Object* to add the remote object to the References pane.
- Expand the remote object and select a method to be tested.
- Enter parameter values (as above) and *Invoke* the method. Result values or objects are displayed and can be added as references (EJB references or object references).
- This cycle can be repeated with any method selected in the References pane.

Universal Test Client Functionality

Can use built-in browser or external browser

JNDI explorer

- Search name space, find any EJB

Dynamic method invocation

- Creation and passing of complex objects as parameters

Object clipboard

- Save returned objects
  - **Constructors when building parameters**
    - **BigDecimal values for example**
- Reuse in other method calls
- Stored under Object References

Universal Test Client can be installed in stand-alone WebSphere Application Server

Class loader

- Load a JavaBean, instantiate, run methods (Web Services)

Visual 7-26 Universal Test Client Functionality

The universal test client can be run in the internal browser or in an external browser.

The object clipboard can be used to save any result object and reuse it as parameter for further method calls.

The test client is a Web application and is available as an EAR project that can be installed into a stand-alone WebSphere Application Server for testing.

Summary

EJB project and J2EE Perspective provide

- EJB development environment
- J2EE-conforming deployment
- EJB Editor and EJB Extension Editor
- Wizard for creation
- Three mapping approaches

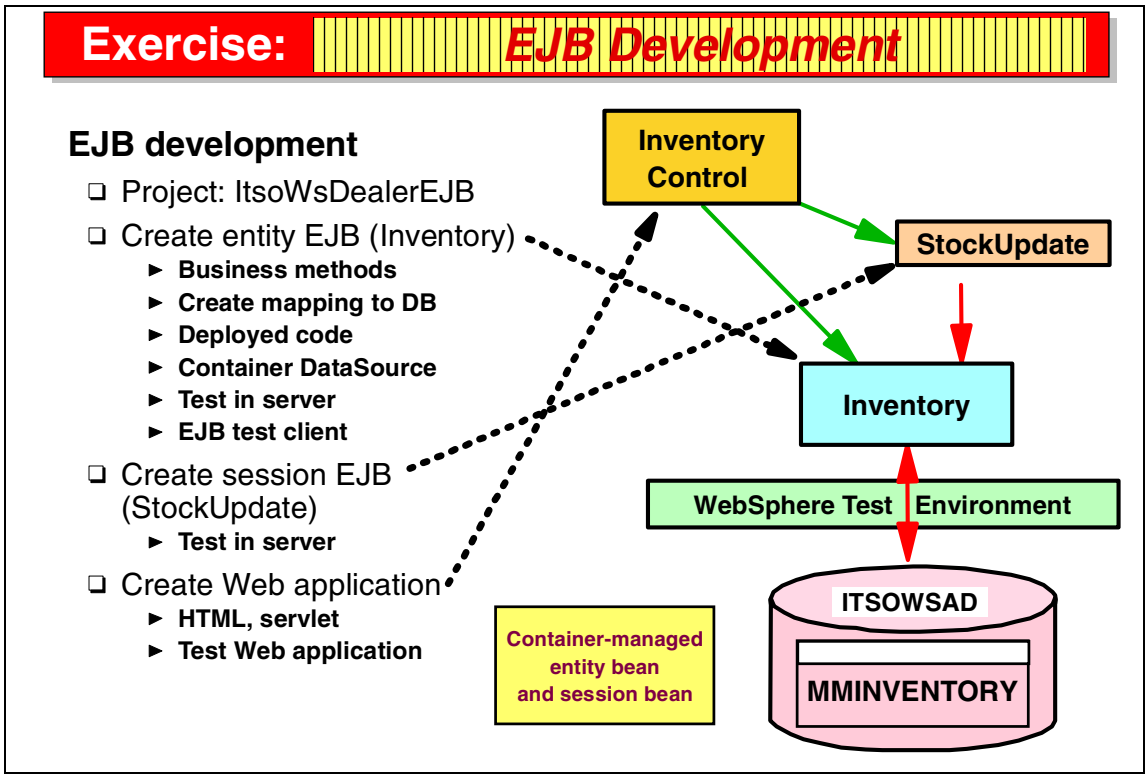
Server project and Perspective provide

- Full EJB test environment
- JNDI lookup
- Universal Test Client

Visual 7-27 Summary

The EJB tooling support of the Application Developer provides full support for J2EE application development, including support for the EJB 1.1 specification.

The universal test client provides support for dynamic testing of EJBs (as well as for Web Services).



Visual 7-28 Exercise: EJB Development

The EJB development exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with two EJBs:

- Inventory—an entity bean that maps to the inventory table in the ITSOWSAD database
- StockUpdate—a session bean with business methods to manipulate stock values in the entity EJB

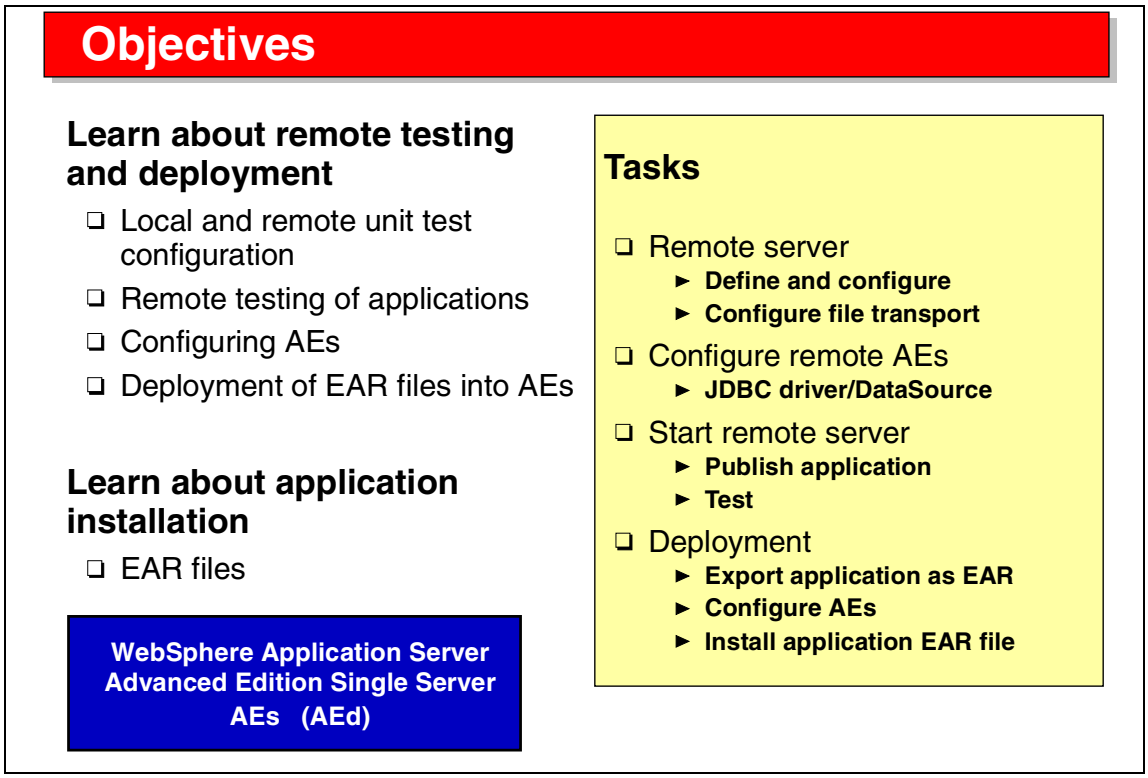
You also create a Web application that uses the EJBs to update the database.

See Exercise 5, “EJB development” on page 329 for the instructions for this exercise.

8

Application Developer: Deployment to WebSphere

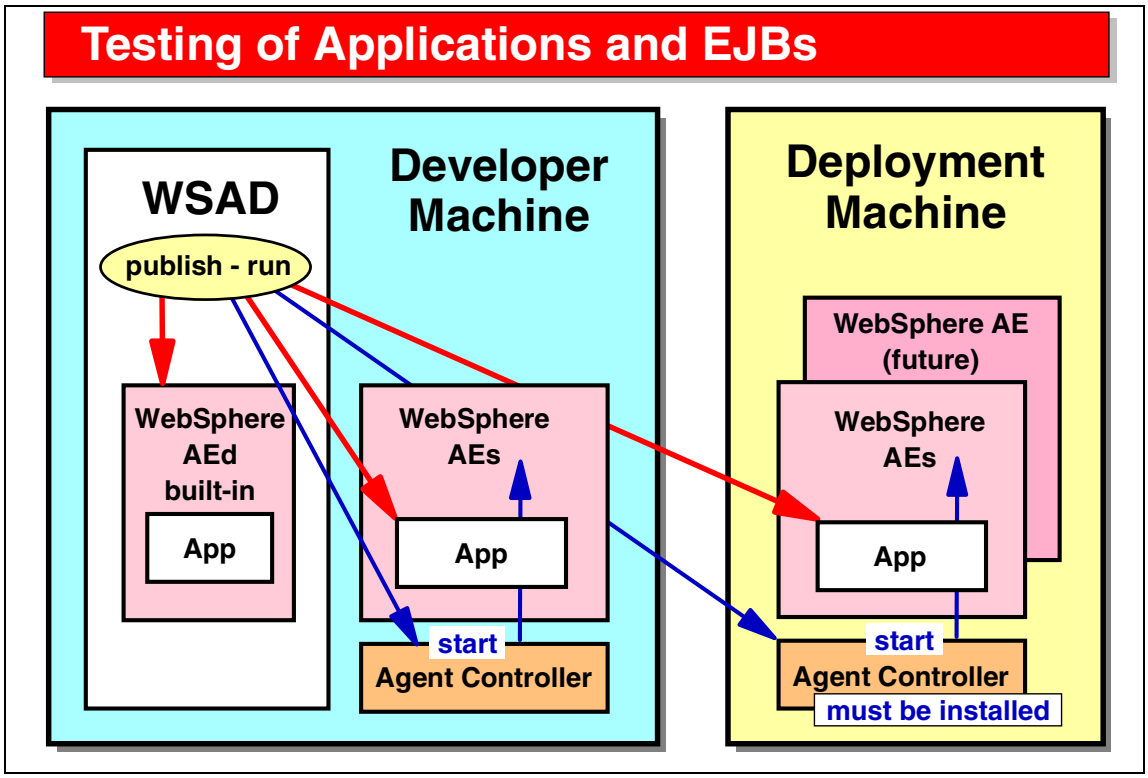
Figure 8-1 Title



Visual 8-2 Objectives

The objectives of this unit are to:

- Understand local and remote testing of Web applications and EJBs
- Understand how to set up a remote test configuration
- Understand deployment of Web applications and EJBs to WebSphere Application Server AEs and AE
- Understand installation of an application archive (EAR) files



Visual 8-3 Testing of Applications and EJBs

The Application Developer provides a local and remote test environment for testing of Web applications:

- WebSphere Application Server AEd (developer edition, same code as AEs, but free) is built into the Application Developer.
- WebSphere Application Server AEs (single server edition) can be installed on the same or on a remote machine. A remote server is started through the IBM Agent Controller, which must be installed on the machine where the server runs.

For testing, a Web application is published to the selected server by installing the owning EAR project file into the application server. Then the server is started and the Web application can be tested in a Web browser.

Publishing and Testing

Built-in server

- Define server configuration/instance with JDBC driver/data source
- Assign project to server configuration
- Start server and browser or *Run on Server*

WebSphere AEs on same or other machine

- Define server configuration/instance with JDBC driver/data source
- Remote server instance: **WebSphere v4.0 Remote Test Environment**
  - **hostname, AE installation and deployment directory**
- Remote file transfer instance
  - **Copy files or FTP files**
  - **Remote target directory**
- Assign project to server configuration
- Start server and browser or *Run on Server*
  - **IBM Agent Controller must be installed on remote server to start remote AEs**
    - **serviceconfig.xml must point to AEs home directory**

EAR file
- **Web modules**
- **EJB modules**

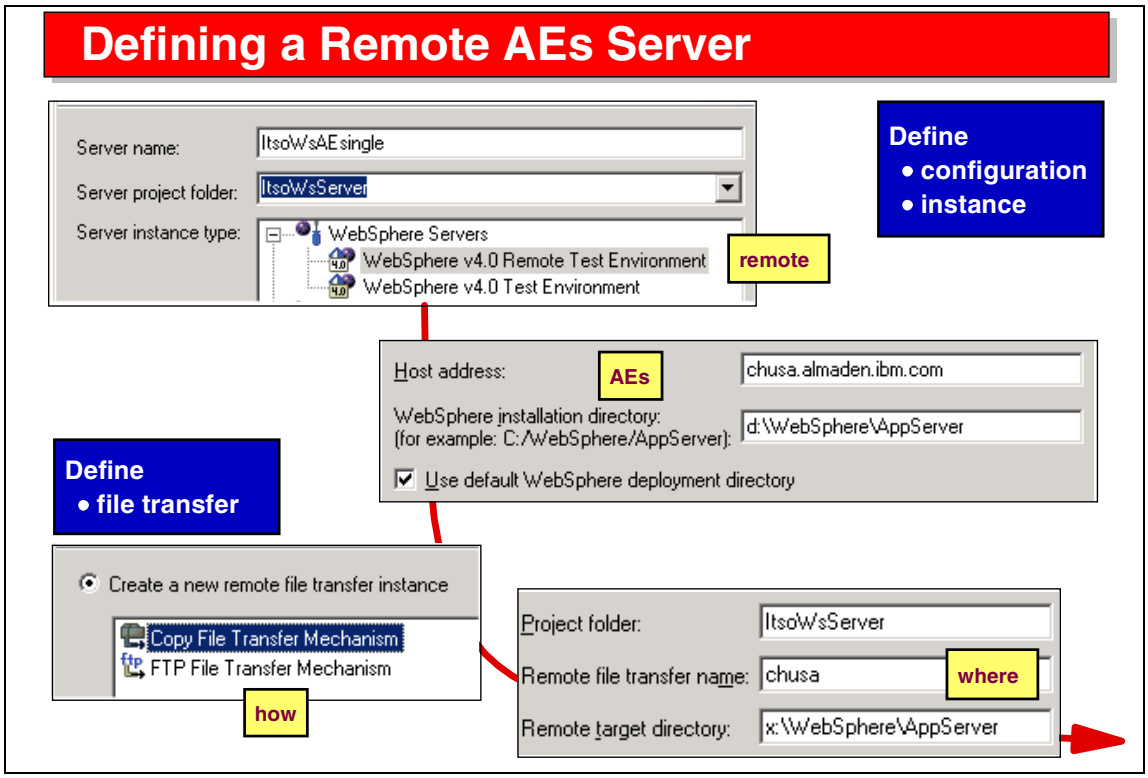
Visual 8-4 Publishing and Testing

The built-in server (WebSphere AEd) has to be configured with JDBC drivers and data sources, and the EAR project has to be assigned to the server.

The EAR file is installed into the test server before the server is started; this is called *publishing the code to the server*.

If WebSphere AEs is installed separately from the Application Developer, on the same or another machine, we can test applications remotely. In a remote test environment, the application code has to be transferred to the application server. This can be done by file copy (LAN drive) or by FTP.

The remote server is started from the Application Developer through the IBM Agent Controller that must be installed on the machine where AEs is installed.

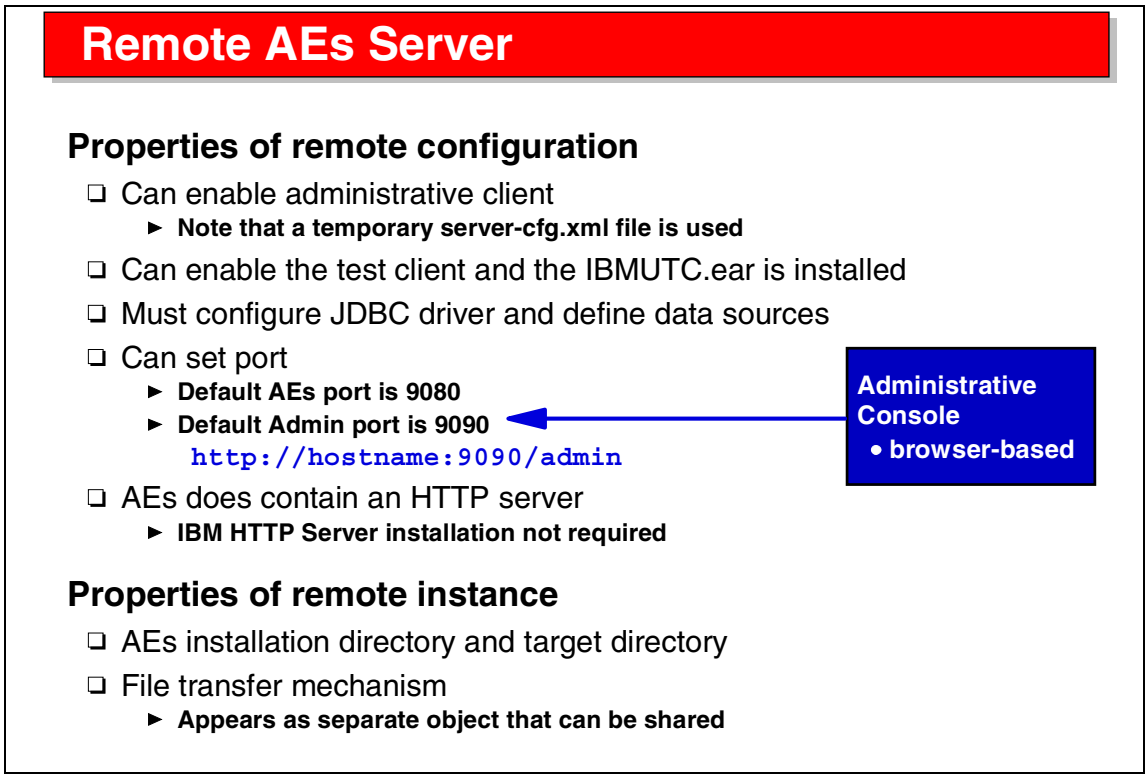


Visual 8-5 Defining a Remote AEs Server

To define a remote AEs server, you have to specify:

- The host name or address
- The WebSphere installation directory
- The mechanism of file transfer (copy of FTP)
- The remote target directory

The remote file transfer mechanism is stored as an object (XMI file) in the server project.



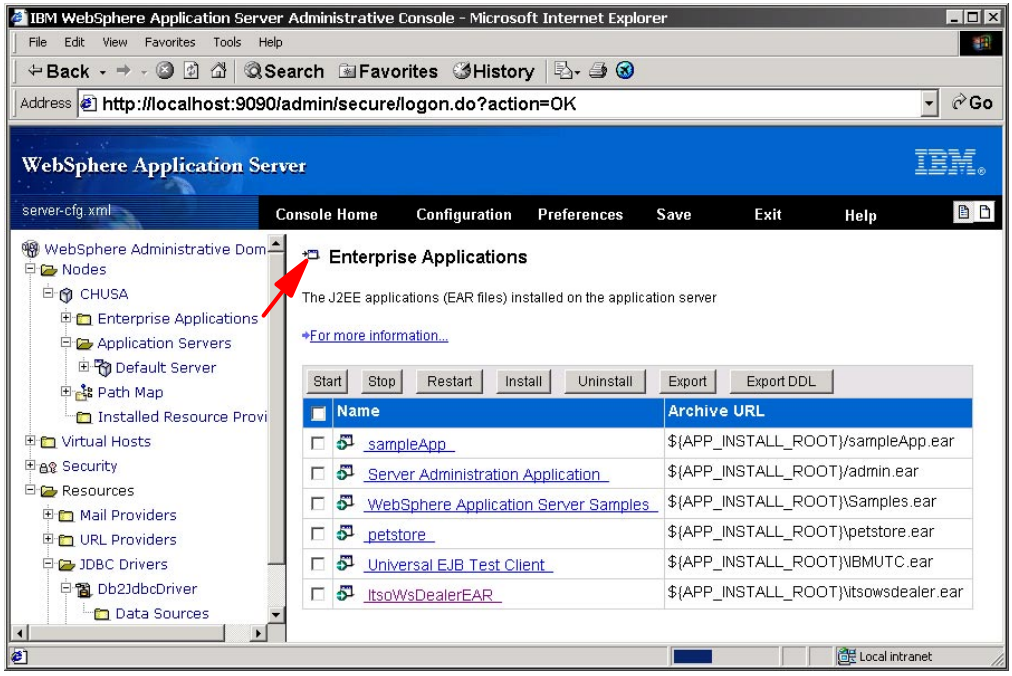
Visual 8-6 Remote AEs Server

For a remote AEs server, you can:

- Activate the administrative client of AEs so you can do configuration tasks right from the Application Developer
- Enable the universal test client so that EJBs can be tested
- Configure JDBC drivers and data sources
- Set up port numbers for the different servers

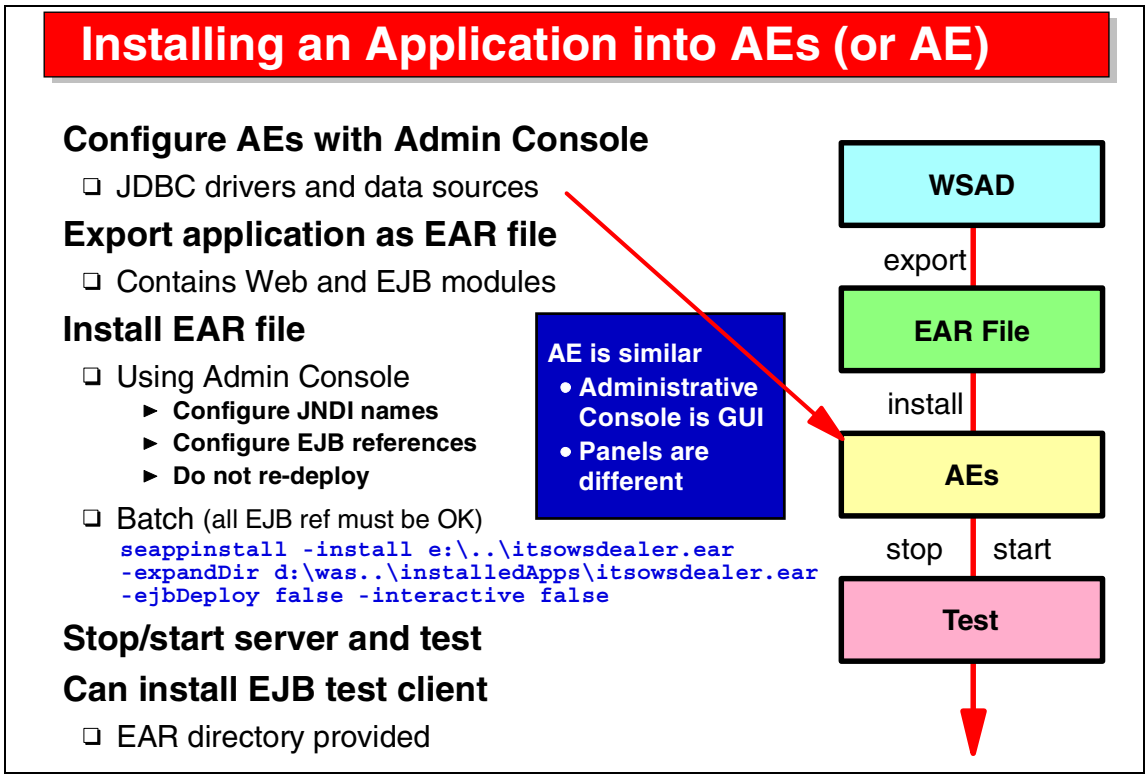
Note that AEs has a built-in HTTP server, so you do not require a separate HTTP server (such as the IBM HTTP server).

Administrative Console of AEs



Visual 8-7 Administrative Console of AEs

The administrative console of AEs shows the installed applications, such as the universal test client and the application that you want to test.



Visual 8-8 Installing an Application into AEs or AE

To deploy applications into a real WebSphere AEs or AE server, you have to:

- Configure JDBC drivers and data sources used in the real environment.
- Export the EAR file of the containing EAR project.
- Install the EAR file in the application server. This can be performed in two ways:
  - Using the administrative console. This allows to perform additional configuration tasks, such as JNDI names and EJB references.
  - Using a batch command (seappinstall). In this case you should have done all configuration work in the Application Developer.
- Stop and start the server to enable the application.

The universal test client can also be installed manually on any WebSphere application server by copying the UTC EAR directory and installing it. This enables testing of EJBs from a browser in the real environment.

Deployment Activities

Deployment activities

- Configure JNDI names
- Bind EJB references (local names) to global JNDI names

  `Can be done in WSAD`
  - **EJB-to-EJB references, servlet-to-EJB references**
- Deploy EJBs (generate code)

  `Can be done in WSAD`

Application Assembly Tool (AAT)

- Can perform all deployment activities
  - **Install EAR file after that is easy**

Batch installation (SEAppInstall)

- All mapping should have been done before
  - **Fully deployed EAR file**
- Interactive mode possible

Visual 8-9 Deployment Activities

Deployment activities in WebSphere Application Server include:

- Configure global JNDI names
- Bind the local JNDI names to global JNDI names (EJB references, such as associations and session to entity references, and servlet to EJB references)
- Regenerate the deployed code

These activities can be performed in the administrative console or by using the Application Assembly Tool (AAT).

The fastest way to deploy is by performing all deployment specifications in the Application Developer and by using the batch command to deploy the EAR file.

Summary

Server perspective provides

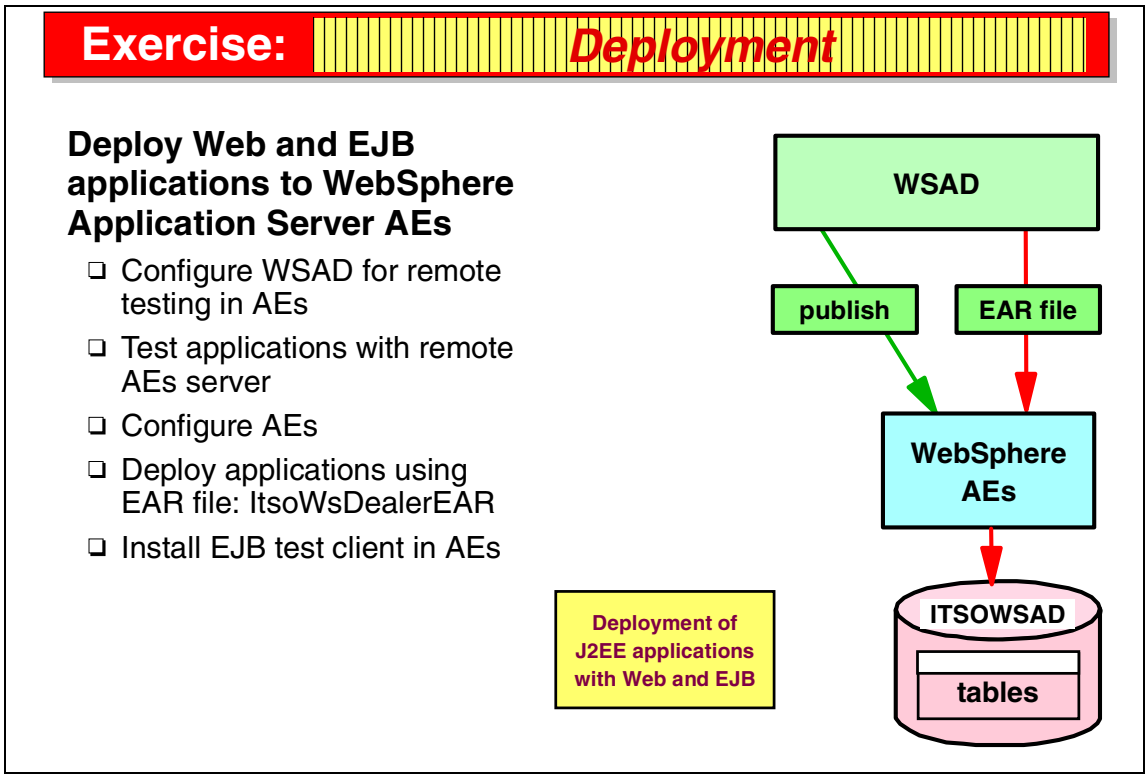
- Remote server for unit testing
- Remote server and file transfer configuration
- Remote start/stop of server through the IBM Agent Controller

EAR file

- Deployable application
  - **Contains Web and EJB modules**
- Can be installed in WebSphere Application Server AE and AEs

Visual 8-10 Summary

The support for WebSphere Application Server in the Application Developer is outstanding. This enables easy testing inside the Application Developer, outside in an AEs server, and deployment to AEs or AE full function.



Visual 8-11 Exercise: Deployment

The deployment exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with the EAR project that contains the Web and EJB projects developed in the previous exercises:

- Set up a server for remote testing and test the applications
- Configure AEs for deployment
- Export the EAR file and install it in the application server
- Optionally, install the universal test client

See Exercise 6, “Test and deploy using WebSphere AEs” on page 339 for the instructions for this exercise.

9

Application Developer: Profiling Tools

Figure 9-1 Title

Objectives

Learn about performance analysis tools included with Application Developer

- ❑ Architecture
- ❑ Class and method path length
- ❑ Object leaks
- ❑ Performance bottlenecks

Learn about the different views provided for performance analysis

Tasks

- ❑ Install IBM Agent Controller on remote systems
- ❑ Configure WebSphere Test Environment
  - ► **Enable agent**
  - ► **Disable JIT compiler**
- ❑ Start trace
- ❑ Run application
- ❑ Use viewers to analyze trace data

Visual 9-2 Objectives

The objectives of this unit are to:

- ► Understand the profiling tools of the Application Developer
- ► Understand how to set up performance measurement
- ► Understand the reports that are produced

Overview

WSAD Performance Analyzer can gather application information on

- Applications running in WebSphere Application Server
- Stand-alone applications
- Same or remote machine from Application Developer

Performance analysis early in the development cycle

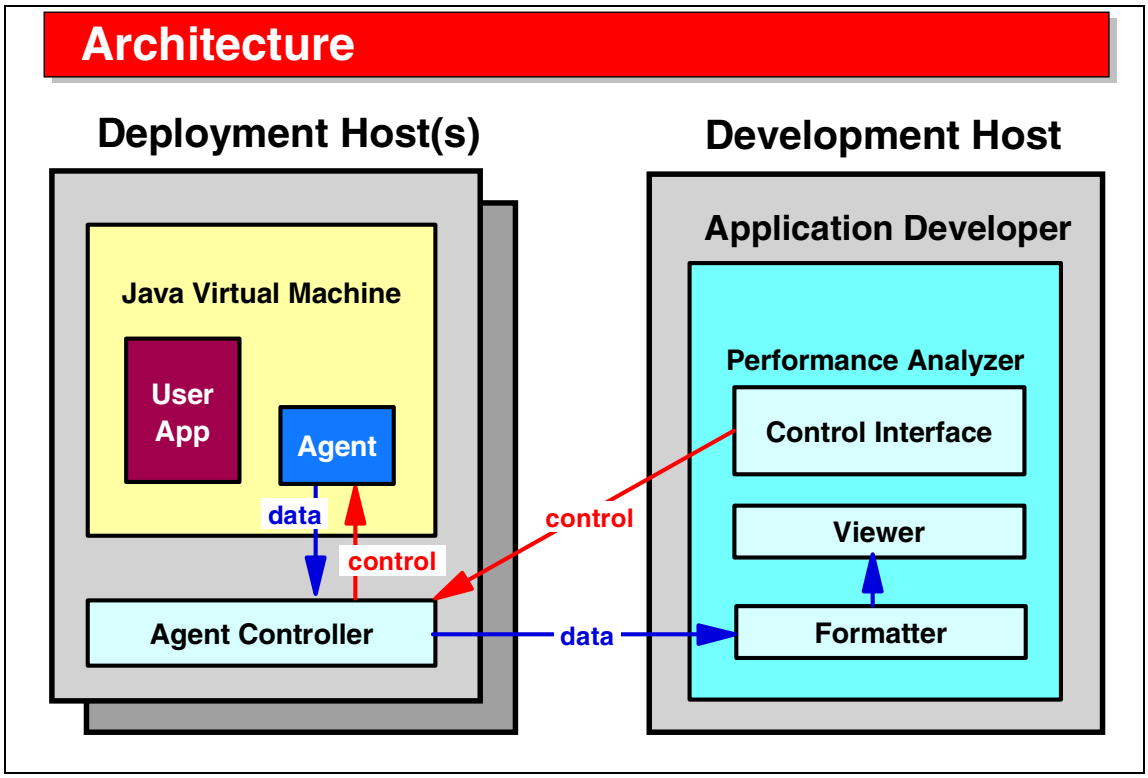
- Within WSAD WebSphere Test Environment
- When testing on WebSphere AEs and AE
- Early detection leads to architectural changes before it is too late
  - **Reduced risk**

IBM Agent Controller to route trace information from execution JVS to WSAD for analysis

Visual 9-3 Overview

Performance analysis should be done early in the project cycle to identify bottlenecks and correct them before going into production with Web applications.

The profiling tools can measure performance inside the Application Developer or outside, for example in WebSphere Application Servers.



Visual 9-4 Architecture

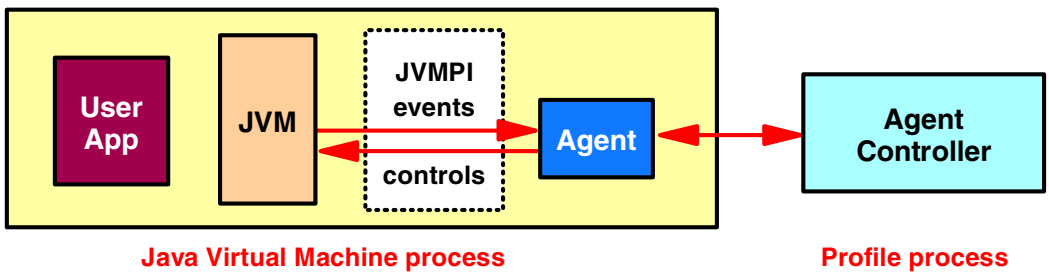
The architecture of the profiling tools involves:

- The Java Virtual Machine (JVM) where the application is running
- An agent inside the JVM that captures the events (entering methods, memory management)
- The IBM Agent Controller that is used between the Application Developer and the remote machine to control the agent and to retrieve the performance measurement data
- The performance analyzer inside the Application Developer that controls the agent and that invokes the formatters and viewers

Remote Agent Controller

Stand-alone daemon or service on the deployment host

- Platforms: Windows, z/OS, 400, AIX, Solaris, HP
- Must be installed (provided with Application Developer)
- Can attach to running agent or launch a new process
- Agents are based on Java Virtual Machine Profiler Interface (JVMPI)
  - **Profile agent receives notification of events (heap alloc, thread start,...)**
  - **Agent can request additional information from JVM**
- WebSphere Test Environment can enable agent (-XrunPIAgent flag)
  - **Set for a server instance**



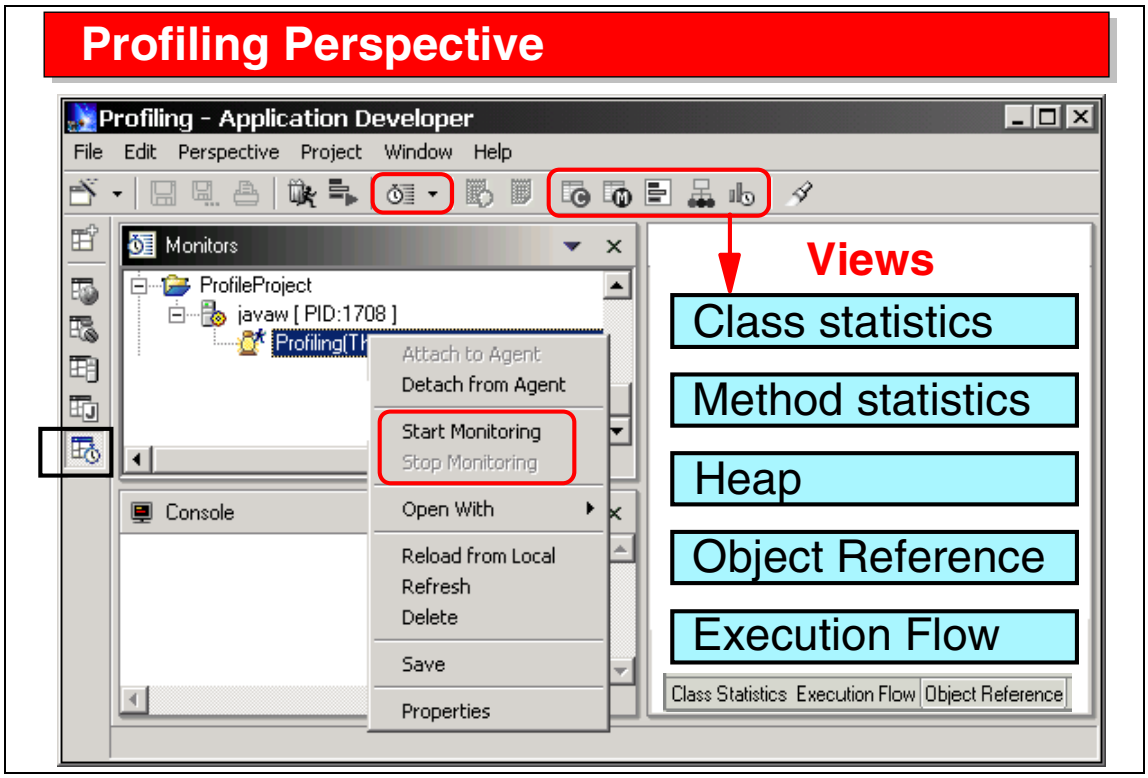
Visual 9-5 Remote Agent Controller

The IBM Agent Controller runs on many platforms, where it must be installed.

The agents that run in the JVM and capture events use the official Java Virtual Machine Profiler Interface (JVMPI) to gather the performance data.

WebSphere AEs (and AE) provide facilities to enable the profiling agent inside their JVM.

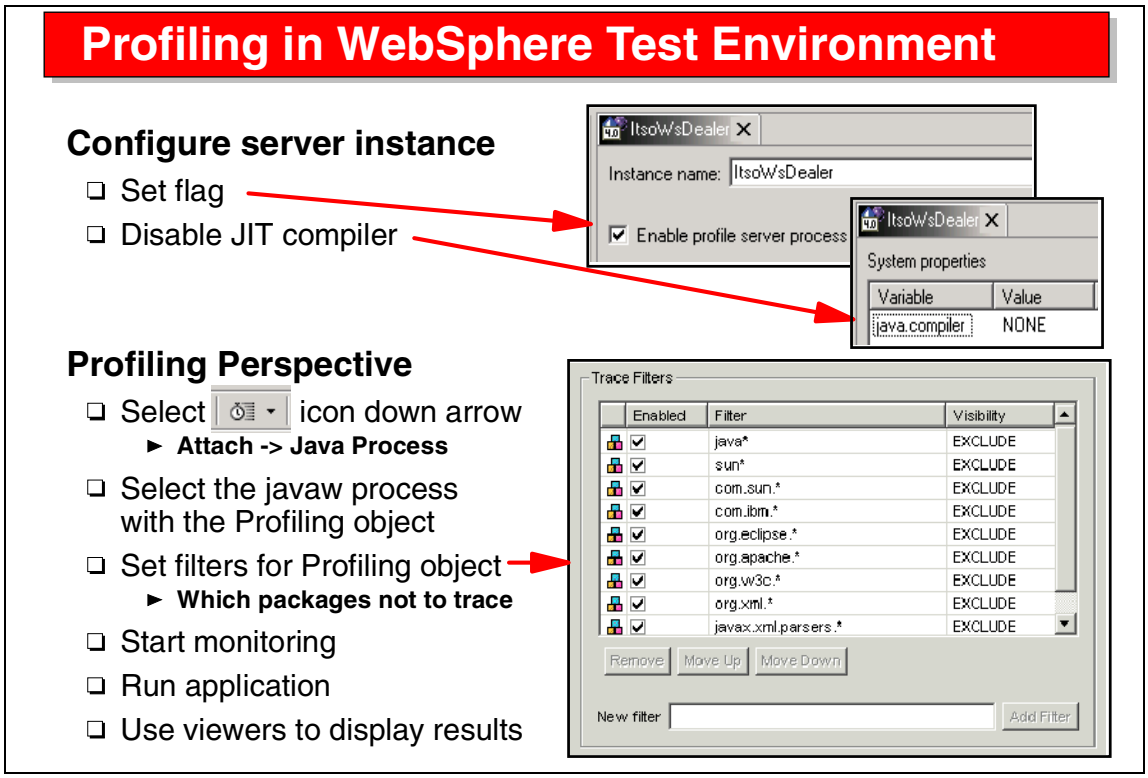
- WebSphere AE:
  - Open the Admin Console
  - Stop the server to be used for profiling, for example, the Default Server
  - On the right-hand side, select *JVM Settings*
  - Click *Advanced JVM Settings*
  - Add this line as the command line arguments:
    `-XrunPIAgent:server=enabled`
  - Click *Apply*
  - Start the server



Visual 9-6 Profiling Perspective

The Profiling Perspective is used to activate an agent by launching an application or by attaching to a running process.

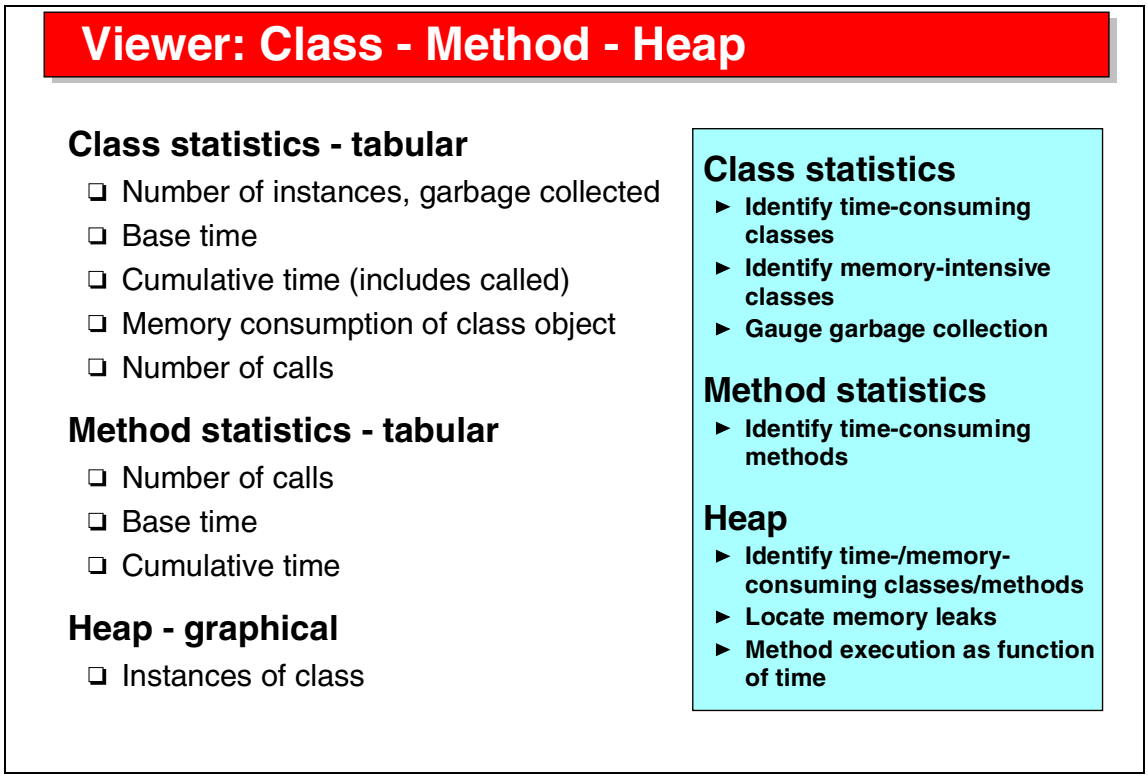
The control interface is then used to start and stop monitoring, retrieve performance measurement data, and display that data in a number of textual and graphical views.



Visual 9-7 Profiling in WebSphere Test Environment

The tasks involved in profiling an application inside WebSphere AEs are:

- Configure the WebSphere AEs server for profiling. For example, the just-in-time compiler (JIT) must be disabled so that all code runs through the JVM.
- Attach to the WebSphere AEs server process.
- Set filters (that is, which classes should not be trace).
- Start monitoring.
- Run the Web application.
- Retrieve and performance data and display results in the viewers.
- Stop monitoring.



Visual 9-8 Viewers: Class - Method - Heap

A number of viewers are provided to display the performance data:

- Class statistics—the time spent in each class
- Method statistics—the time spent in each method of each class
- Heap—class instances
- Object references—all objects with their references to other objects
- Execution flow—a graphical view of the execution through the methods of the involved classes

Viewers: Objects - Execution Flow

Object reference - graphical

- Base set of objects
- References between instances
- Node repetition
- Old/new objects against specific time

Execution flow - graphical

- Objects and time scale
- Time when method is called
- Time spent executing a method
- Time when method returns

Object reference

- **Determine cause of memory leak**
- **Determine why object is not garbage-collected**

Execution flow

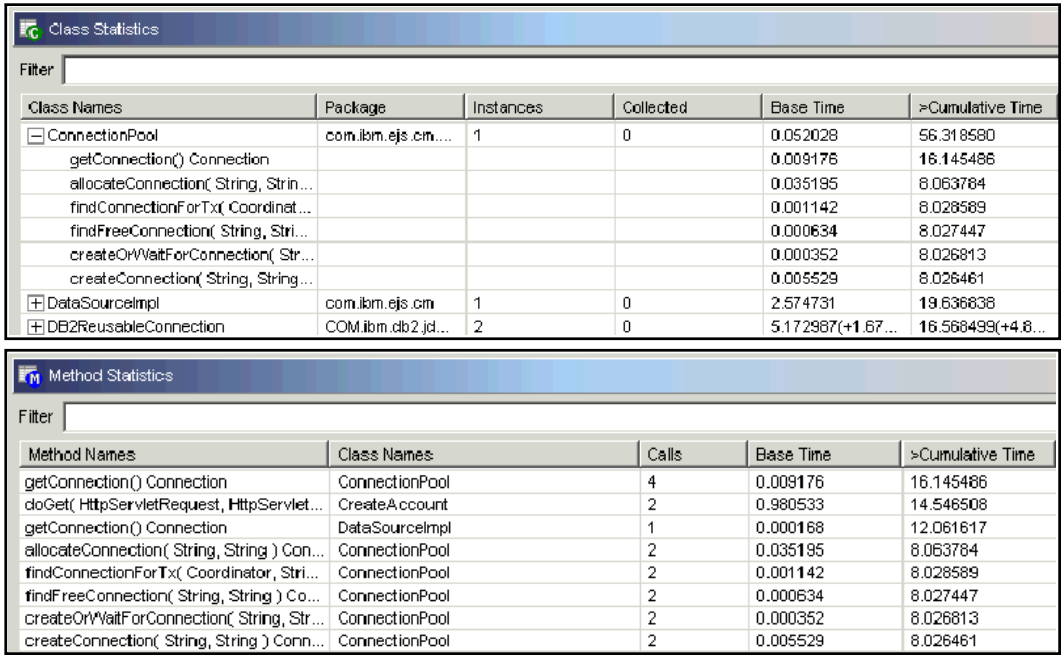
- **Identify which threads are active when**
- **Identify long-lived or frequently called methods**
- **Gauge the amount of garbage collection**
- **Identify phases of program execution**

Visual 9-9 Viewers: Objects - Execution Flow

Performance analysis identifies:

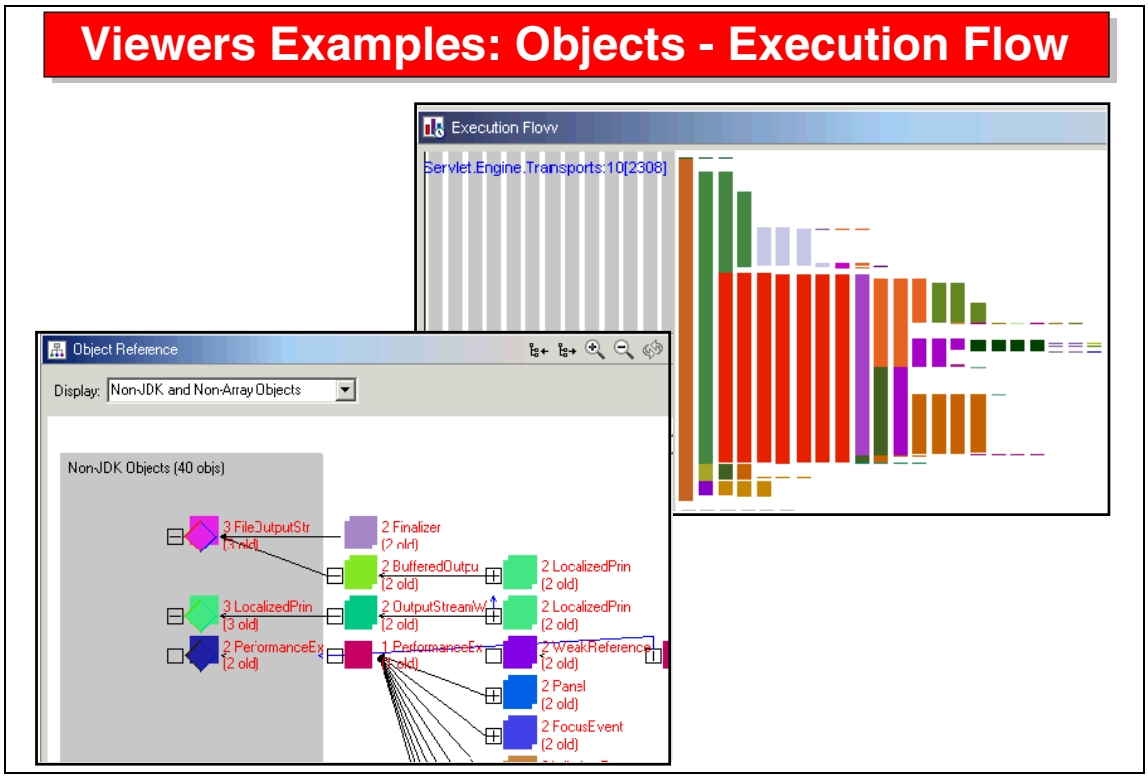
- Time-consuming classes and methods
- Garbage collection and memory leaks (references not freed for garbage collection)
- Long-lived and frequently called methods

Viewers Example: Class - Method



Visual 9-10 Viewers Examples: Class - Method

These views show the class and method statistics of a Web application.



Visual 9-11 Viewers Examples: Objects - Execution Flow

This views shows the object references and execution flow of an application.

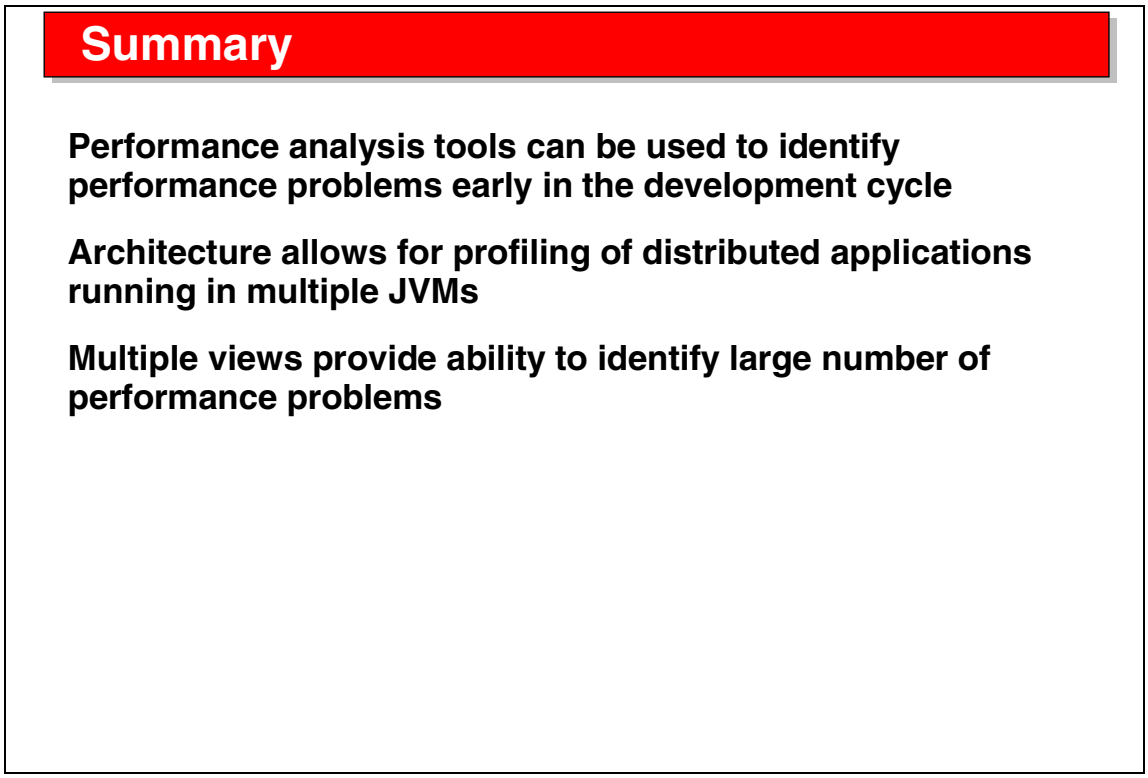
Hints and Tips

- IBM Agent Controller service must be started
- External JVM requires agent controller BIN directory in PATH
- Disable the JIT compiler

  `-Djava.compiler=NONE`
  - **System property in WSAD WebSphere Test Environment**
  - **Disable JIT check box in WebSphere Application Server Version 4**
- Limitation in JDK:
  - **Cannot use profiling and debugging at the same time**
- Communication to/from RAC uses TCP/IP socket 10002 (beta)
  - **Configurable at GA**
  - **Firewall may block**
- Currently no authentication for RAC
- Choose good filters
- Refresh views to ensure consistent and current data
- Execution flow view has no correlation when using multiple processes

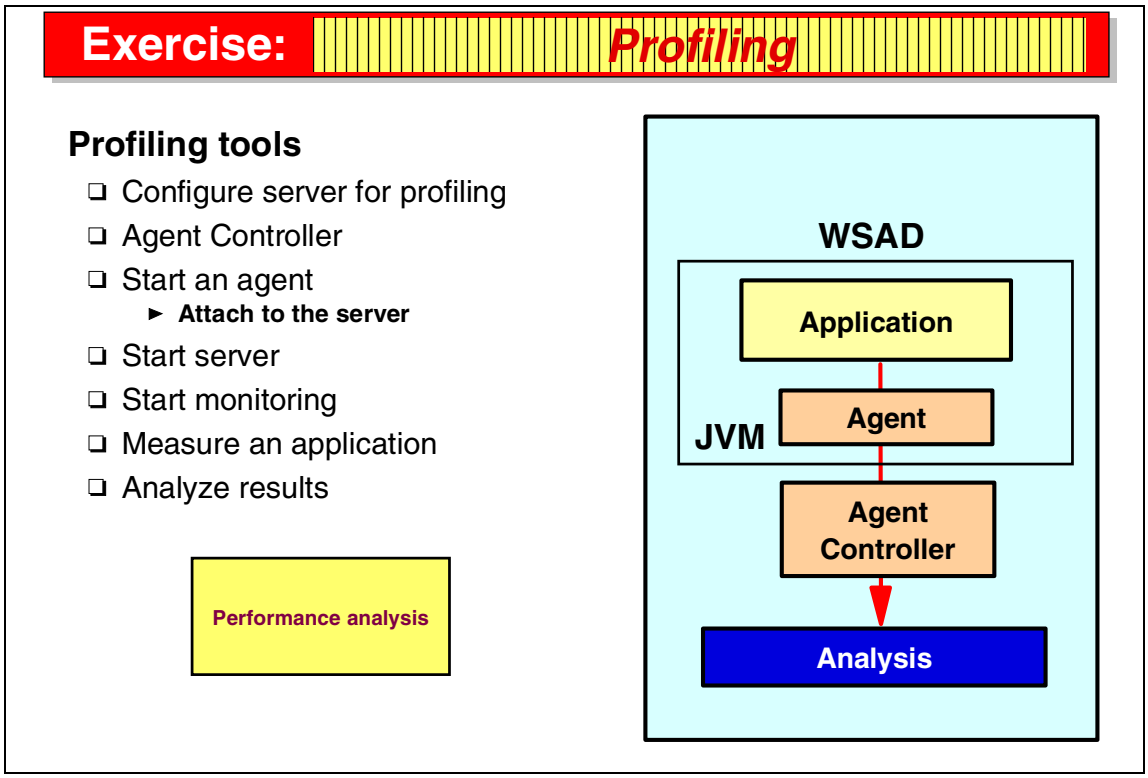
Visual 9-12 Hints and Tips

This visual lists a number of hints and tips to make performance measurement effective.



Visual 9-13 Summary

The profiling tools provide sufficient function to analyze application performance early in the life cycle, during development time, instead of measuring deployed applications in a test or production environment.



Visual 9-14 Exercise: Profiling

The profiling exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with a Web application developed in a previous exercise:

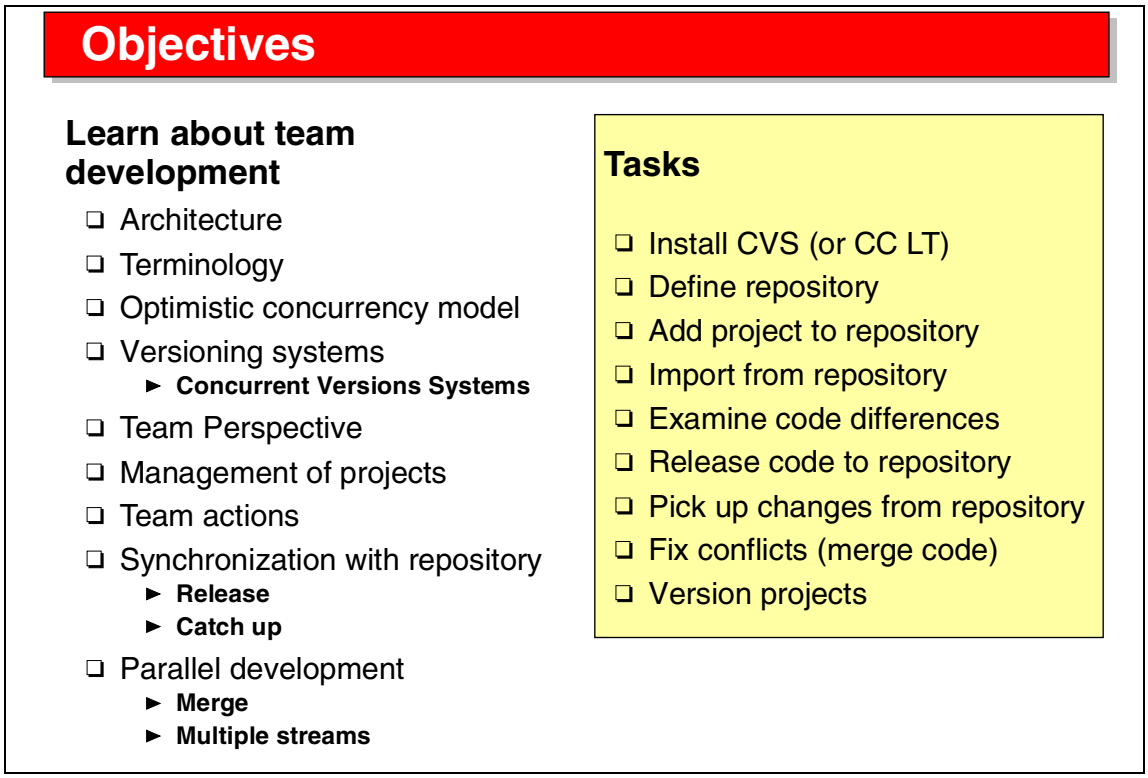
- Configure a server for profiling
- Attach an agent to the server
- Start the server and enable monitoring
- Collect results and display in viewers

See Exercise 7, “Profiling an application” on page 345 for the instructions for this exercise.

10

Application Developer: Team Development

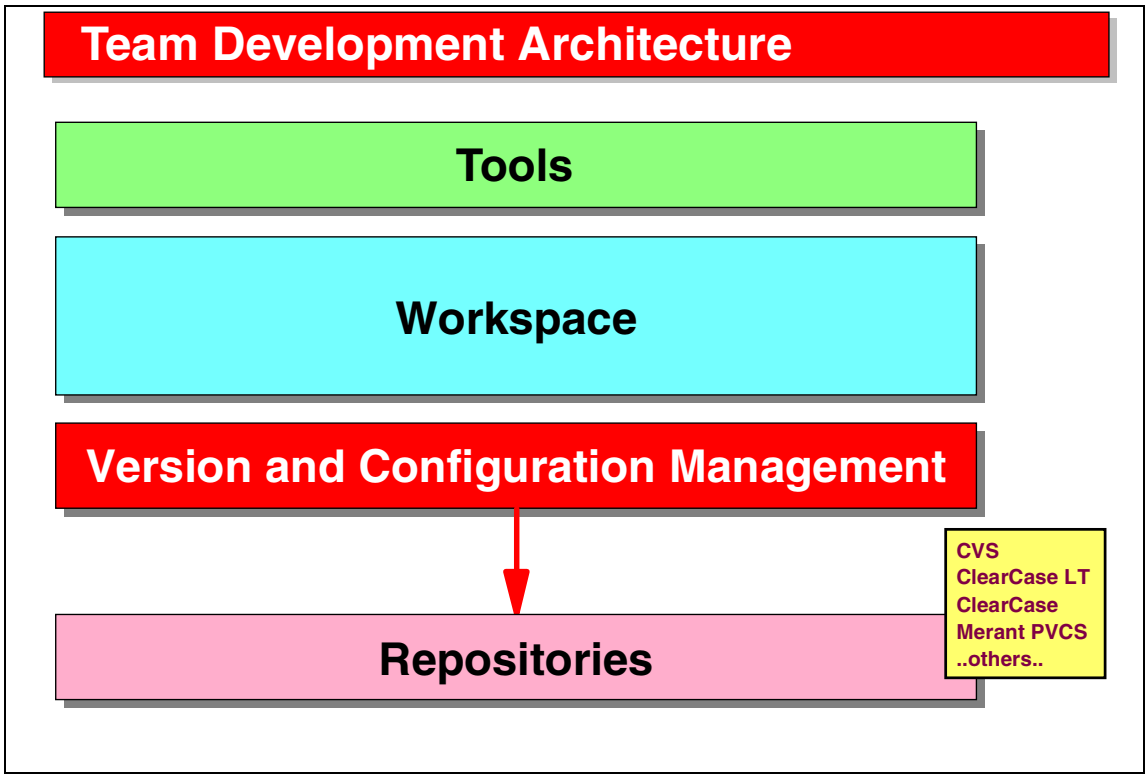
Visual 10-1 Title



Visual 10-2 Objectives

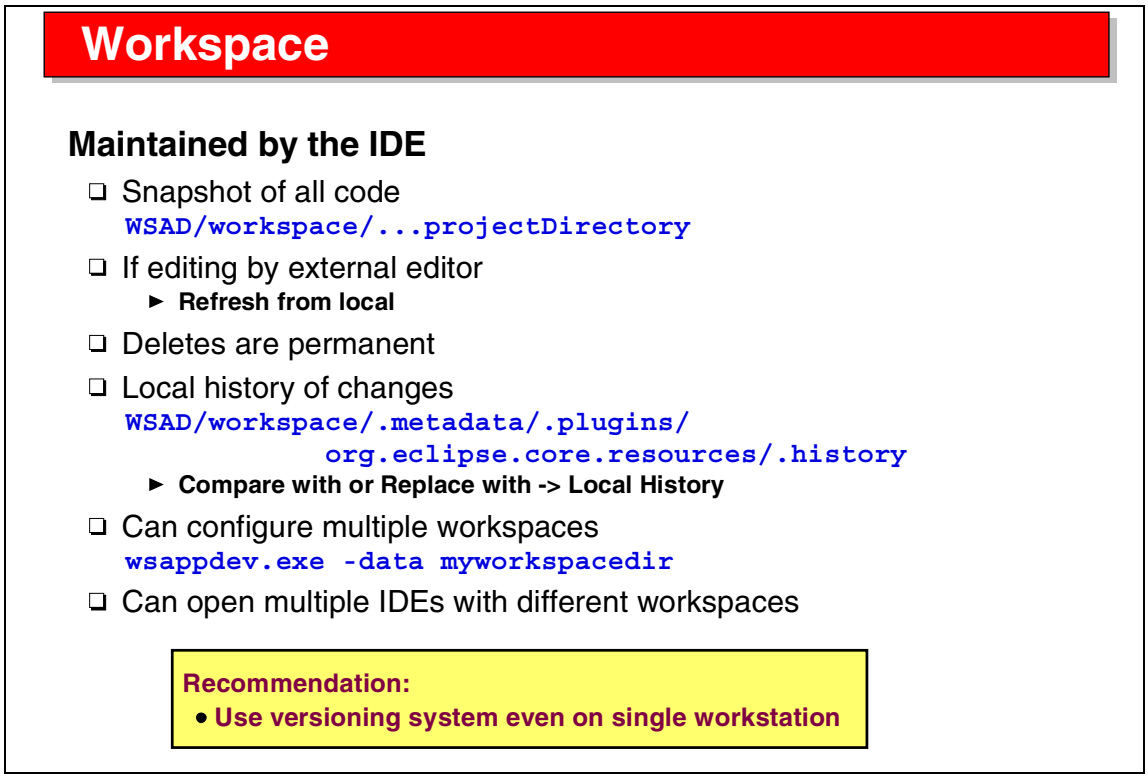
The objectives of this unit are to:

- Understand the team development environment provided by the Application Developer
- Understand the optimistic concurrency model
- Understand the versioning systems supported, Concurrent Versions System (CVS) and ClearCase Light (CC LT)
- Understand the team member actions



Visual 10-3 Team Development Architecture

The version and configuration management architecture enables vendor tools to enable to the Workbench platform.



Visual 10-4 Workspace

The Application Developer maintains a workspace where the project data is stored. By default it is the directory:

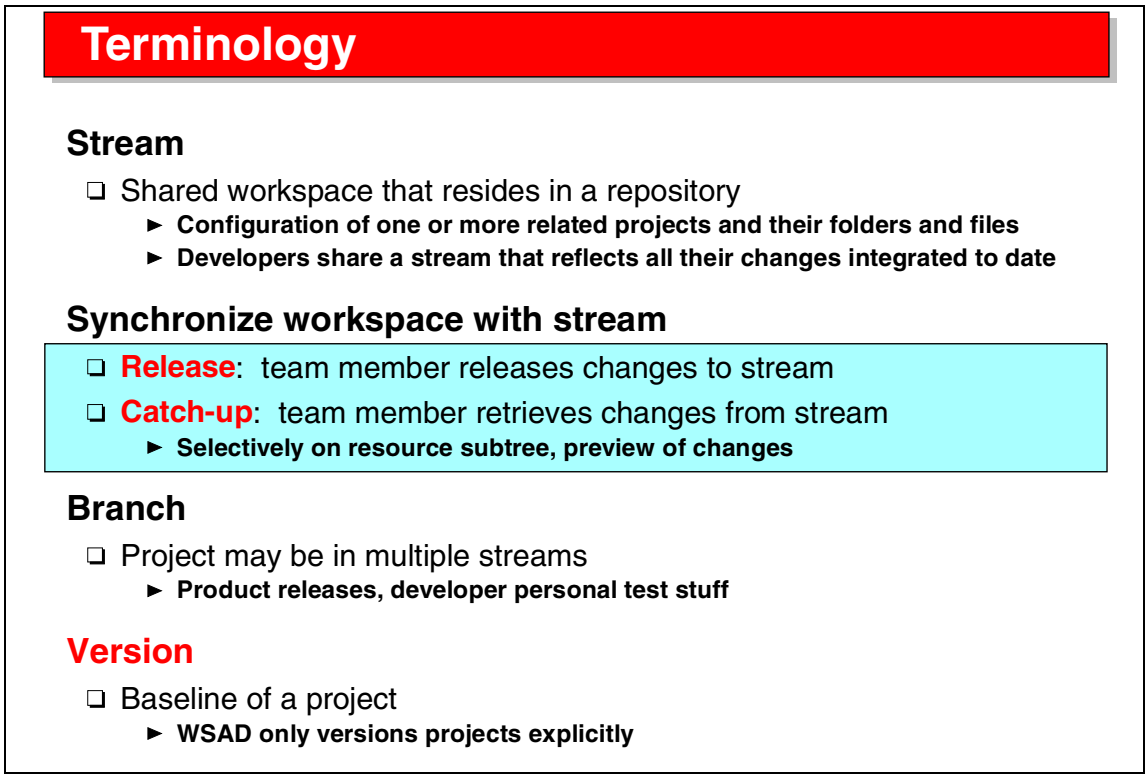
```
d:\<WSADROOT>\workspace
```

The workspace directory can be specified when the Application Developer is started (-data workspacedirectory).

The interactive development environment has these facilities:

- Deletes are permanent (no recycle bin)
- A history of all changes is maintained locally, and files can be reset to a previous state

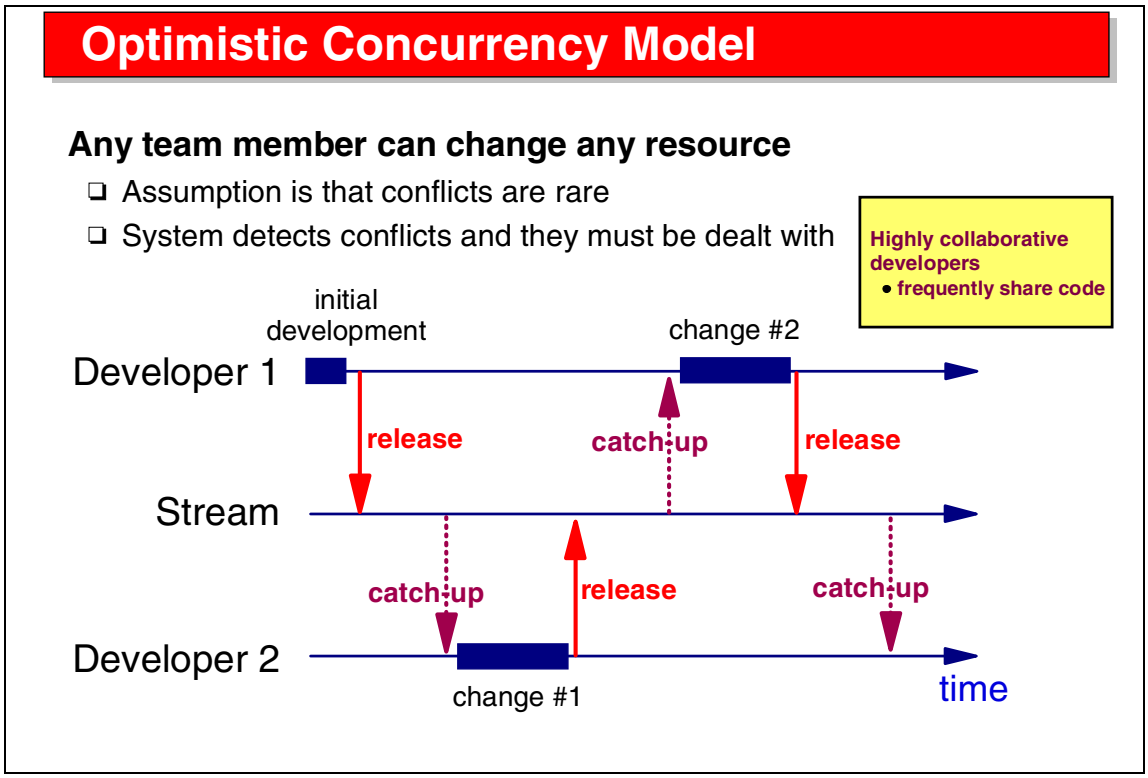
To enable versioning of project data, it is suggested that a versioning system (such as CVS) be used even for a single workstation environment.



Visual 10-5 Terminology

The team development environment uses this terminology:

- Stream—a shared workspace on a team server where project data is stored. Developers share a stream and can work on the same projects.
- Synchronize—the action of a developer to synchronize their own (local) project data with the shared repository. There are two actions a developer performs:
  - Release—making their own changes available to the team, that is, copying changed files to the team stream
  - Catch-up—retrieve changes other developers have made to the local workspace
- Branch—a project may be developed in multiple parallel streams, for example developing version 1.3 and version 2 of a product
- Version—a baseline (frozen code) of a project

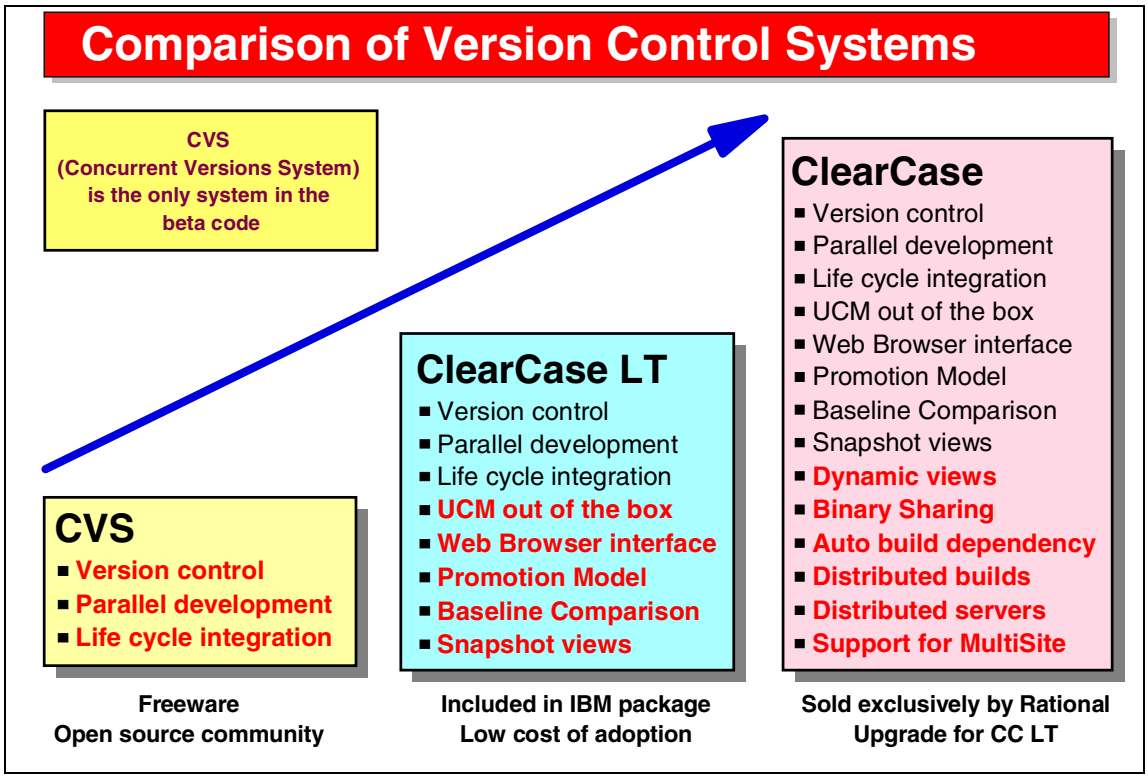


Visual 10-6 Optimistic Concurrency Model

In the optimistic concurrency model, any developer can change any code. This assumes that conflicts are rare because developers usually work on different files.

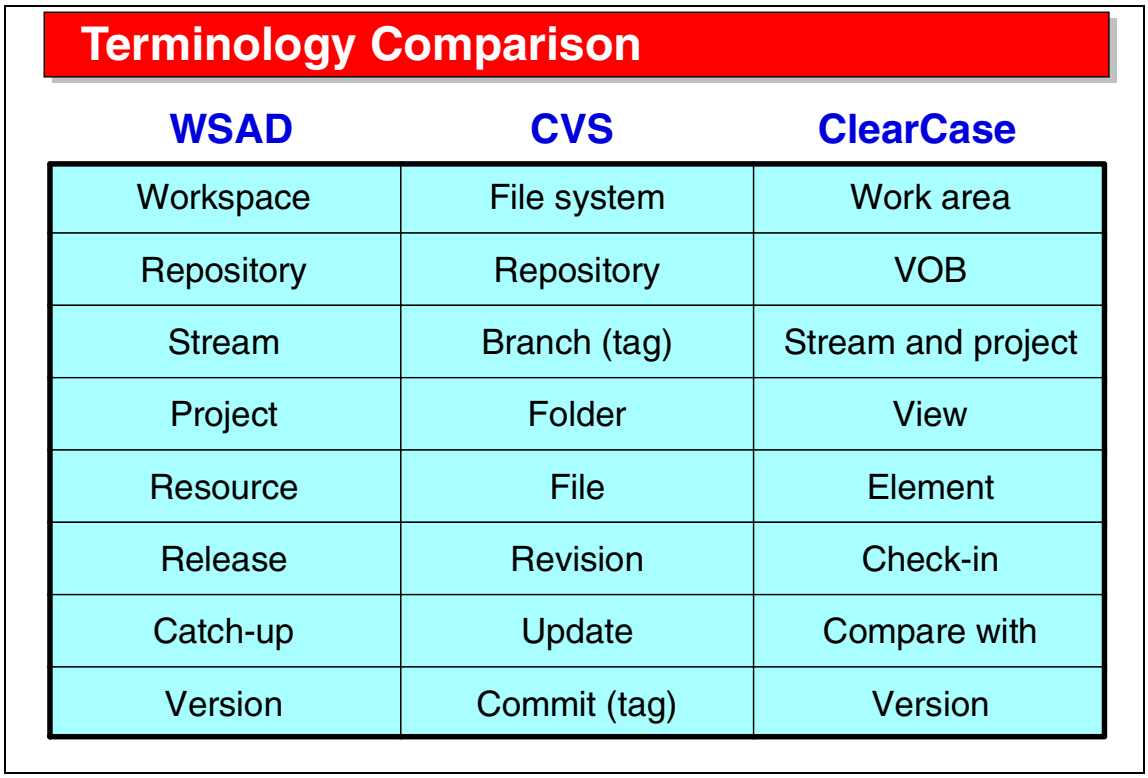
The same file may be updated by multiple developers, but only after changes have been released.

However, conflicts where multiple developers change the same file at the same time do occur and must be dealt with. This will be discussed later in this unit.



Visual 10-7 Comparison of Version Control Systems

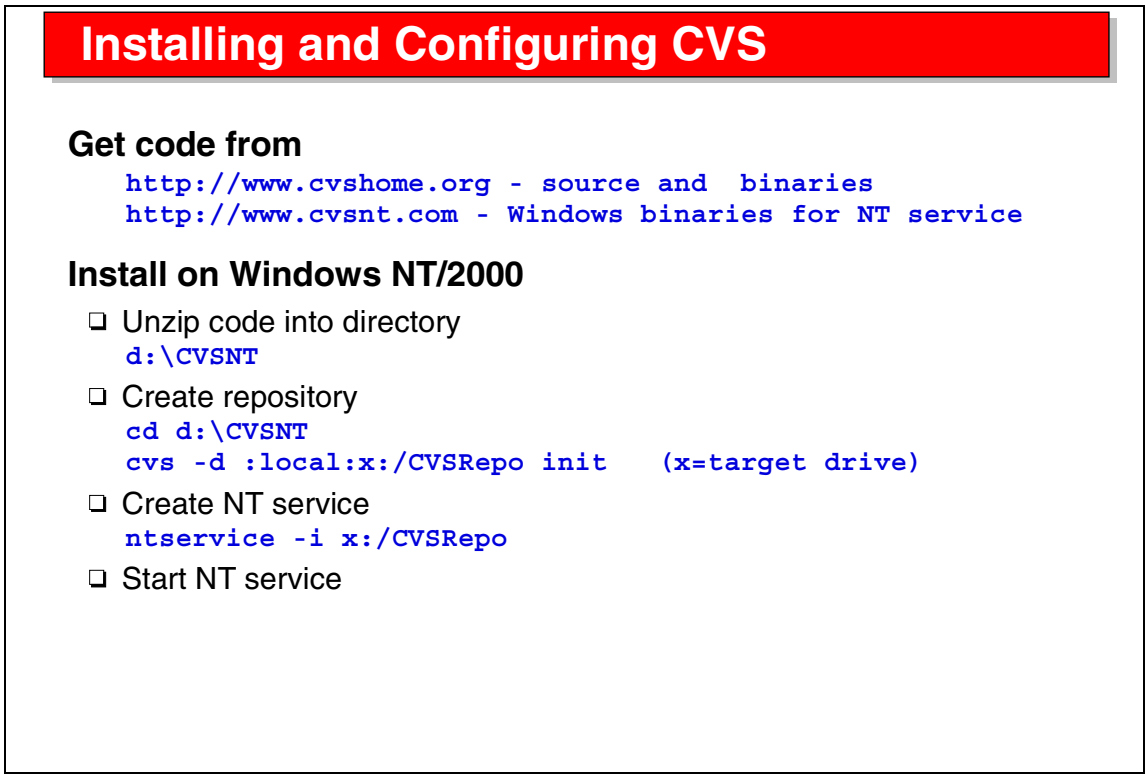
This diagram shows the functionality of the three systems that are supported when the Application Developer became available.



Visual 10-8 Terminology Comparison

When working with the Application Developer, you use the terminology provided by the team development environment.

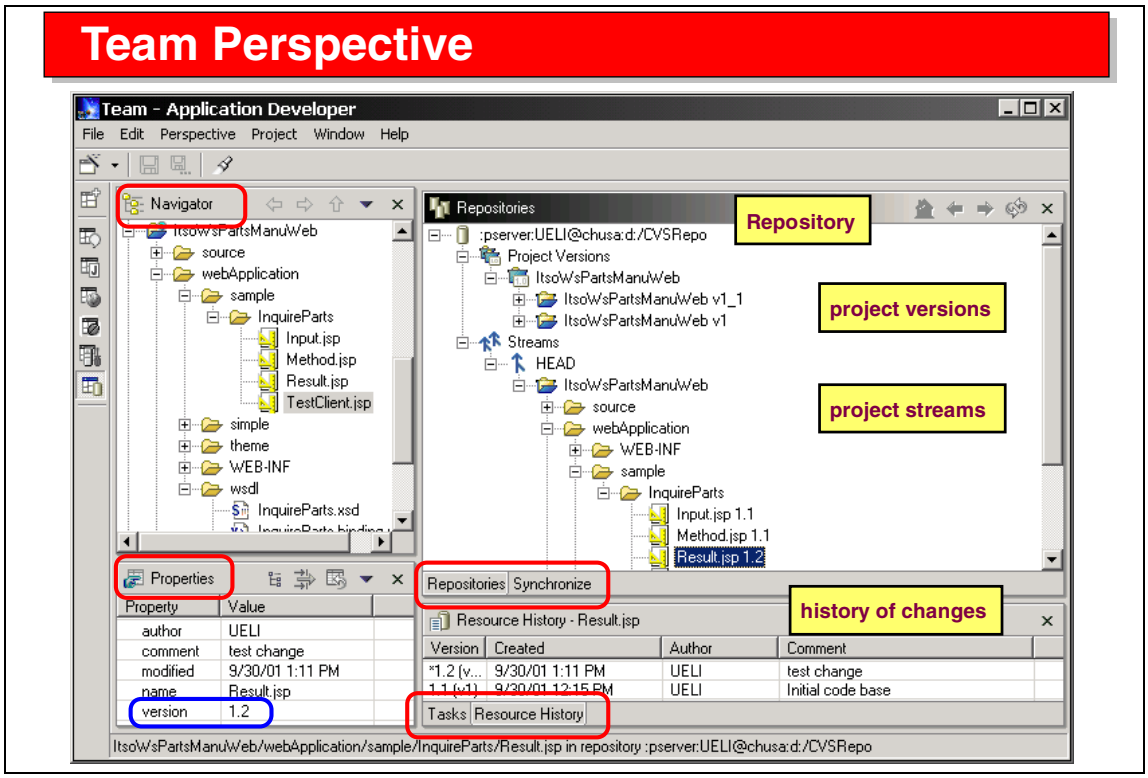
When you use facilities of the versioning systems outside of the Application Developer, you use the terminology provided by those products.



Visual 10-9 Installing and Configuring CVS

Setting up an environment with CVS on Windows is very easy:

- Get the freely available code from the CVS Web site.
- Unzip the code into a product directory.
- Create the repository directory and initialize the repository.
- Create service that can be started and stopped.
- Start the service.



Visual 10-10 Team Perspective

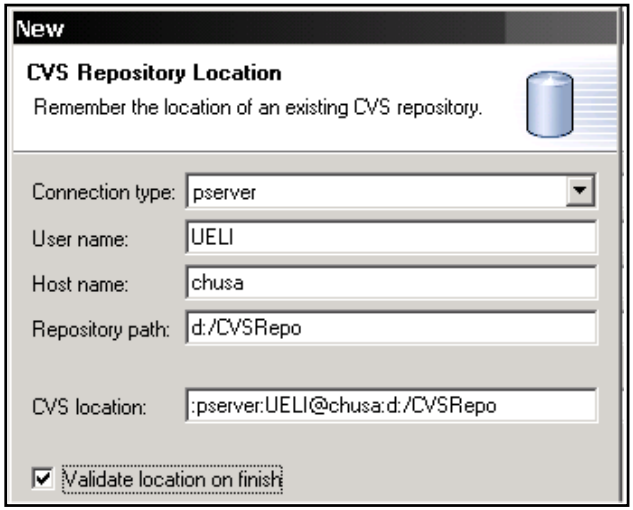
The Team Perspective is used to manage projects in conjunction with a shared repository:

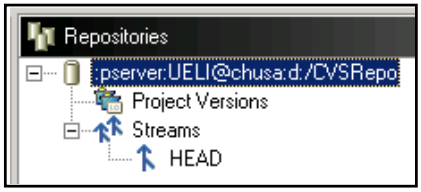
- The Repositories view displays the repository connections, the project versions, and the active project streams with the projects.
- The Synchronize view displays the changes between files in the local workspace and the team stream.
- The Resource History view shows the sequence of changes performed on one file.

Connecting to the Repository

Define repository location

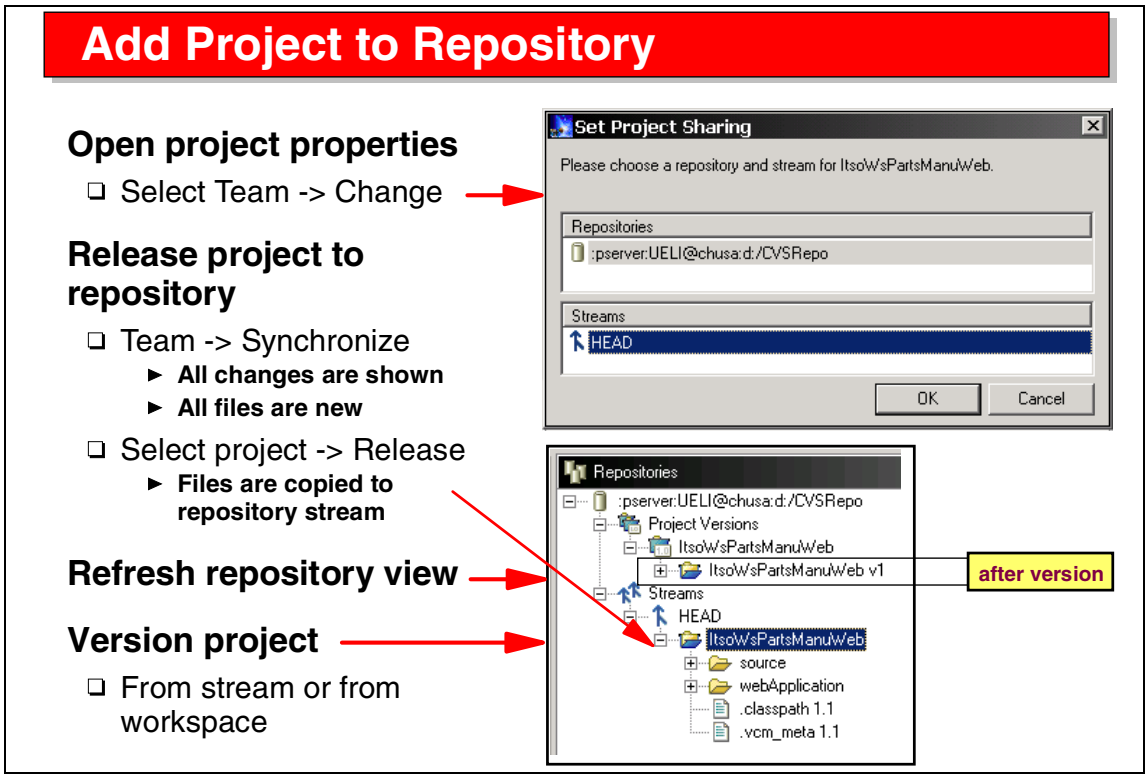
- New CVS Repository
- Connection
  - `pserver`
    - **password server protocol**
  - `ssh`
- Prompt for password
- Connected!





Visual 10-11 Connecting to the Repository

The first task in a CVS environment is to connect to the shared repository.



Visual 10-12 Add Project to Repository

The next task is to add projects to the team stream by invoking the release action. This makes projects available in the shared environment.

An initial version of the project can be established, as well.

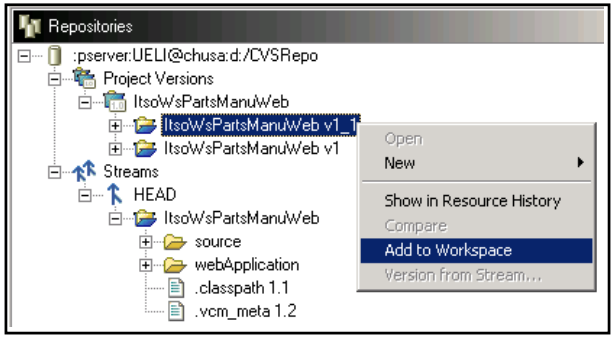
Add Project from Repository

Select project in Repository view

- Select a project version or from a stream

Add to workspace

- Creates the project in the workspace

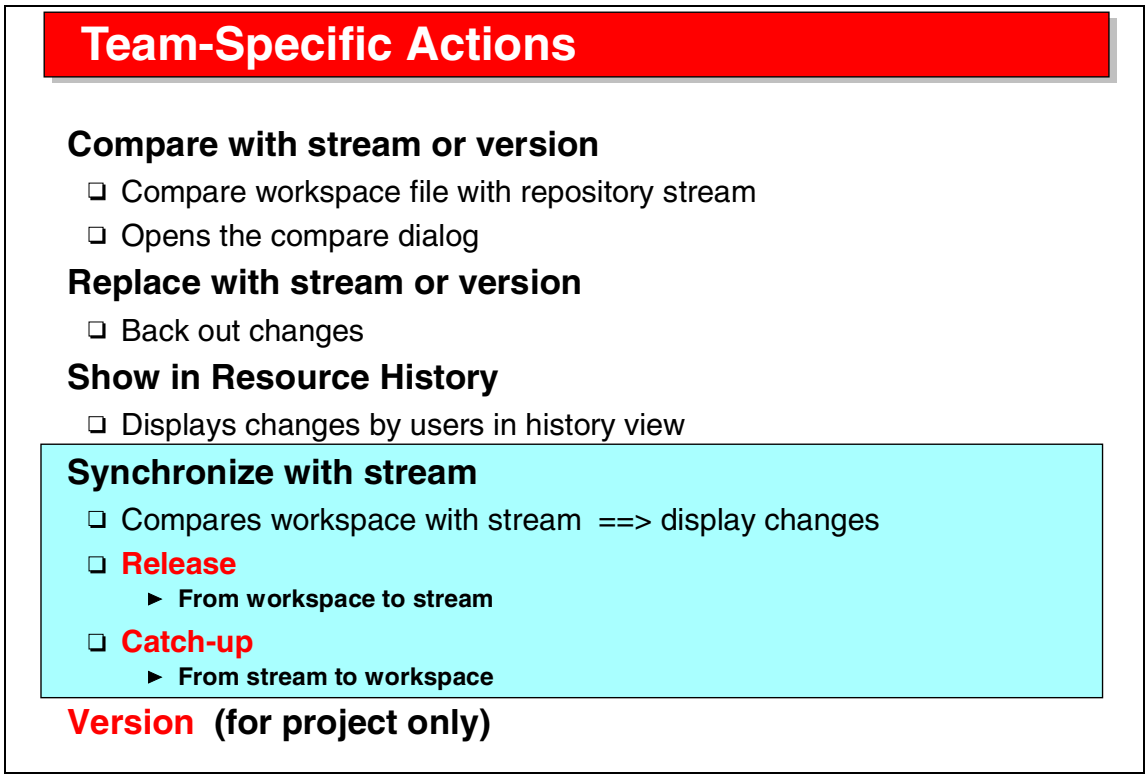


Manage the workspace

- Version project
- Delete from workspace
- Load again when needed

Visual 10-13 Add Project from Repository

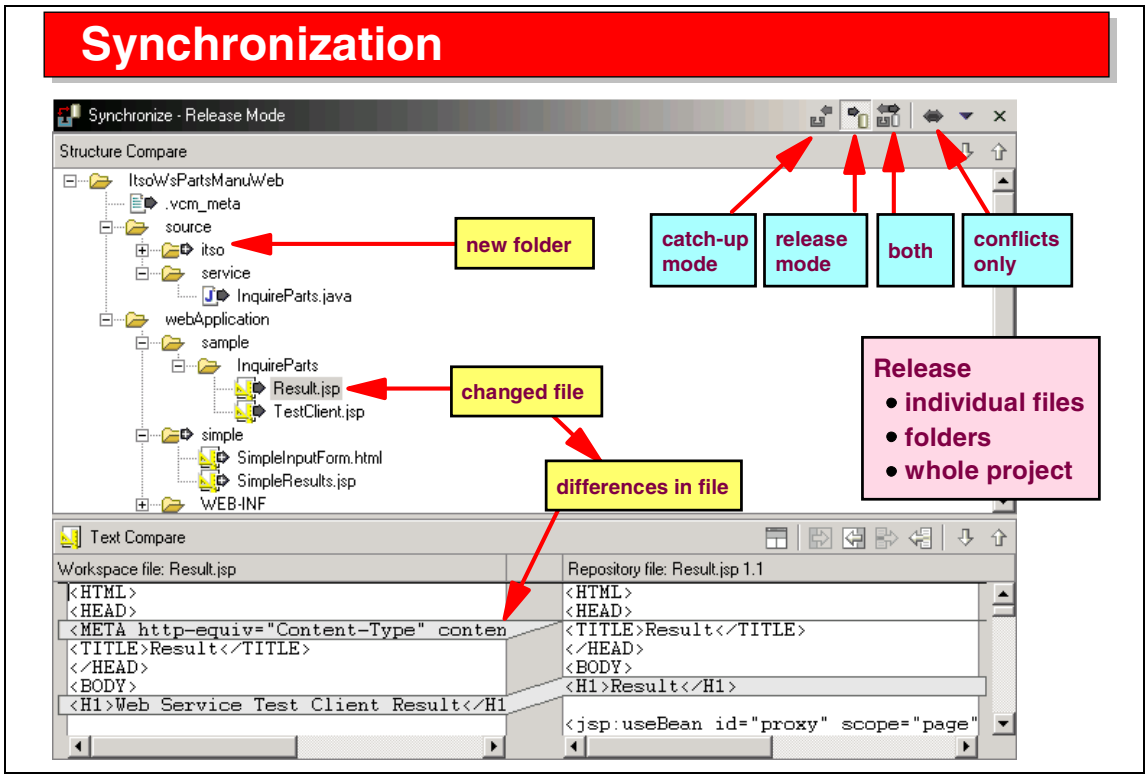
Team members can add projects from the repository by selecting the project from the team stream and adding it to the workspace.



Visual 10-14 Team-Specific Actions

The actions of a team member are:

- Compare the project in the workspace with the project in the team stream.
- Replace the workspace project with the version in the repository.
- Show the history of a file, that is, the changes of all developers that touched the file.
- Synchronize the project in the workspace with the team stream. A dialog showing all the differences is displayed. From this list, the team member can decide to:
  - Release their own changes to the team stream.
  - Catch-up changes of other developers into the workspace.
- A conflict is displayed if the same file has been changed by the team member and other developers. Conflicts must be resolved by merging the changes. This is covered later in this unit under Parallel Development.
- Version the project (from the workspace or from the team stream.



Visual 10-15 Synchronization

The Synchronize view displays the list of changes in the top pane, and the details of a selected change in the bottom pane.

Icons allow to display only catch-up changes (made by other developers), release changes (made by the developer), both together, or conflicts only (same file changed by both).

After viewing changes, a release or catch-up action can be executed.

Synchronization - Conflicts and Ignoring

Conflicts

- User's change conflicts with the change of another user
- Conflicts always shown in synchronized view regardless of mode
- Conflicts must be fixed manually
  - **Parallel development**

Ignoring files from team development

- *.class, *.tmp examples of workspace files to be ignored
- Workbench global ignore
  - **Specify file extension pattern in Team Preferences**
  - ***.class set as default**
- CVS ignore
  - **.cvsignore file in each directory that should be ignored**

Visual 10-16 Synchronization - Conflicts and Ignoring

Conflicts are always displayed and must be resolved.

Certain files are not managed by the team environment:

- Class files
- Temporary files
- Files with extensions set in team preferences
- CVS ignore files, identified by .cvsignore files in directories

Versioning

Version from stream

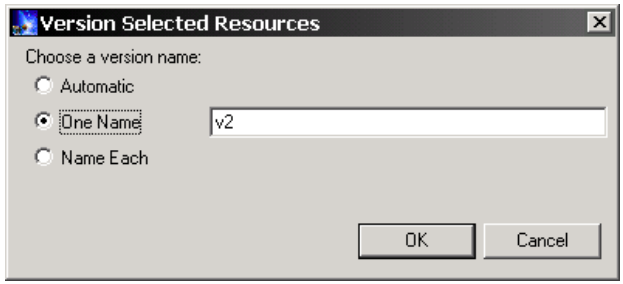
- Attach version name to current state of stream
- Local workspace contents not involved
- Version name is arbitrary string
- Version name retrieves same state of files in the future

Version from workspace

- Releases workspace changes to team stream
- Attaches version name

Version names:

- **Automatic naming**
- **Apply one name to selected stream**
- **Prompt for different name for each project**

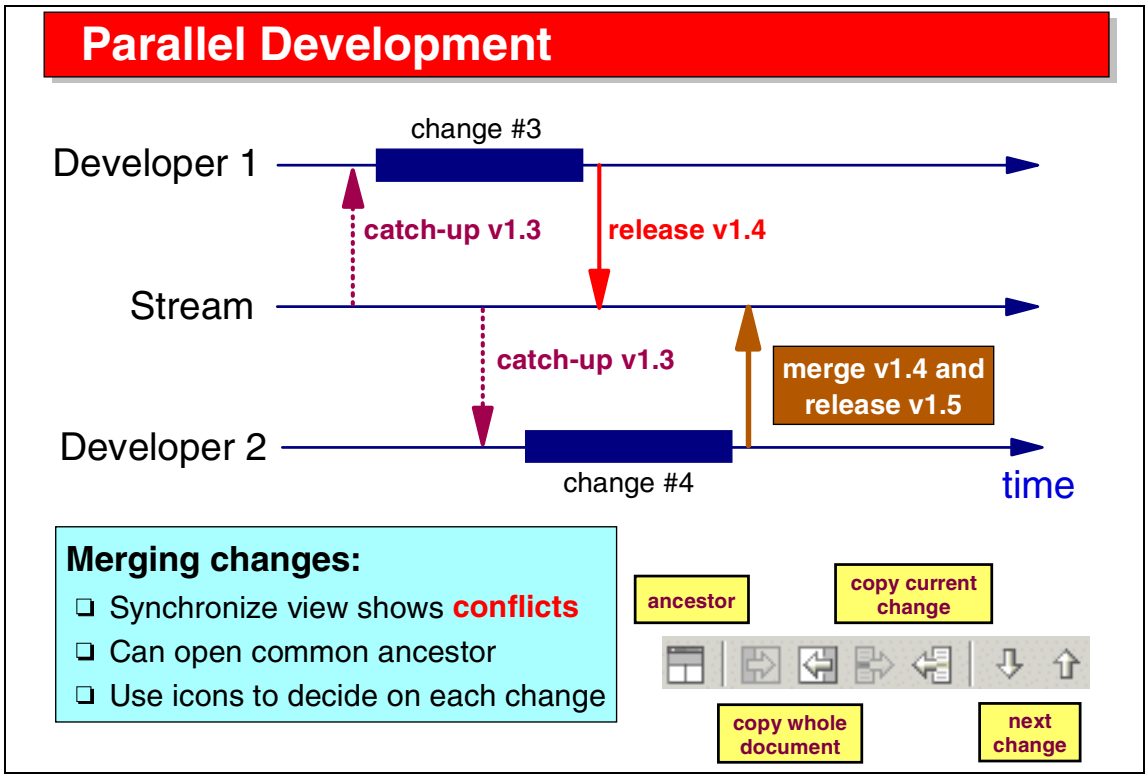


Visual 10-17 Versioning

Project versions can be created from the workspace or from the team stream.

When versioning from the workspace, a synchronize action is performed first to update the team stream with the workspace data.

Version names can be assigned automatically, or by specification.



Visual 10-18 Parallel Development

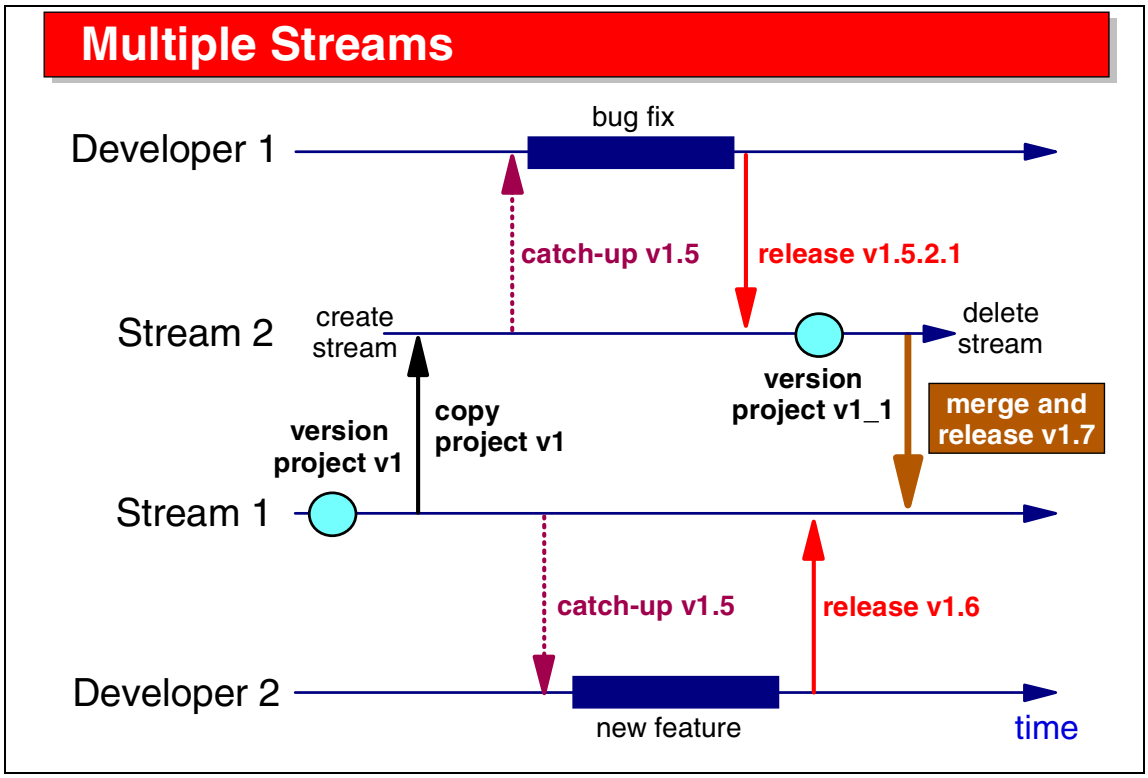
In parallel development, the same file (or files) are updated by multiple team members at the same time.

The first member to synchronize with the stream has no conflicts and can release the changes.

The second member to synchronize gets a conflict and must resolve it by:

- Studying the changes made by the developers
- Optionally, opening the common ancestor from which the changes were made
- Acting on individual changed lines and copying from either of the two developers
- Acting on the whole file and copying one of the two files
- Releasing the merged copy

Icons are provided for these operations.



Visual 10-19 Multiple Streams

A project can be developed in multiple streams, for example to perform maintenance on one version of a product, while developing a new version at the same time.

It is possible to merge the two stream later in the process.

A new stream must be created from a version of the project.

Summary

Team development provides for

- Shared repository
  - **Manage projects in repository**
  - **Add project: workspace to repository, repository to workspace**
- Synchronize workspace with repository
  - **Release changes to repository**
  - **Catch-up up changes from other developers**
- Conflict management
  - **Optimistic concurrency allows conflicts**
  - **Conflicts must be resolved**
    - **Merge of code**

CVS or CCLT

- Suggested even for single developer
- Workspace management ===> versions of own projects

Visual 10-20 Summary

The team development environment provides the necessary function for multiple developers to work on the same project.

Even for a single workstation, a team repository provides the much-needed function of versioning the projects.

11

Web Services Overview

Visual 11-1 Title

Objectives

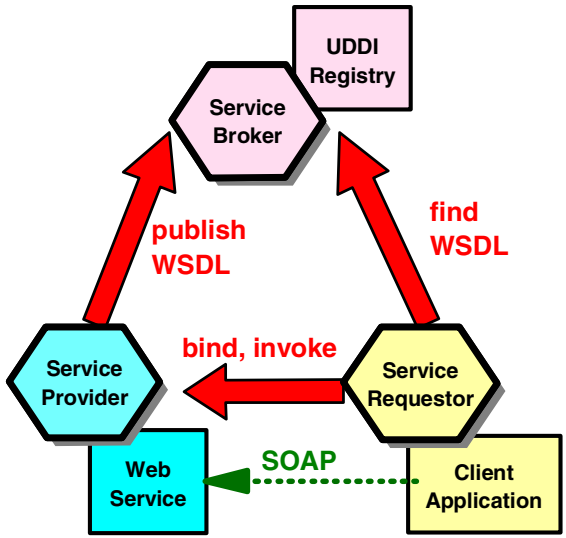
Learn about Web Services

- Service Web
- Web Services components
- SOAP
  - **Messages**
  - **Data Model**
  - **Apache SOAP server**
- WSDL
  - **Interface**
  - **Implementation**
- UDDI Registry
  - **Business entities**
  - **Business services**

Web Services development

- **Static and dynamic Web Services**

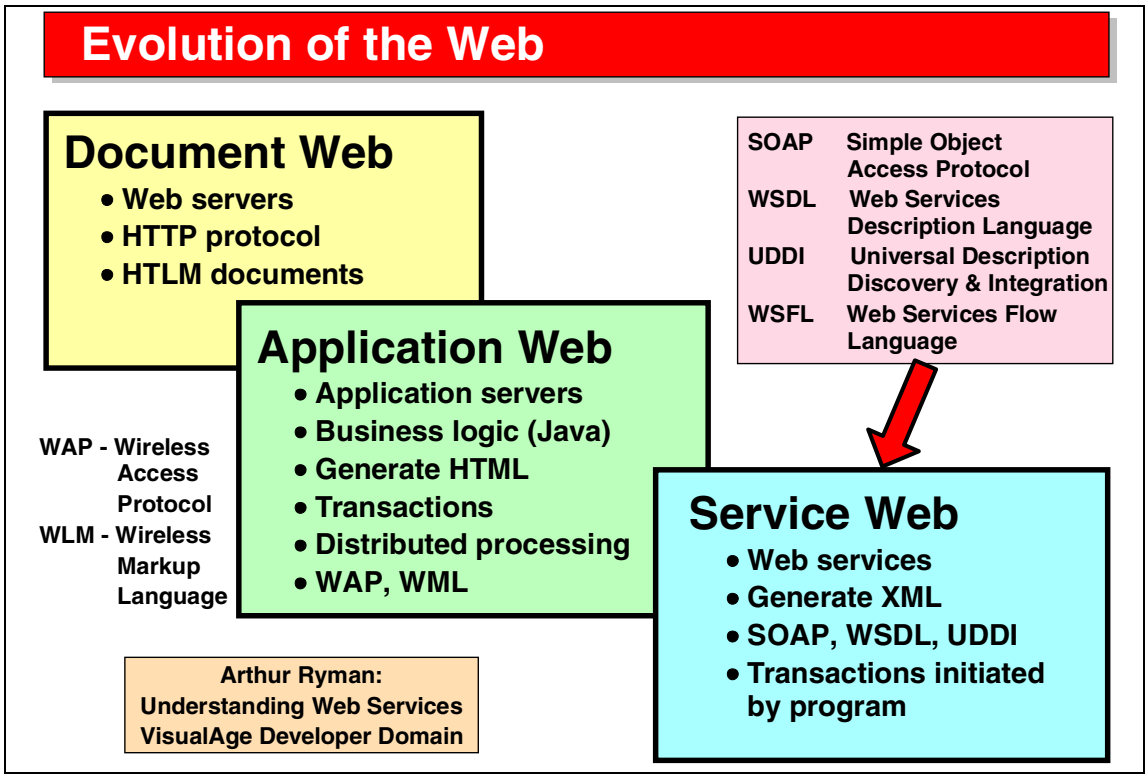
WSAD Web Services tools



Visual 11-2 Objectives

The objectives of this unit are to:

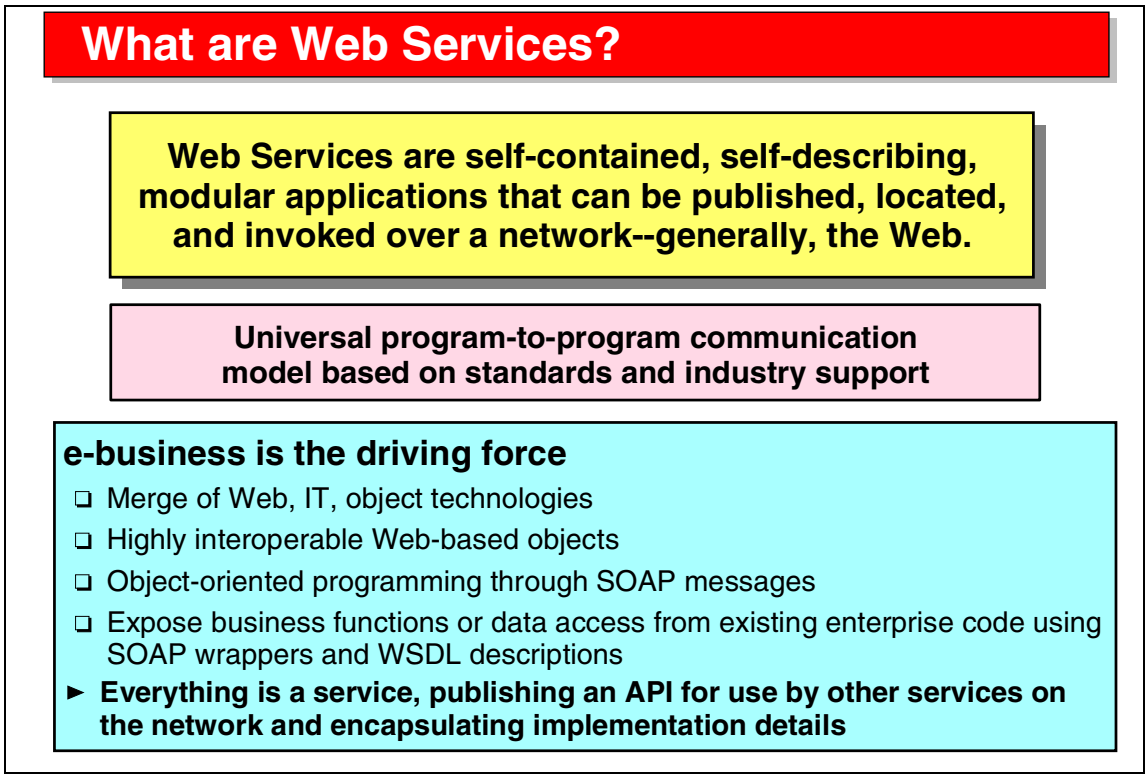
- Understand the Web Services technology
- Understand the underlying technologies of
  - Simple Object Access Protocol (SOAP)
  - Web Services Description Language (WSDL)
  - UDDI Registry
- Understand how Web Services are developed and the tools that are provided within the Application Developer



Visual 11-3 Evolution of the Web

The Web has evolved from static content (documents) to dynamic content through application servers that provide business logic (CGI programs and Java) and transaction, as well as new protocols such as wireless access protocol (WAP) and wireless markup language (WML).

The next step is the introduction of Web Services that provide access to Web applications, dynamic content, and transactions from programs.



Visual 11-4 What are Web Services?

There are a number of definitions what Web Services are.

The important aspect is that callable functions are made available to programs on the Web.

Web Services Attributes and Examples

- Self-contained
  - **No additional software (HTTP, XML, Application Server)**
- Self-describing
  - **Definition of message travels with message**
- Modular
  - **Callable services**
- Published, located, invoked (SOAP, WSDL, UDDI)
- Language-independent and interoperable
  - **Different environments, can make existing code into a Web Service**
- Open and standards-based
  - **HTTP, XML**
- Dynamic
  - **Discovery and invocation can be automated**
- Composable
  - **Web Service can invoke other Web Services**

Examples

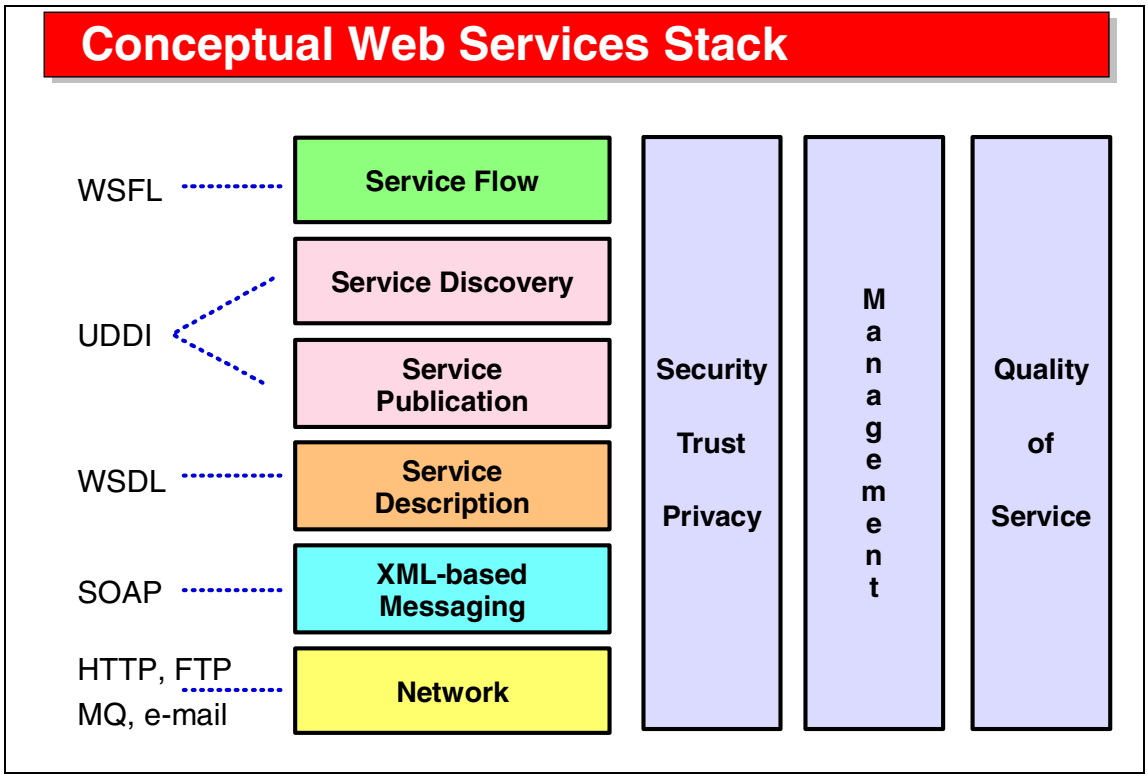
- Business information with rich content
  - **Weather reports**
  - **Stock quotes**
  - **Airline schedules**
  - **Credit check**
  - **News feed**
- Transactional Web Services for B2B, B2C
  - **Airline reservation**
  - **Rental car agreement**
  - **Supply chain mgmt**
- Business process externalization
  - **Business linkage at workflow level**
  - **Complete integration at process level**

Visual 11-5 Web Services Attributes and Examples

Web Services have a number of characteristics.

The most important aspect is the interoperability through platform- and language-independence.

A number of Web Services are already available on the Internet.



Visual 11-6 Conceptual Web Services Stack

SOAP is the XML-based messaging facility built on top of protocols such as HTTP and others.

WSDL describes the interface of the Web Service and where an implementation is running.

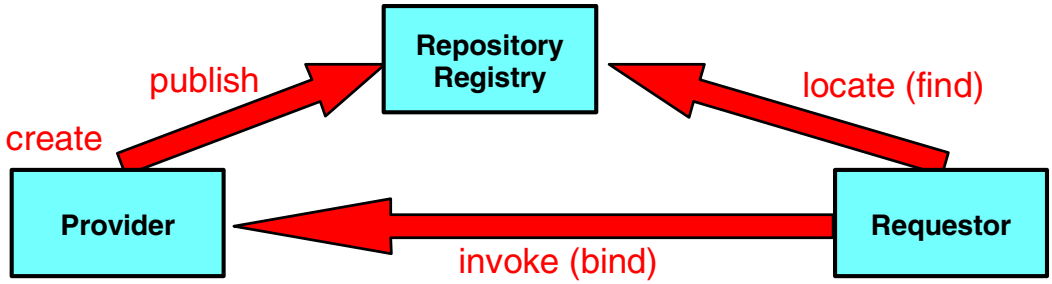
UDDI provides facilities to publish and find Web Services.

WSFL, the Web Services Flow Language, provides higher-level information how business applications flow through a series of Web Services.

There are issues to be resolved, such as a standardized mechanism for security, management facilities, and good standards for quality of service.

Web Services Components

- A Web Service has to be **create**d, and its interfaces and invocation methods must be defined
- A Web Service has to be **publish**ed to one or more intranet or Internet repositories for potential users to locate
- A Web Service has to be **locate**d to be invoked by potential users
- A Web Service has to be **invoke**d to be of any benefit
- A Web Service may have to be unpublished when it is no longer available or needed



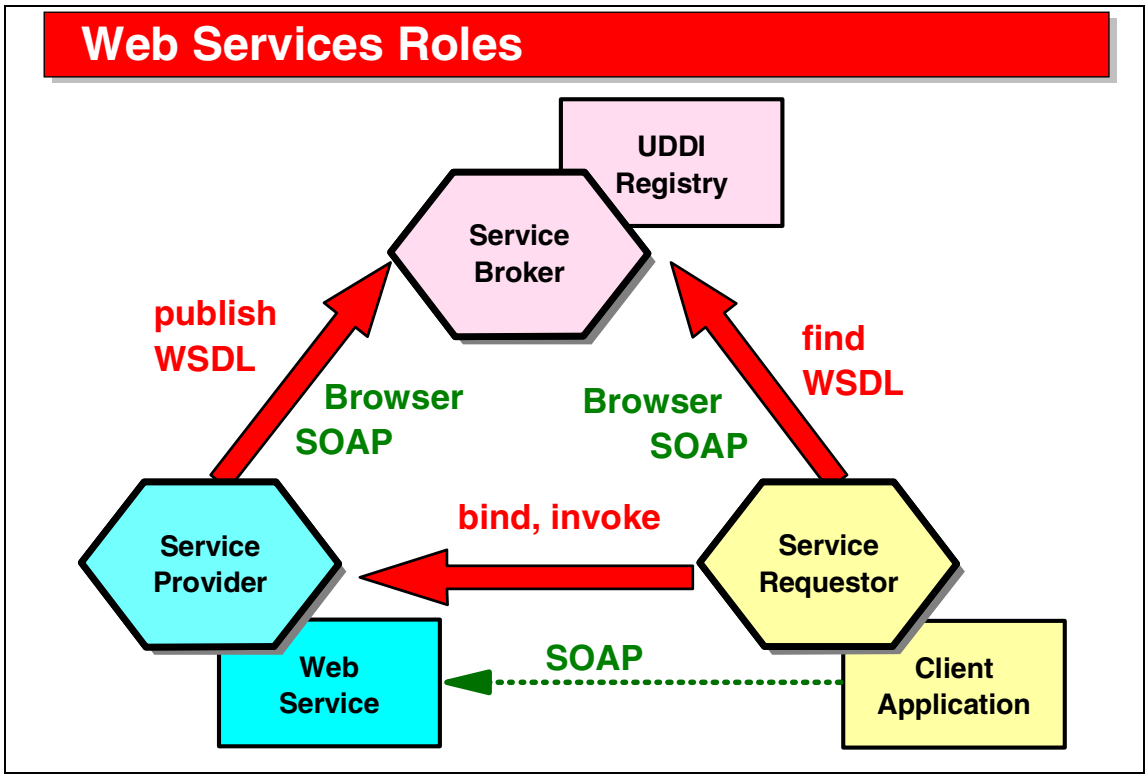
Visual 11-7 Web Services Components

First, a Web Service must be created, and its interfaces well defined (WSDL).

To advertise Web Services they can be published in a UDDI Registry, where they can be located by potential users.

A Web Service can then be invoked on the application server where the service is installed.

In the Web Services world we therefore have providers that create and publish Web Services, requestors that find and invoke Web Services, and registries where Web Services are published.



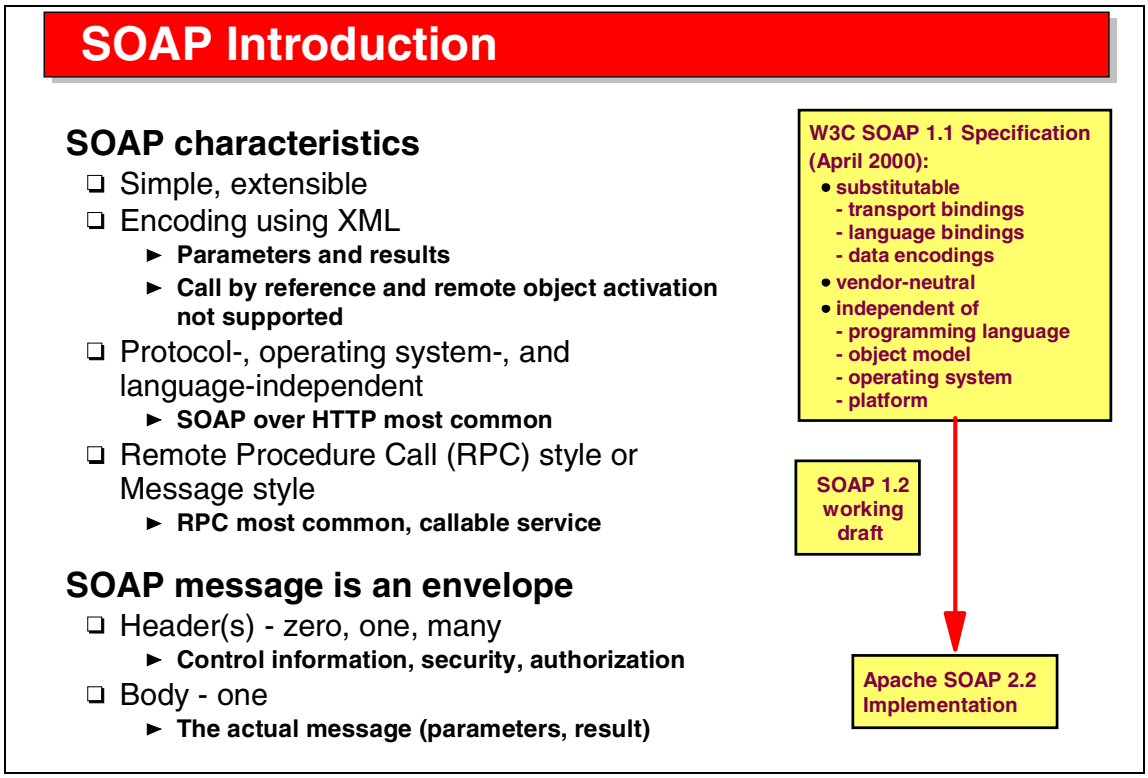
Visual 11-8 Web Services Roles

The Web Services provider creates the Web Service and installs it on an application server.

The Web Services requestor writes the client application that invokes the Web Service.

The Web Services broker runs a UDDI Registry where providers publish their Web Services and requestors find the Web Services.

SOAP is the protocol to invoke a Web Services, and it is also a protocol that can be used to publish and locate Web Services in the UDDI Registry. The registry can also be accessed from a Web browser.



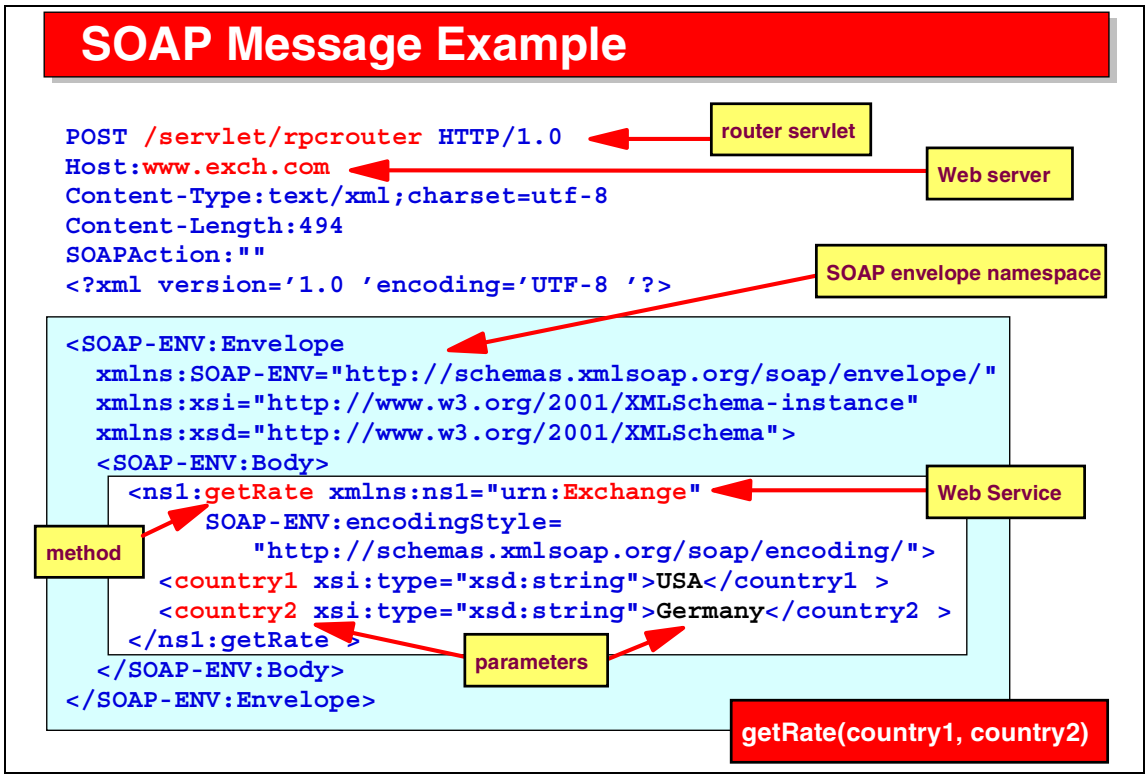
Visual 11-9 SOAP Introduction

The major advantages of SOAP are:

- All data is transferred in XML format. This includes the parameters used in a call, as well as the result data.
- It is protocol-, platform-, operating system-, and language-independent. HTTP is the most used protocol.

SOAP supports the remote procedure call (RPC) protocol, as well as an asynchronous message style protocol.

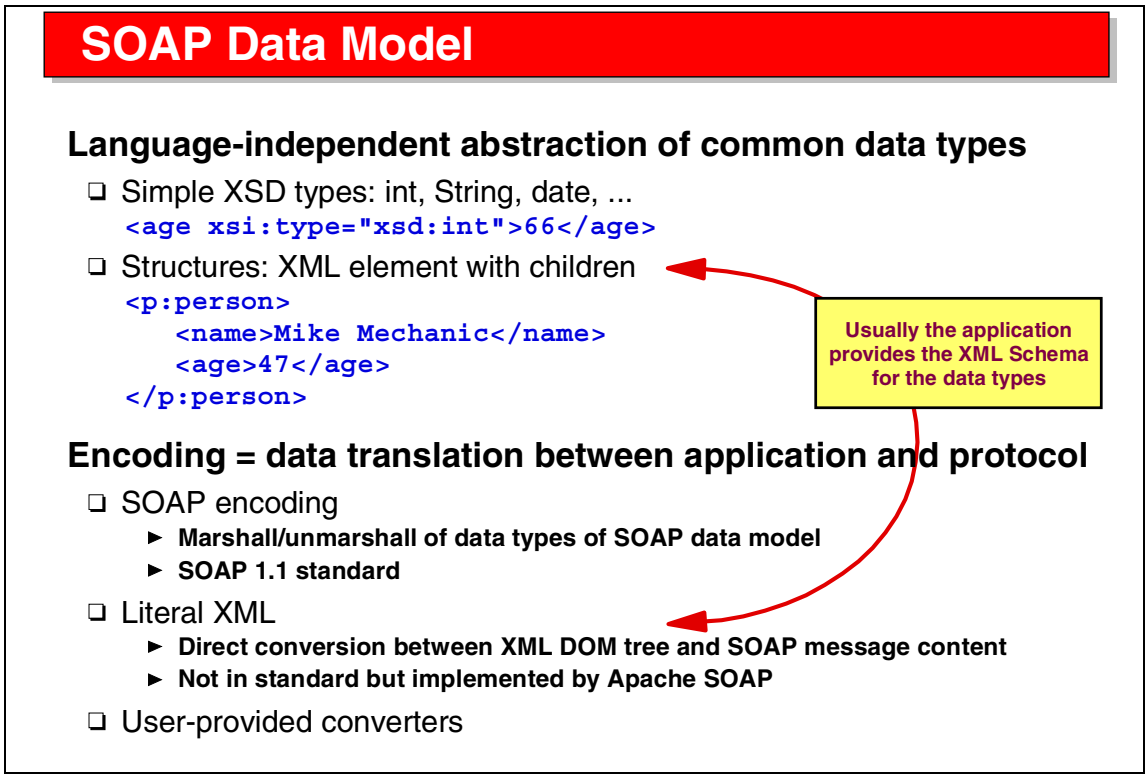
A SOAP message is an envelope containing optional headers and always one body with the actual message containing the parameters or results.



Visual 11-10 SOAP Message Example

This SOAP message example shows the invocation of a Web Service:

- The Web Service returns the exchange rate between the currencies of two countries.
- The service is installed at www.exch.com.
- An rpcrouter servlet is invoked to route the call to the Exchange Web Service.
- The getRate method of the Exchange service is invoked with two parameters named country1 and country2, both being of string data type.
- The Web Service call is embedded in the body within the SOAP envelope.

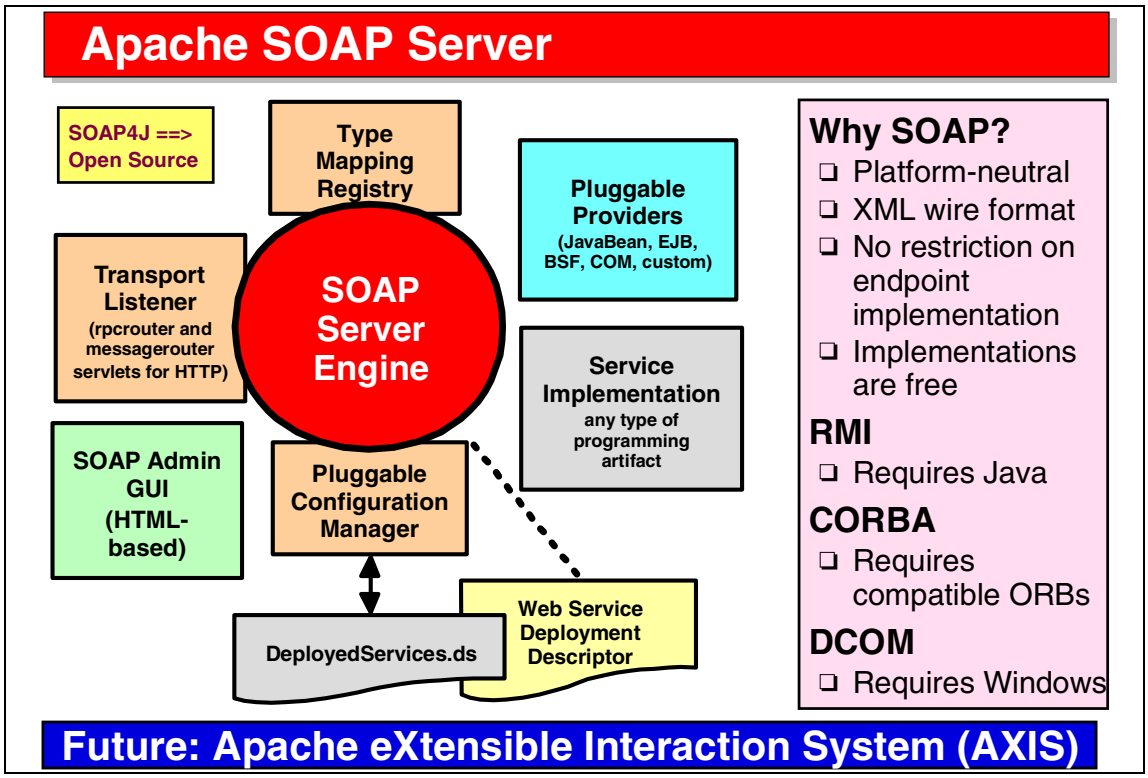


Visual 11-11 SOAP Data Model

The SOAP data model provides definitions for the most used data types, such as strings, integers, float, double, date.

Result data is usually more complicated and is described in an XML schema provided by the application.

The process of translating the data (parameters and result) into XML is called *encoding*. Simple data can be encoded by the SOAP data types, the Element class of an XML DOM tree (XML in memory) can be encoded using literal XML, and user-provided converters can be used as well.



Visual 11-12 Apache SOAP Server

The Apache SOAP server is implemented in WebSphere Application Server and in the Application Developer.

The Apache SOAP server is based on the IBM SOAP4J API and provides the transport listeners (such as the rpcrouter servlet), an administration GUI, a pluggable configuration manager that reads SOAP deployment descriptors.

Service Implementation and Client Example

```
public class Exchange {
   public float getRate(String country1, String country2) {
       // lookup exchange rate in table
       return rate; }
}
```

provider

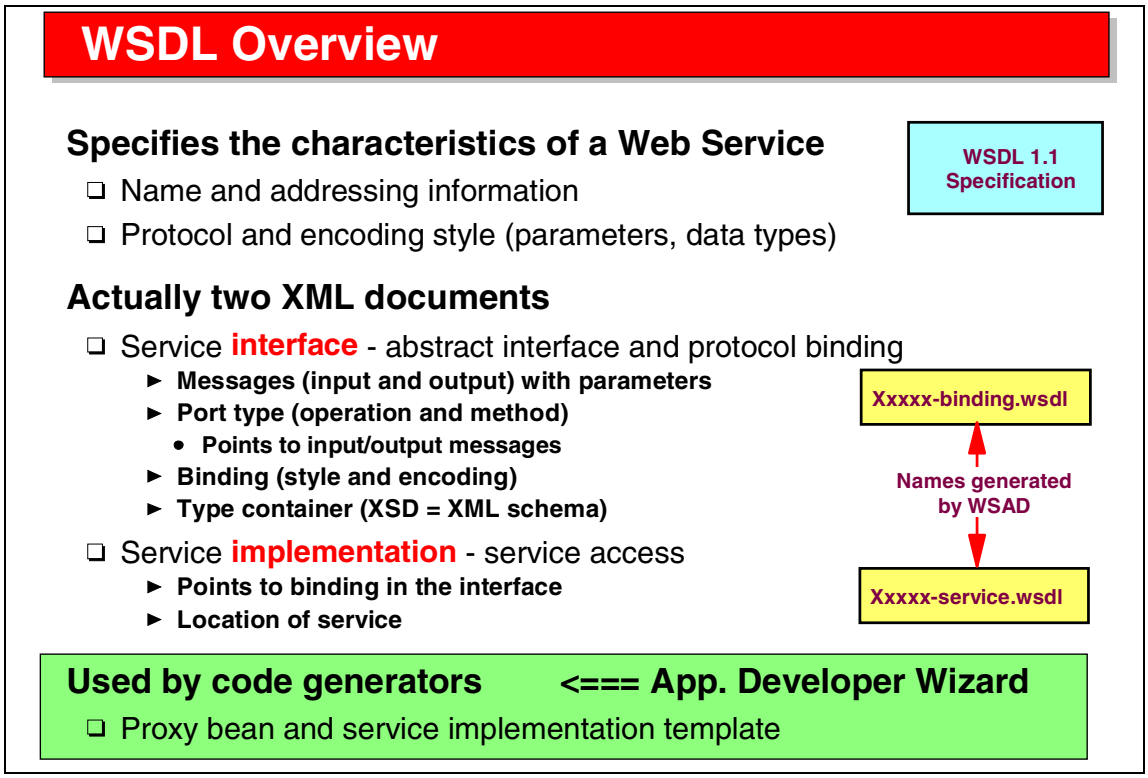
```
public class SoapClient {
public void static main(String[] args) {
  Call call = new Call();
  call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
  call.setTargetObjectURI("urn:Exchange");
  call.setMethod("getRate");
  Vector parms = new Vector();
  parms.addElement( new Parameter("country1", String.class,
    "USA", Constants.NS_URI_SOAP_ENC); // ... 2nd parm
  call.setParams(parms);
  URL url = new URL("http://www.exch.com/soap/servlet/rpcrouter");
  Response resp = call.invoke(url, "");
  if ( !resp.generatedFault() )
    Object obj = (resp.getReturnValue()).getValue();
    // process result
}
```

requestor

Visual 11-13 Service Implementation and Client Example

The two code fragments show the server (provider code) and client (requestor code) of the Exchange Web Service:

- The server code shows a JavaBean that implements the Web Service. (The actual code to query a database for the exchange rate is not shown.)
- The client code shows that a SOAP Call object is instantiated and initialized with encoding style, Web Service name, method name, and parameters (stored in a Vector). Then the Web Service is invoked and the result object is extracted. (The processing of the result is not shown.)



Visual 11-14 WSDL Overview

The WSDL language is used to describe a Web Service. Two XML files are used:

- The interface file describes the Web Service, including the method that is called, the parameters that are passed, and the encoding that is used.
- The implementation file describes where the Web Service is installed and how it is accessed. The implementation file points to the interface file.

The coding of WSDL files is quite difficult, but in the case of the Application Developer these files are generated. The naming convention that is used by the Application Developer is:

```
Xxxxxxx-binding.wsdl
Xxxxxxx-service.wsdl
```

WSDL Interface Example

Service type in UDDI

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ExchangeRemoteInterface"
  targetNamespace="http://www.exch.com/definitions/Exchange...
      ...RemoteInterface" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://www.exch.com/definitions/ExchangeRem..face"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <message name="getRateRequest">
    <part name="country1" type="xsd:string"/>
    <part name="country2" type="xsd:string"/>
  </message>
  <message name="getRateResponse">
    <part name="result" type="xsd:float"/>
  </message>
  <portType name="Exchange">
    <operation name="getRate">
      <input name="getRateRequest" message="tns:getRateRequest"/>
      <output name="getRateResponse" message="tns:getRateResponse"/>
    </operation>
  </portType>
  ...continued...
```

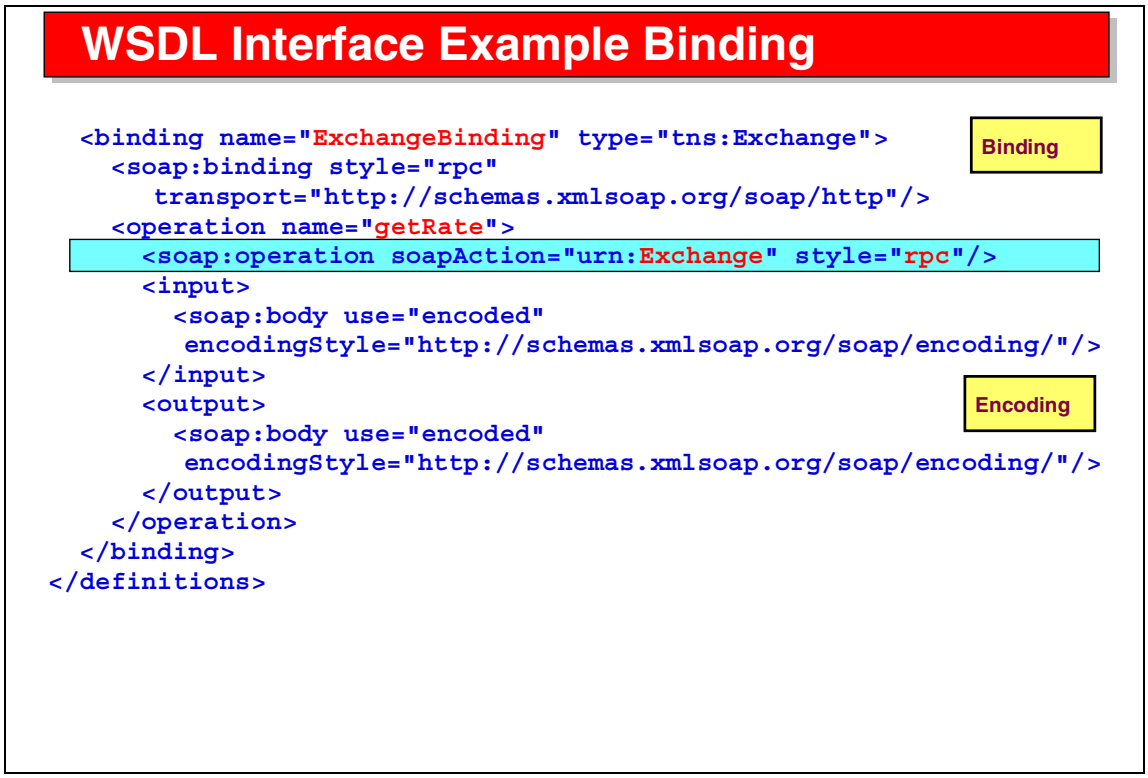
Messages

Port

Visual 11-15 WSDL Interface Example

This example shows the interface file of the Exchange Web Service:

- The targetNamespace is the name of the Web Service as used in the UDDI Registry.
- The message entries describe the input and output messages, with their parameters.
- The portType specifies the name and operation (method name) and points to the input and output message.
- Continue to visual on next page.



Visual 11-16 WSDL Interface Example Binding

- The binding specifies the SOAP operation and style (RPC, in this case) and the default encoding

WSDL Implementation Example

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ExchangeService"
    targetNamespace="http://www.exch.com/wsdl/Exchange-service.wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://localhost/wsdl/Exchange-service.wsdl"
    xmlns:binding=
      "http://www.exch.com/definitions/ExchangeRemoteInterface"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <import namespace=
      "http://www.exch.com/definitions/ExchangeRemoteInterface"
    location="http://www.exch.com/wsdl/Exchange-binding.wsdl"/>

  <service name="ExchangeService">
    <port name="ExchangePort" binding="binding:ExchangeBinding">
      <soap:address
          location="http://www.exch.com/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

WSDL files

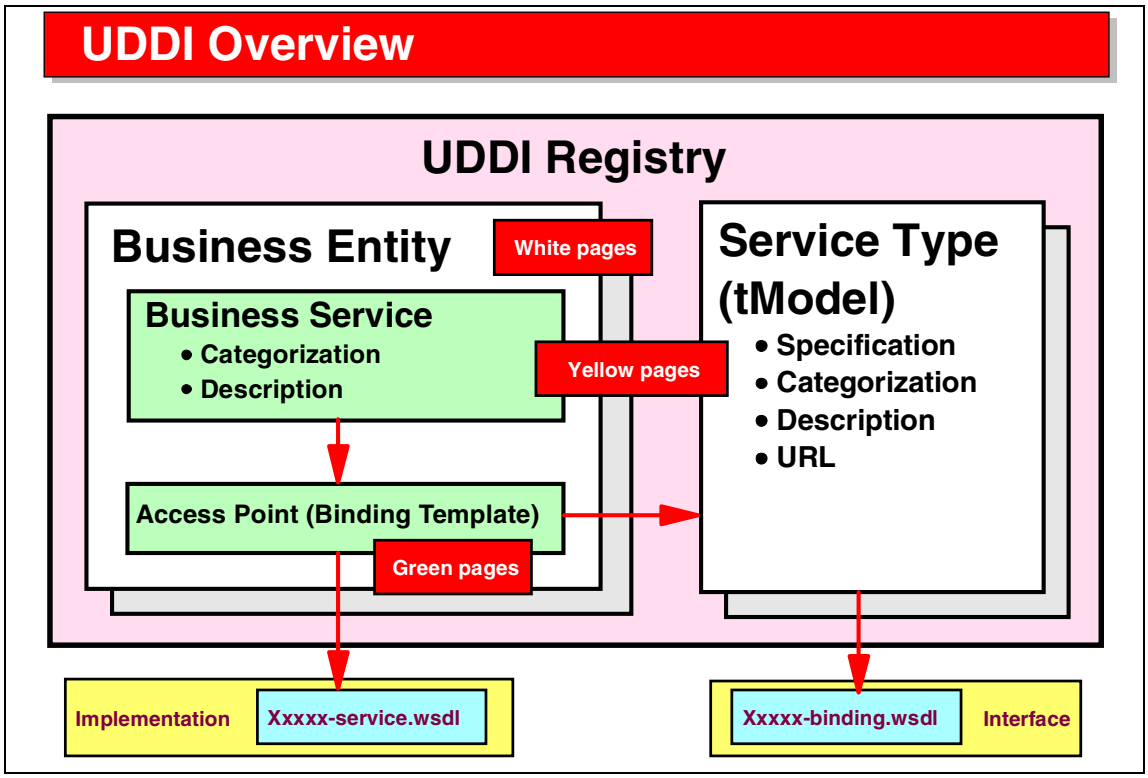
Target location

Link to interface

Visual 11-17 WSDL Implementation Example

This example shows the implementation file of the Exchange Web Service:

- The targetNamespace is the name of the implementation file itself.
- The import points to the binding file (the interface).
- The soap:address points to the location where the services is running.



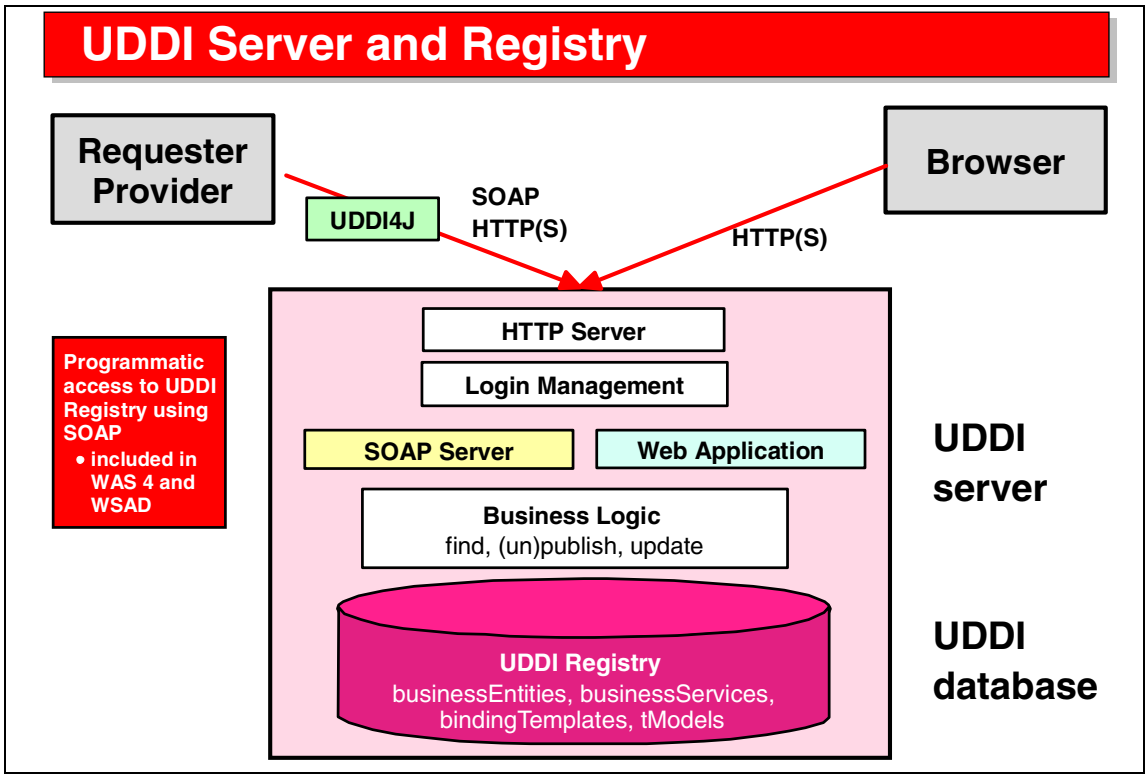
Visual 11-18 UDDI Overview

A UDDI Registry contains these entries:

- Business entities—companies that want to register Web Services
- Business services—a Web Service that the company registers (this is a descriptive entry)
- Access point (called binding template)—points to an installed Web Service (the target address) and to the matching WSDL service (implementation) file
- Web Service type (called tModel)—a Web Service definition that points to the matching Web Service binding (interface) file

Entries in the registry can be qualified with descriptions and categorizations.

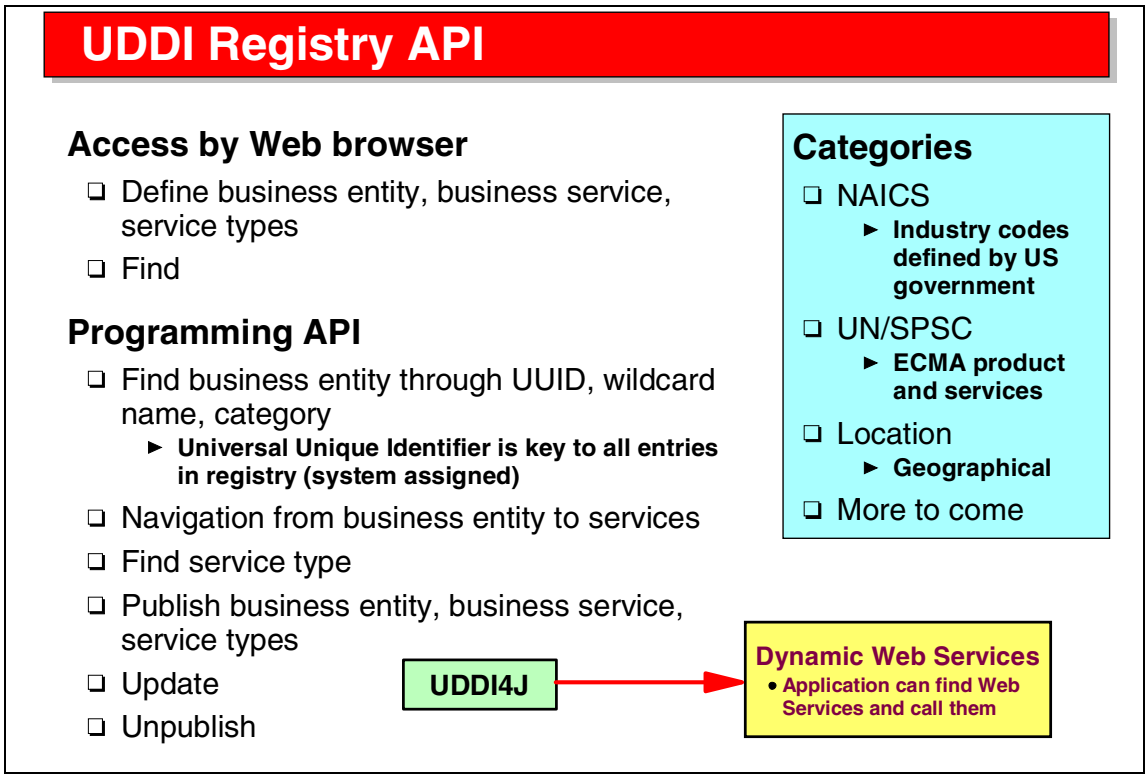
Note that the WSDL files are not stored in the registry, rather they are pointed as HTTP addresses to the Web Service provider.



Visual 11-19 UDDI Server and Registry

A UDDI Registry runs on a UDDI server. The registry is really a Web application that can be accessed by a browser or by a programmable API, such as UDDI4J (UDDI for Java), through the SOAP protocol.

In the IBM implementation, the registry entries are stored in a DB2 database.

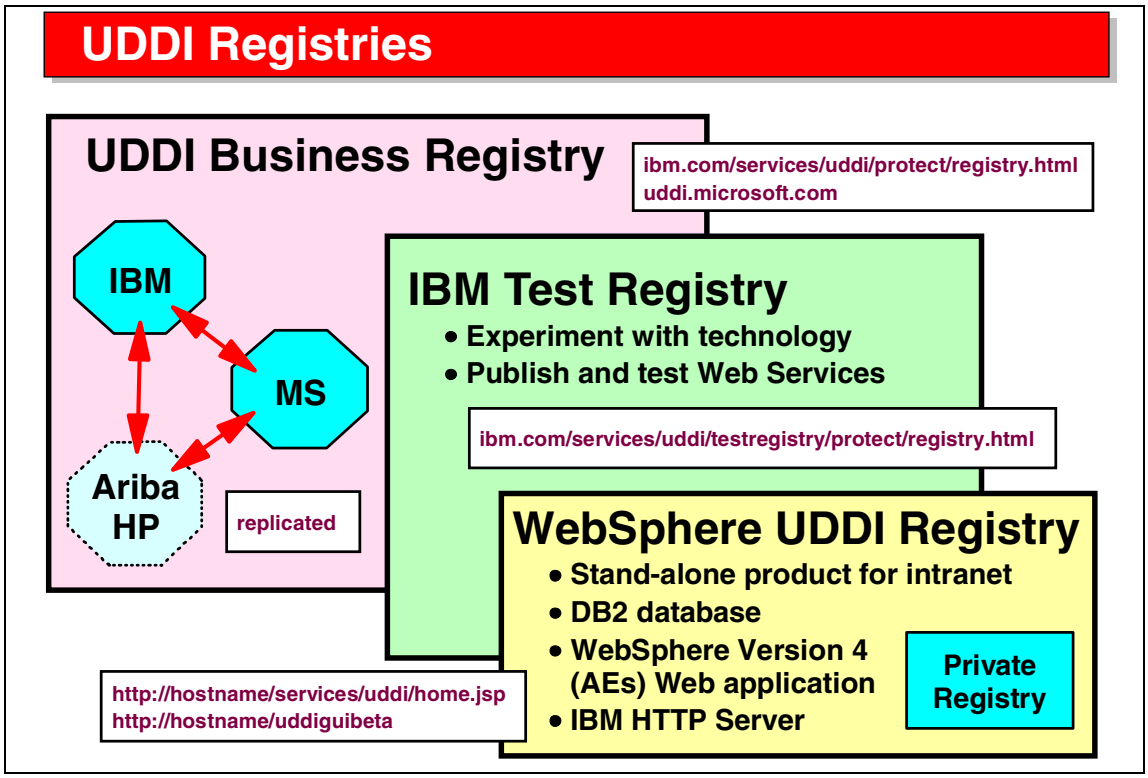


Visual 11-20 UDDI Registry API

Each entry in the UDDI Registry has a universal unique identifier (the key in the database).

Through a browser or through the UDDI4J API a user can traverse the registry from business entities, to business services, to binding templates, and to tModels (and can also move in the reverse direction).

The categorization of entries can be done according to the NAICS or UN/SPSC standards, or geographical location.



Visual 11-21 UDDI Registries

IBM, Microsoft, and other companies run the official UDDI Business Registry, which is replicated between the companies (so it does not matter where an entry is made).

IBM provides a UDDI Test Registry, where any company can make a few entries for testing of Web Service and UDDI.

IBM provides a stand-alone registry product, the IBM WebSphere UDDI Registry. This product, currently at beta level, can be installed by a company as a private UDDI Registry. This product is a Web/EJB application that is installed into WebSphere Application Server AE or AEs.

Web Services Flow Language

WSFL

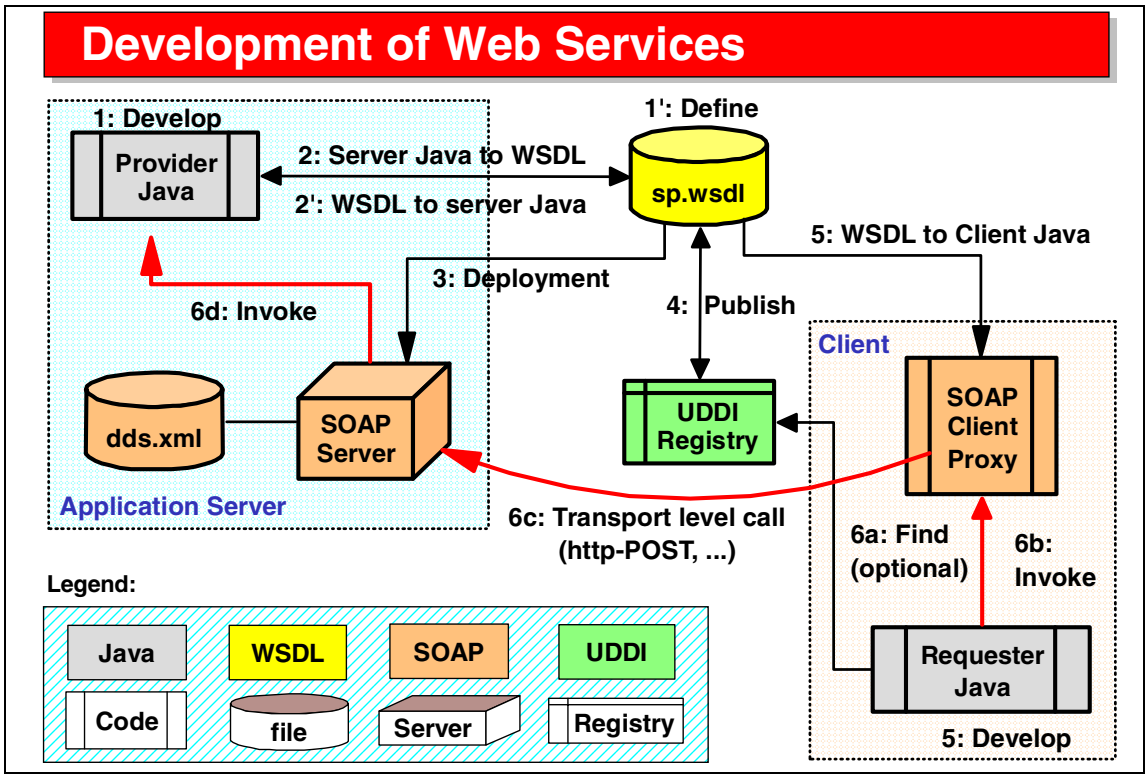
- XML language to describe Web Services compositions
- Usage pattern of a collection of Web Services
  - **How to achieve a business goal**
  - **How to execute a business process**
  - **Flow composition (orchestration, choreography)**
  - **Defines flow of control and data**
- Interaction pattern of a collection of Web Services
  - **Describe overall partner interactions**
- Extensive support for recursive composition of services
- Layered on top of WSDL

WSFL white paper:

ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Visual 11-22 Web Services Flow Language

The Web Services Flow Language (WSFL) is a new proposed specification to describe higher-level processes that involve multiple Web Services.



Visual 11-23 Development of Web Services

The activities for development of Java Web Services applications and clients are:

1. Develop the provider application (the Web Service).
2. Create the WSDL for the Web Service (1 and 2 can also be done in reverse sequence).
3. Deploy the Web Service to an application server.
4. Publish the Web Service to a UDDI Registry so that a client can find the Web Service.
5. Client retrieves WSDL file and generates a SOAP client proxy object for the client application (Application Developer tooling). Client develops the client application.
6. Client application invokes the Web Service where it is installed:
   - The application invokes the proxy.
   - The proxy invokes the Web Service through the SOAP server.

Static and Dynamic Web Services

Static Web Service

- Requester calls fixed provider
- Get WSDL file through e-mail, FTP, UDDI Registry
- Client application calls the provider Web Service

Dynamic Web Service

- Provider not known in advance
- Requester client
  - **Interacts with the UDDI Registry through the API**
  - **Dynamically retrieves service types from registry**
  - **Finds providers that implement the service type**
  - **Decides which ones to call**
  - **Calls the provider Web Service**

Visual 11-24 Static and Dynamic Web Services

When a requestor knows the provider and the Web Service, then we talk about a static Web Service. The client gets the WSDL file with the specification of the Web Service from the provider, and implements the client application.

When the provider is not known in advance, we call it a dynamic Web Service. The client application interrogates the UDDI Registry to find providers that implement a specific Web Service, and then calls all (or selected) requestors.

Web Services and Security

SOAP uses HTTP port 80

- Right through the firewall
  - **Must address security with other means**

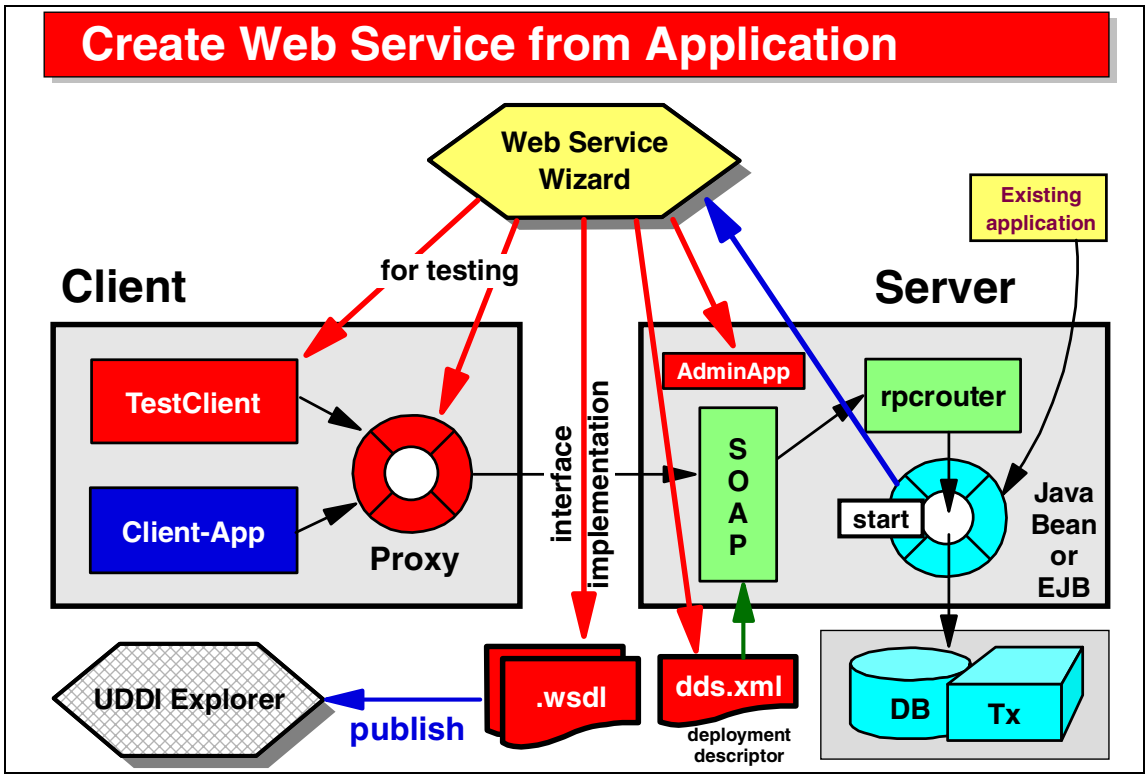
Security options

- HTTPS ==> **identification** and **authentication**
  - **Who are you and is your identity true**
- W3C digital signatures ==> **integrity**
  - **Is the data you sent the same data I received**
- W3C encryption ==> **privacy**
  - **Nobody can read the data you sent me**
- WebSphere/LDAP ==> **authorization**
  - **Are you allowed to perform this transaction**
- HTTPR protocol ==> **non-repudiation**
  (protocol enhancement proposed by IBM)
  - **Reliable one-time delivery of a message**

Visual 11-25 Web Services and Security

Security has not been standardized for Web Services.

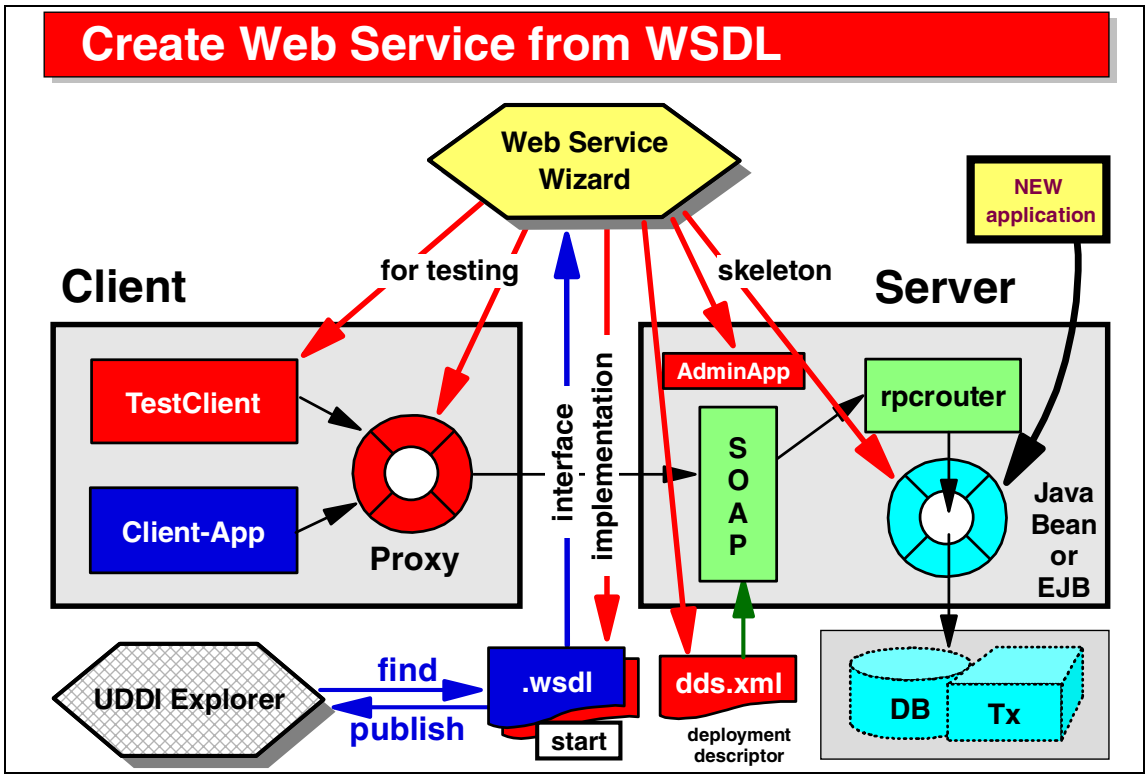
However, all kind of security concerns can be resolved and implemented using currently available techniques.



Visual 11-26 Create Web Service from Application

The Application Developer provides a Web Service wizard to generate a number of components from a given provider Web Service application. The Web Service is usually implemented by a JavaBean or EJB. This is the input to the wizard. The generated components are:

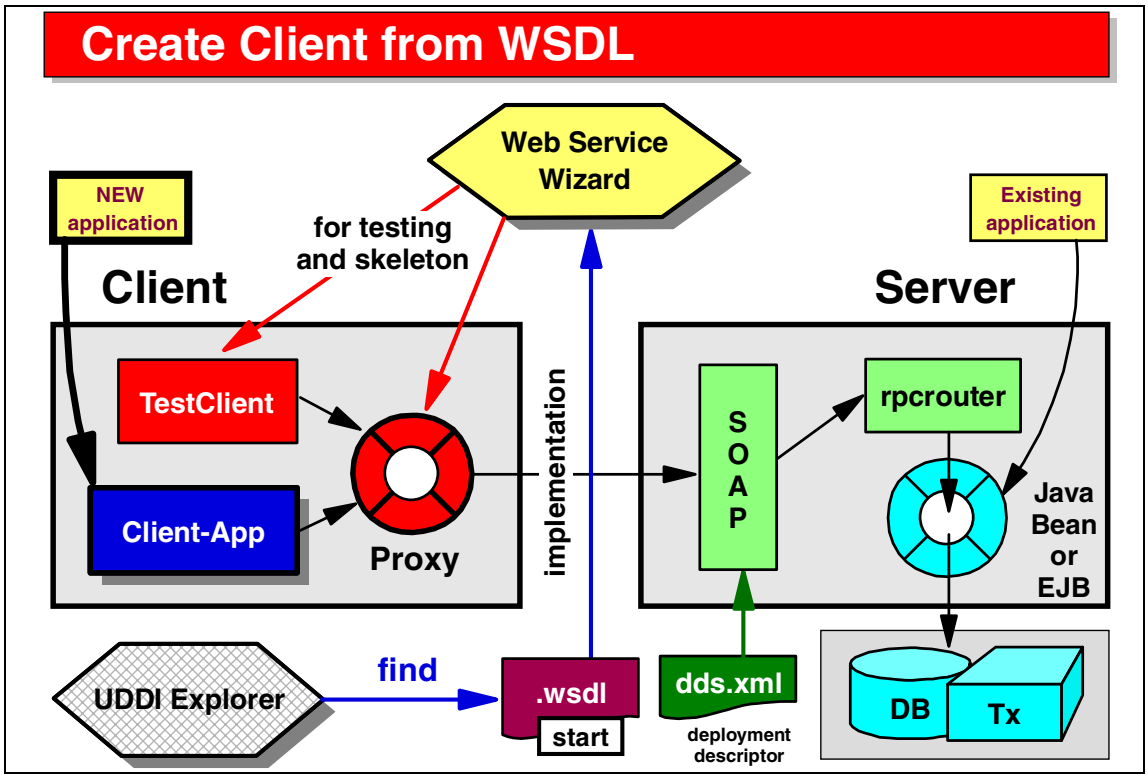
- WSDL files (interface and implementation
- The SOAP deployment descriptor (dds.xml)
- An administrative application to list, stop, and start Web Services in the application server
- A proxy class for client programming
- A test client (Web application) that uses the proxy to invoke the Web Service
- Optionally the WSDL files can be published to the UDDI Registry



Visual 11-27 Create Web Service from WSDL

The wizard can take a WSDL interface (or implementation) file as input. This is useful when a Web Service specification is known and the Web Service must be implemented.

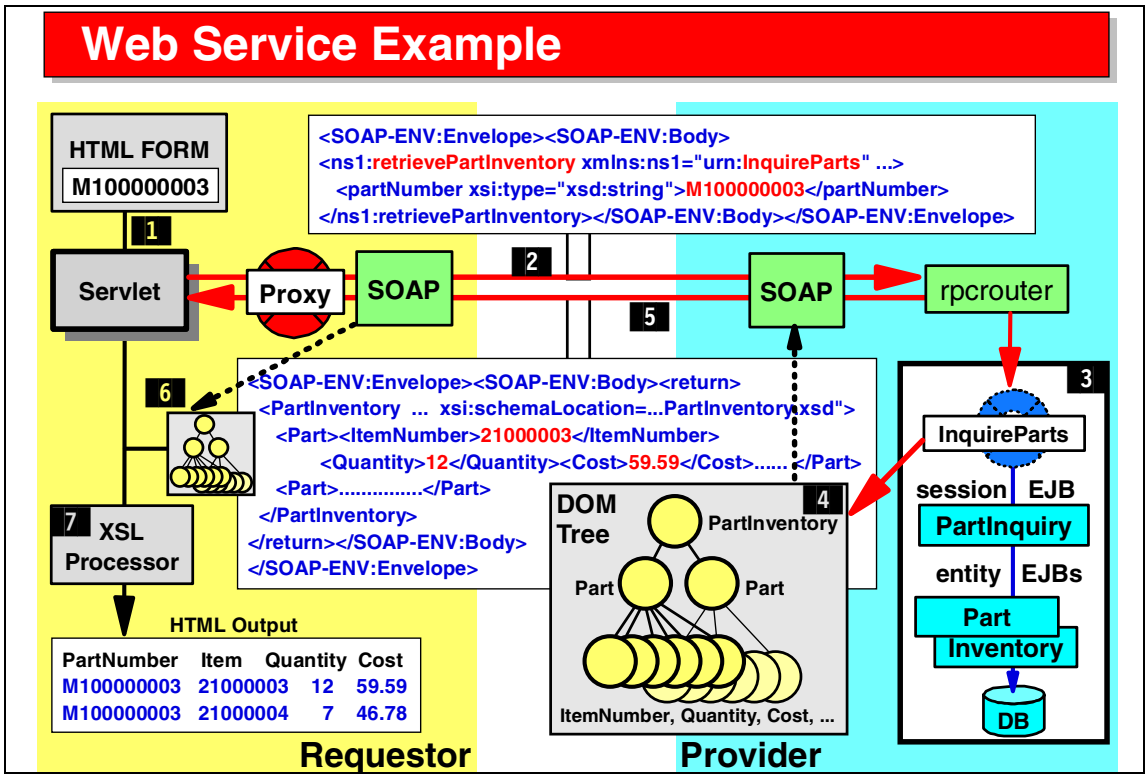
The same components are generated. In addition, a skeleton JavaBean for the server is generated. This skeleton bean can be used as the starting point to implement the Web Service.



Visual 11-28 Create Client from WSDL

To create a client for a Web Service, the wizard is run with a WSDL implementation file as input to generate the starting skeleton code for a client application.

The proxy bean and the test client are generated, then a real client can be implemented.



Visual 11-29 Web Service Example

The diagram shows an example of a Web application client calling a Web Service implementation:

1. An HTML form with a part number as input is used to invoke a servlet.
2. The servlet calls the proxy bean, and a SOAP XML message with the part number is sent to the server.
3. The Web Service is implemented in a JavaBean that invokes a session EJB that uses entity EJBs for database access.
4. The result of the Web Service is an XML tree in memory.
5. The SOAP server converts the result into an XML SOAP message (in the client, the XML tree in memory is rebuilt).
6. The servlet gets the result, the XML tree in memory.
7. The servlet calls an XSL processor to convert the XML into an HTML table that is displayed in a browser.

More Information

Lots of information on the Internet

IBM sites:

- developerWorks
  `http://www.ibm.com/developerworks/webservices/library/w-wsdl.html`
- alphaWorks
  `http://www.alphaworks.com/tech/wsde/webservicestoolkit`
- WebSphere Developer Domain
  `http://www.ibm.com/websphere/developer`
- VisualAge Developer Domain
  `http://www.ibm.com/software/vadd`
- UDDI
  `http://www.ibm.com/services/uddi`

Visual 11-30 More Information

There are a number of IBM and non-IBM sites on the Internet that have information about Web Services.

Summary

Web Services are based on

- SOAP
  - **Protocol for RPC-like invocation of Web Services**
  - **Vehicle for transport of data (parameters, results)**
  - **XML-based**
- WSDL
  - **Description language for Web Services**
- UDDI
  - **Registry for publication of Web Services**
  - **API to access and find Web Services**

Application Developer provides tools to generate

- WSDL files
- Java proxy beans for clients
- SOAP deployment descriptors
- Skeleton client and server applications

Visual 11-31 Summary

Web Services are the next wave of applications on the Internet and in intranet solutions.

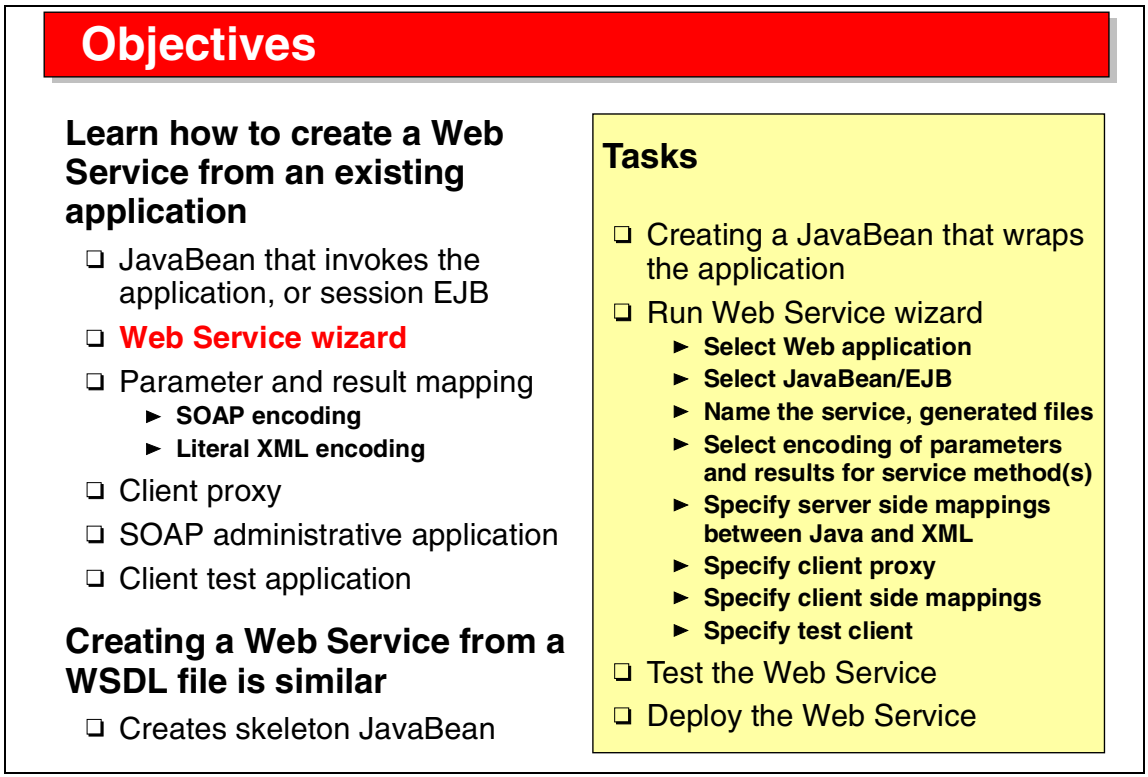
Web Services are based on three standards: SOAP, WSDL, and UDDI.

The Application Developer provides a Web Service wizard that can be used to generated the components for provider and requestor coding for Web Services.

12

Creating Web Services

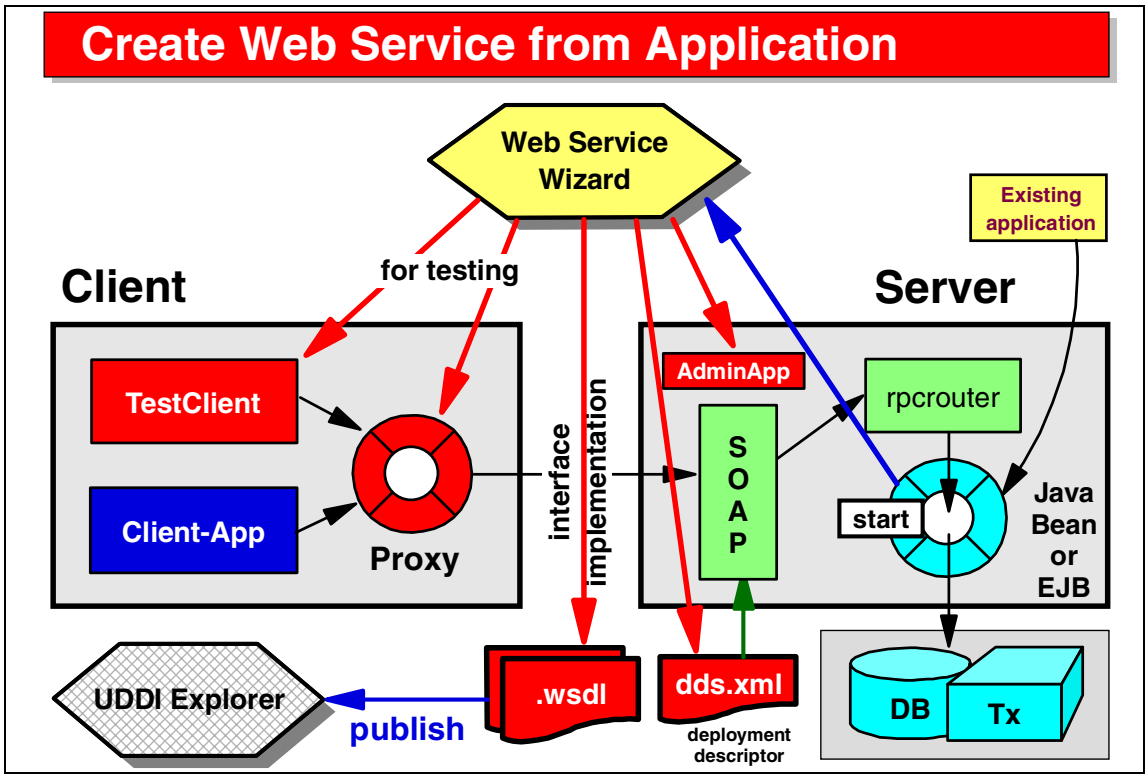
Visual 12-1 Title



Visual 12-2 Objectives

The objectives of this unit are to:

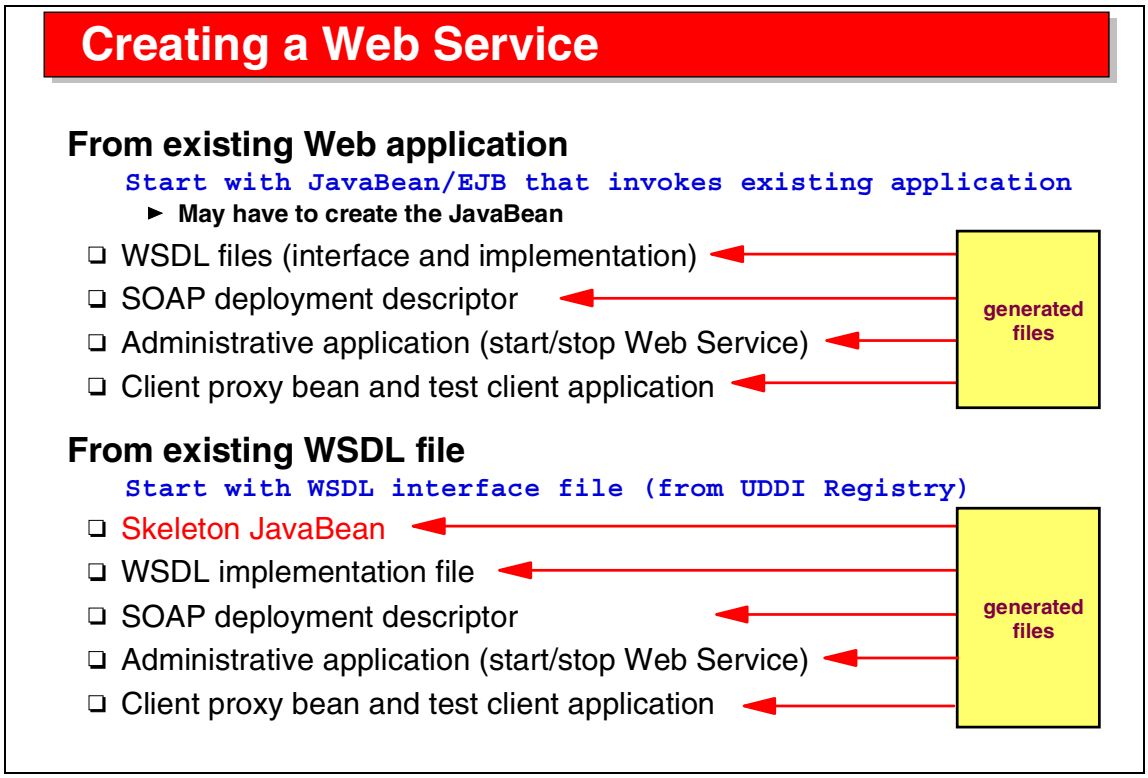
- Understand the functionality of the Web Service wizard of the Application Developer
- Understand SOAP encoding of parameters and results
- Understand the code that is generated by the wizard



Visual 12-3 Create Web Service from Application

The Application Developer provides a Web Service wizard to generate a number of components from a given provider Web Service application. The Web Service is usually implemented by a JavaBean or EJB. This is the input to the wizard. The generated components are:

- WSDL files (interface and implementation
- The SOAP deployment descriptor (dds.xml)
- An administrative application to list, stop, and start Web Services in the application server
- A proxy class for client programming
- A test client (Web application) that uses the proxy to invoke the Web Service
- Optionally the WSDL files can be published to the UDDI Registry

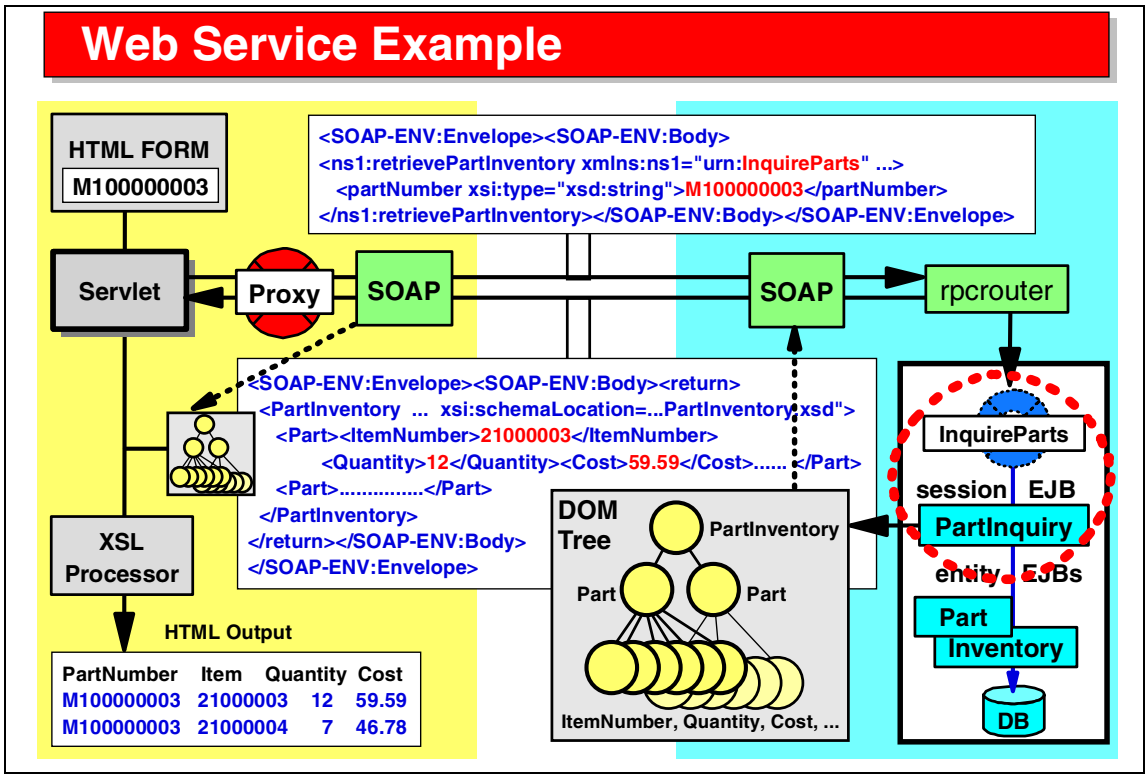


Visual 12-4 Creating a Web Service

The input to the wizard when creating a Web Service is either an existing application (JavaBean or EJB), or a WSDL file.

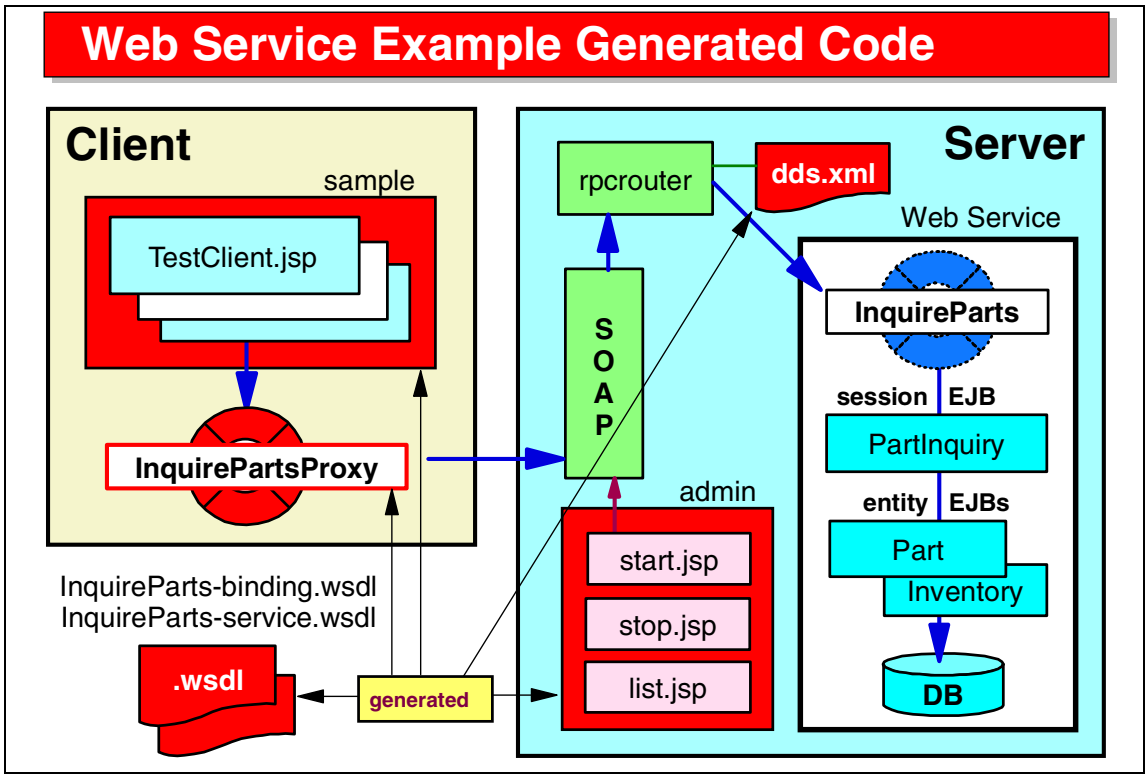
The generated code is almost the same:

- When starting from an application, the WSDL files are generated.
- When starting from a WSDL file, a skeleton JavaBean is generated.



Visual 12-5 Web Service Example

In the Web Service example, described in Visual 11-29 on page 247, we now look in detail at the provider side, where the Web Service is created from an existing application.

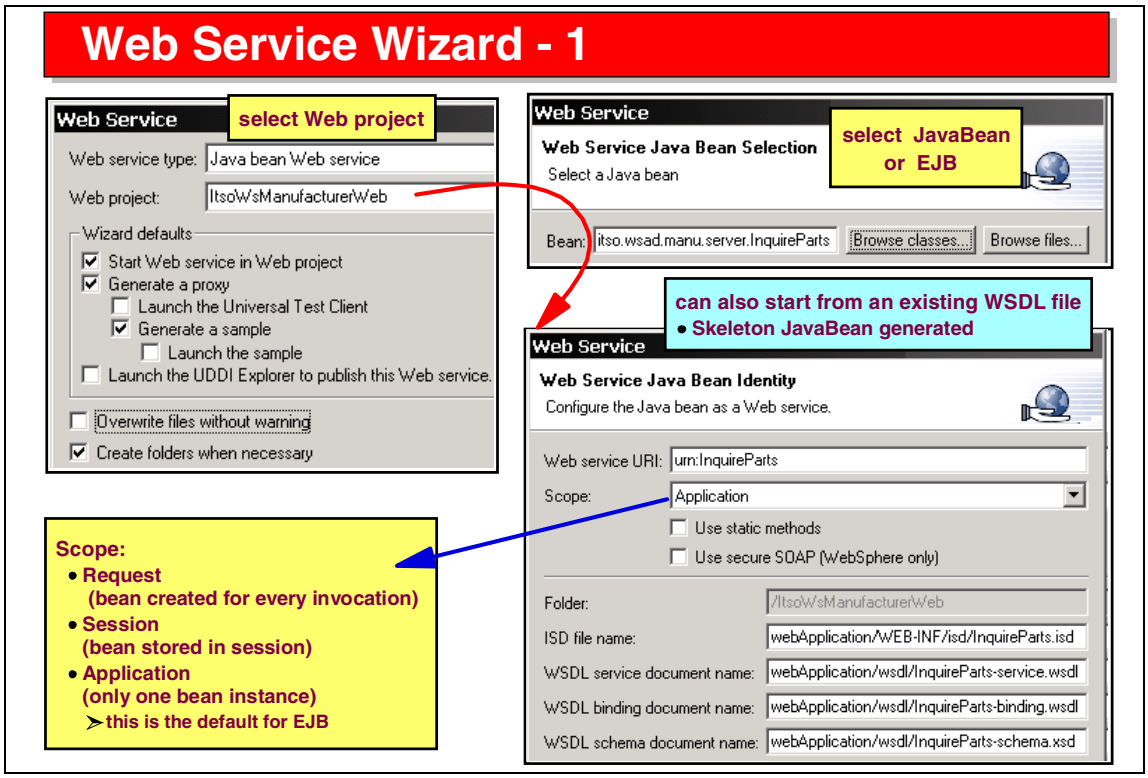


Visual 12-6 Web Service Example Generated Code

We start with a JavaBean (InquireParts) that uses a session EJB to retrieve inventory information for a given part number.

The Web Service wizard generates:

- The SOAP deployment descriptor (dds.xml)
- The administrative Web application with HTML and JSP files
- The WSDL files (interface and implementation)
- The proxy JavaBean (InquirePartsProxy)
- The sample test application (TestClient.jsp, and so forth)



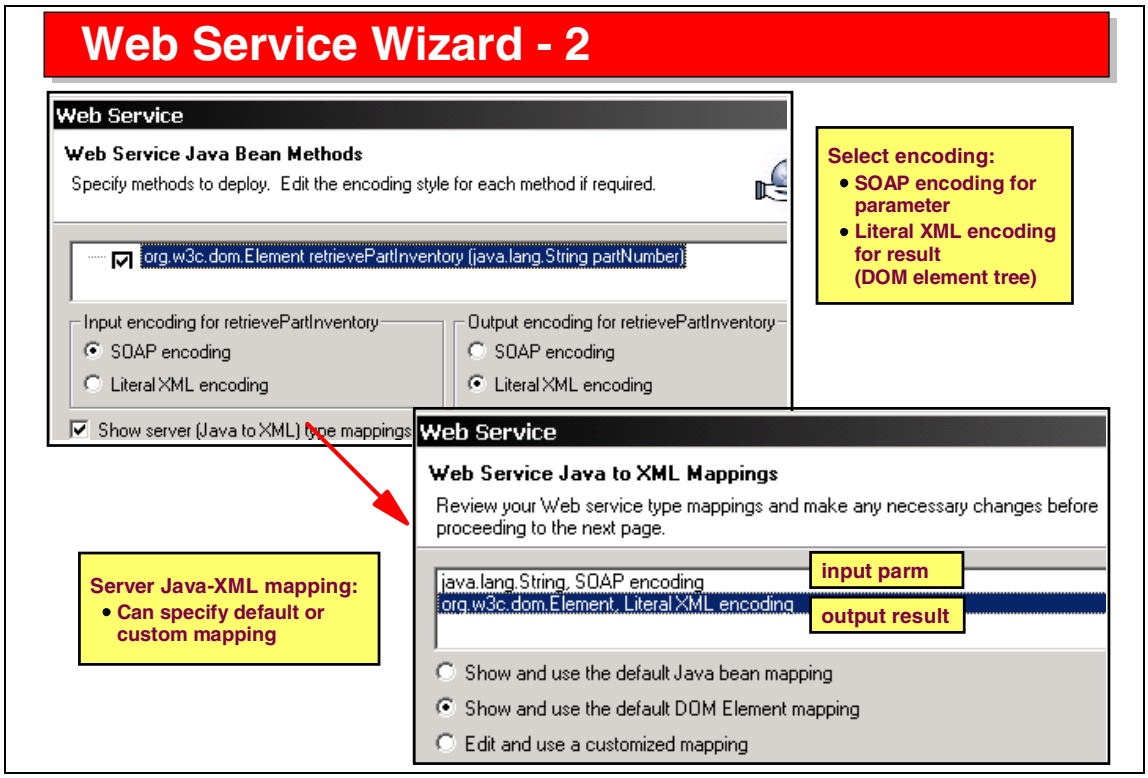
Visual 12-7 Web Service Wizard - 1

The Web Service wizard is a series of dialog panels that guide you through the process:

- Select the Web project for the generated code
- Select the type of Web Service (from JavaBean, EJB, WSDL file)
- Select the JavaBean (or EJB)
- Name the service and the generated output files
- Continue working through the panels

One of the important choices is the scope:

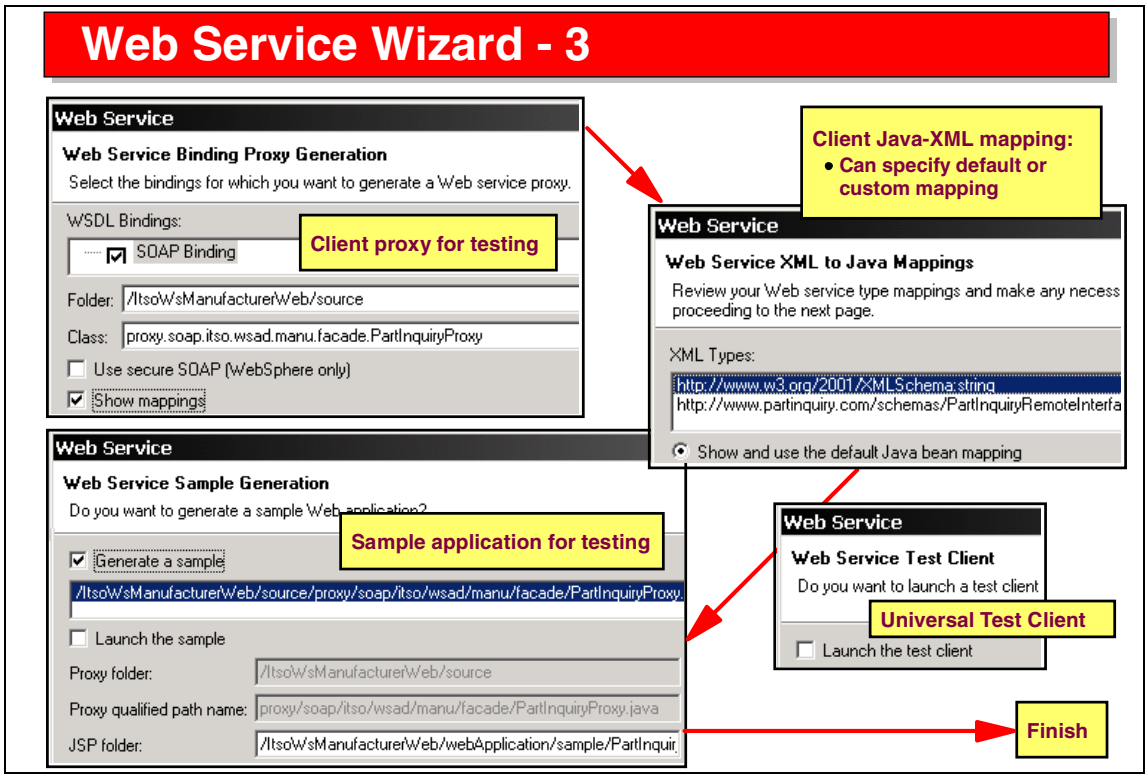
- Request—a new JavaBean is created for each client request
- Session—the JavaBean is stored in a user session for repetitive use
- Application—only one bean instance exists and all requests are using it



Visual 12-8 Web Service Wizard - 2

Web Service wizard (continued):

- Select the method to be invoked and the encoding of the parameters and result
- Check the detailed JavaBean-to-XML mappings (the result of the service is translated into XML for transmitting to the client)

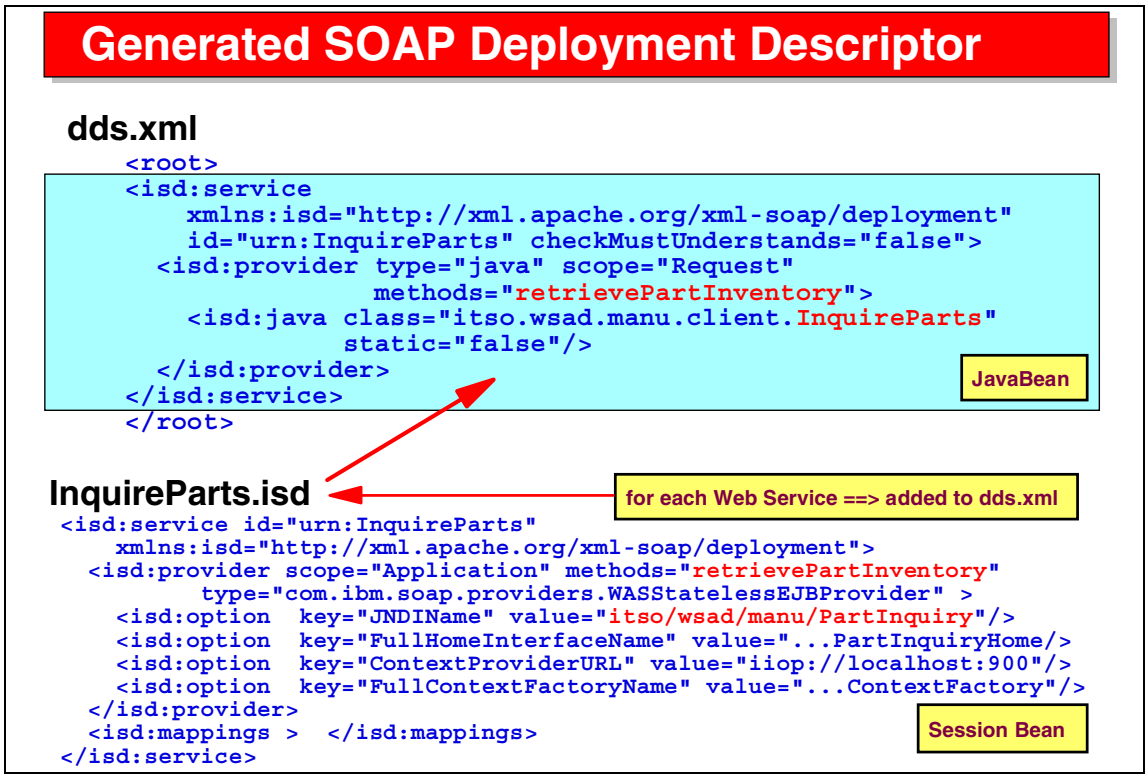


Visual 12-9 Web Service Wizard - 3

Web Service wizard (continued):

- Generation and name of the proxy bean
- XML-to-Java mappings (on the client side, the XML is translated back into Java objects)
- Should the universal test client be started
- Should a sample test client program be generated

At the end of the wizard, the code is generated and installed into the test server associated with the Web project, and the server is started so that the Web Service can be immediately tested.

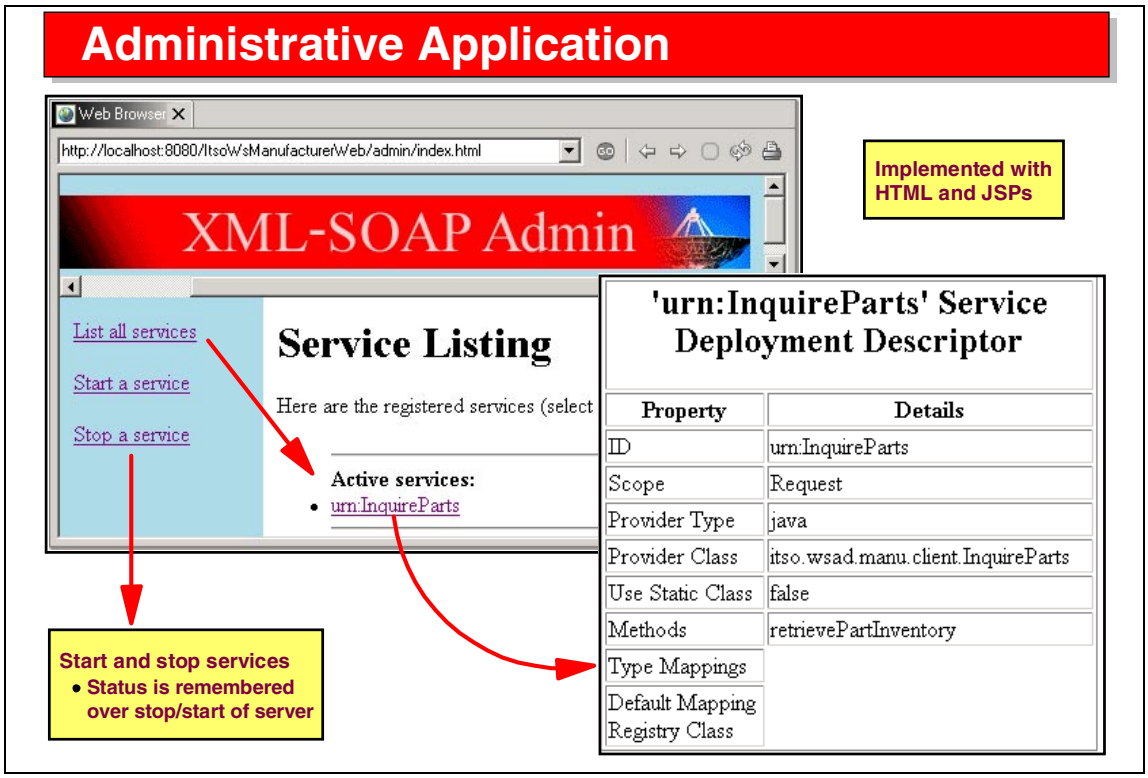


Visual 12-10 Generated SOAP Deployment Descriptor

For the SOAP deployment descriptor (dds.xml), an ISD file is created for each Web Service and then added to the deployment descriptor.

We show here two examples:

- The top shows the ISD file generate for a JavaBean.
- The bottom shows the ISD file generated for a session EJB.

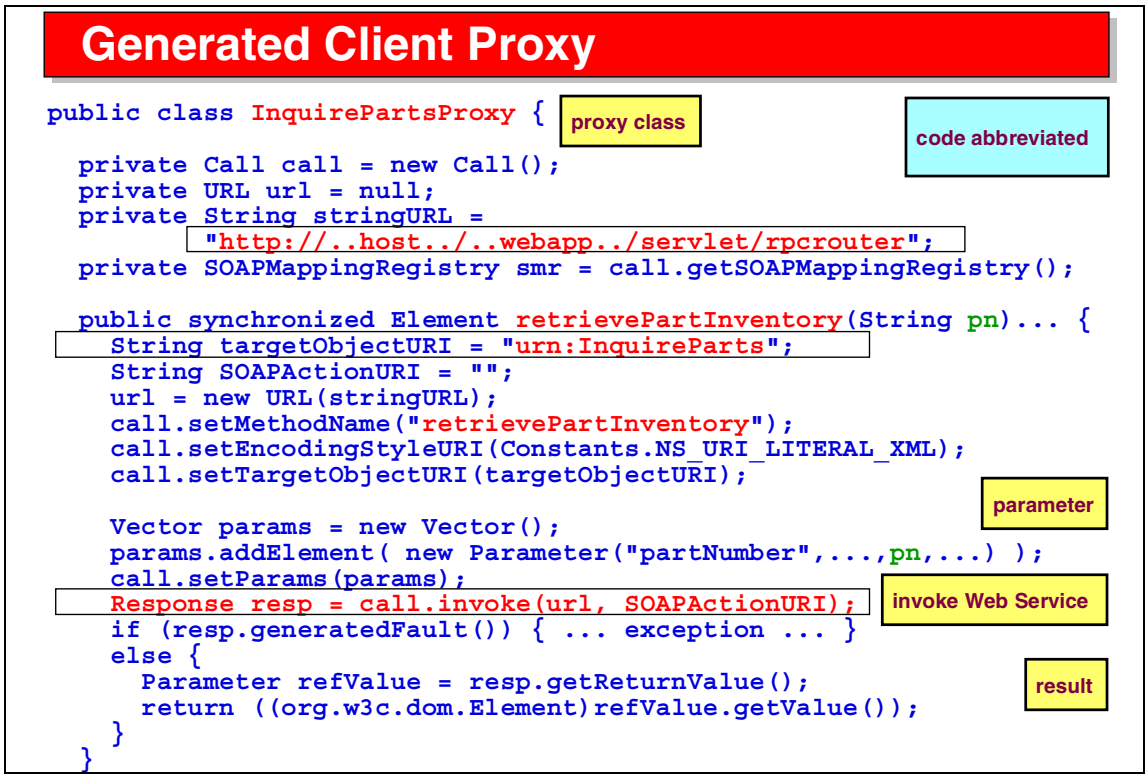


Visual 12-11 Administrative Application

The administrative application is composed of:

- HTML files, such as an index.hmtl file
- A number of JSPs to list the Web Services, display the properties of a Web Service, stop a selected Web Service, and start a selected Web Service

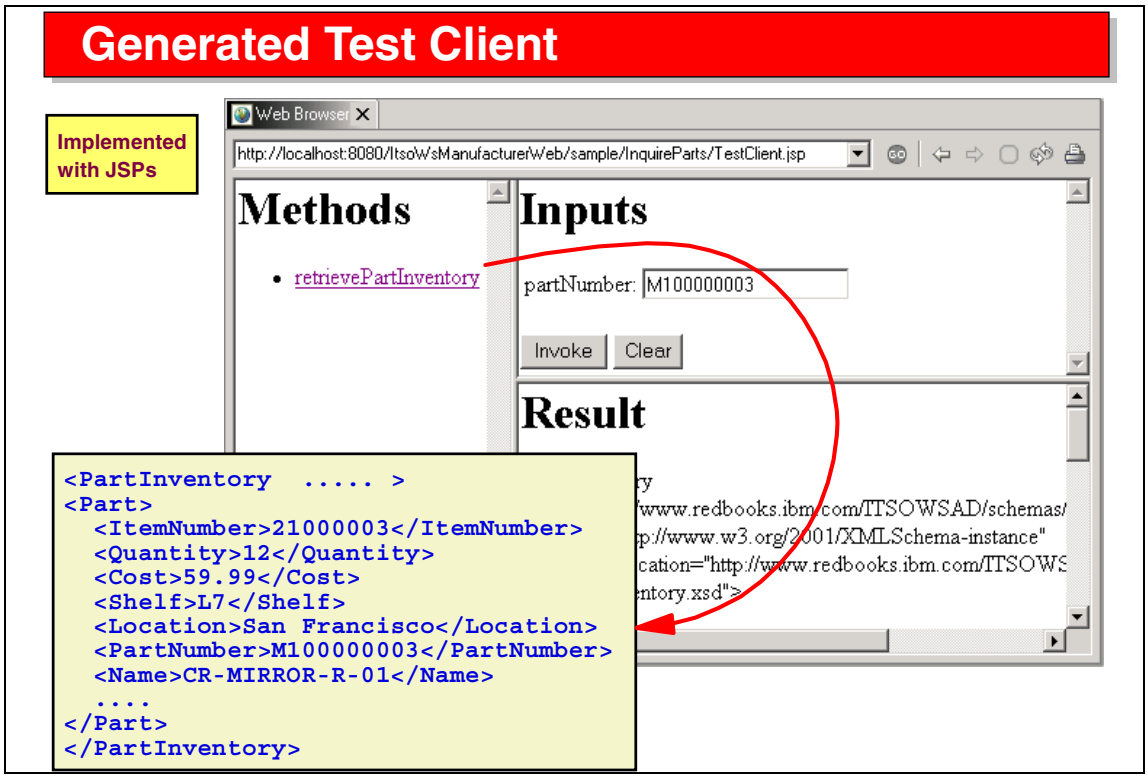
By default, all Web Services are started. If a Web Service is stopped, this is remembered and at the next start of the server the Web Service is still stopped.



Visual 12-12 Generated Client Proxy

This is an example of a generated client proxy:

- The client proxy allocates a SOAP Call object and initializes it with the target servlet (rpcrouter).
- The proxy contains a method that the client can call to invoke the Web Service.
- In this method, the target Web Service and method within the service are set in the Call object, parameters are stored in a Vector that is also set in the Call object, and the Web Service is invoked.
- The result object is extracted from the response object of the Web Service call.



Visual 12-13 Generated Test Client

The sample test client is composed of a set of JSPs, starting with the TestClient.jsp.

When running this JSP, you select a method, enter parameters for the method, and invoke the Web Service. The results of the Web Service are displayed after conversion to XML.

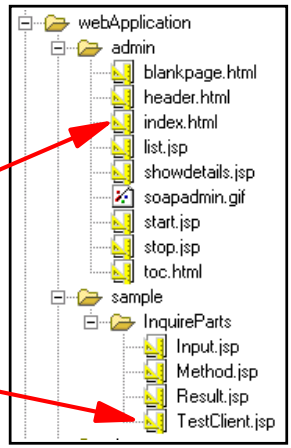
Testing the new Web Service

SOAP runtime/configuration placed into Web application

- **soap.xml**

```
<soapServer>
  <!-- This ConfigManager looks for a dds.xml file ... -->
  <configManager value="com.ibm.soap.server.XMLDrivenConfigManager"/>
</soapServer>
```

- **dds.xml** <== generated deployment descriptor
- **soapcfg.jar** <=== added to Web application lib



Testing

- Start the server for the Web application
- Run **admin/index.html** for SOAP administration application
- Run **sample/InquireParts/TestClient.jsp** for sample test application

Visual 12-14 Testing the new Web Service

To test the Web Service that is created, a number of components are required:

- soap.xml—points to a configuration manager that reads the deployment descriptor
- dds.xml—the SOAP deployment descriptor
- soapcfg.jar—contains required SOAP code
- admin/index.html—starting point of the administrative application
- sample/TestClient.jsp—sample test client

All these components are added to the Web application.

Deployment to WebSphere

The Web Service is in a Web application (Web project)

- Web project is part of EAR project
- Web application may use EJBs (EJB project in same EAR project)

Deploy the EAR file to WebSphere AE or AEs

- Make sure the EJB mapping to JNDI names is complete
- Make sure the Web application uses the correct EJB JNDI names
- Change the Web Service proxy bean

```
http://localhost:8080/ItsoWsManufacturerWeb/servlet/rpcrouter
http://www.realhost.com/ ...
```

- Check WSDL file
- Export EAR file
- Install EAR file (command line or Administrative Console)

```
seappinstall -install itsowsmanufacturer.ear
  -expandDir d:\websphere\appserver\installedApps\
                                    itsowsmanufacturer.ear
  -ejbDeploy false -interactive false
```

Visual 12-15 Deployment to WebSphere

The steps to deploy a Web Service to a real WebSphere Application Server are as follows:

- Make sure all the specifications are complete (for example, JNDI names of EJBs).
- Change the proxy bean to point to the real installed Web application.
- Check the generated WSDL file (make sure they point to the correct server as well).
- Export the EAR file that contains the Web project with the Web Service.
- Install the EAR file into the application server using the administrative facility or the batch command.

Summary

Web Service wizard creates a Web Service

- From a JavaBean (turn existing application into a Web Service)
- From a WSDL file (for a new Web Service)

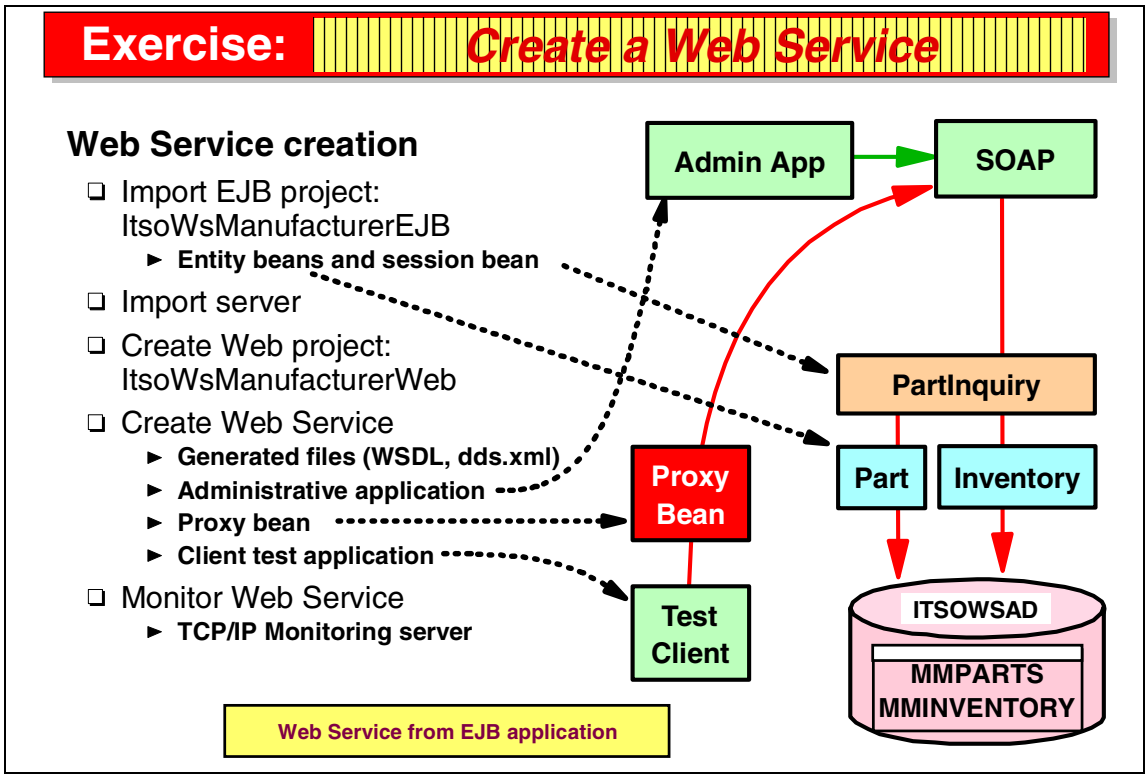
Web Service wizard generates

- WSDL files from JavaBean
- Skeleton JavaBean from WSDL file
- SOAP deployment descriptor (dds.xml)
- Client proxy
  - **Makes client programming easy**
- SOAP administration application
  - **List, start, stop of Web Services**
- Client test application
  - **Test the Web Service and the client proxy**

Deploy the Web Service using an EAR file

Visual 12-16 Summary

The Web Service wizard of the Application Developer generates all the code required to test and deploy a Web Service.



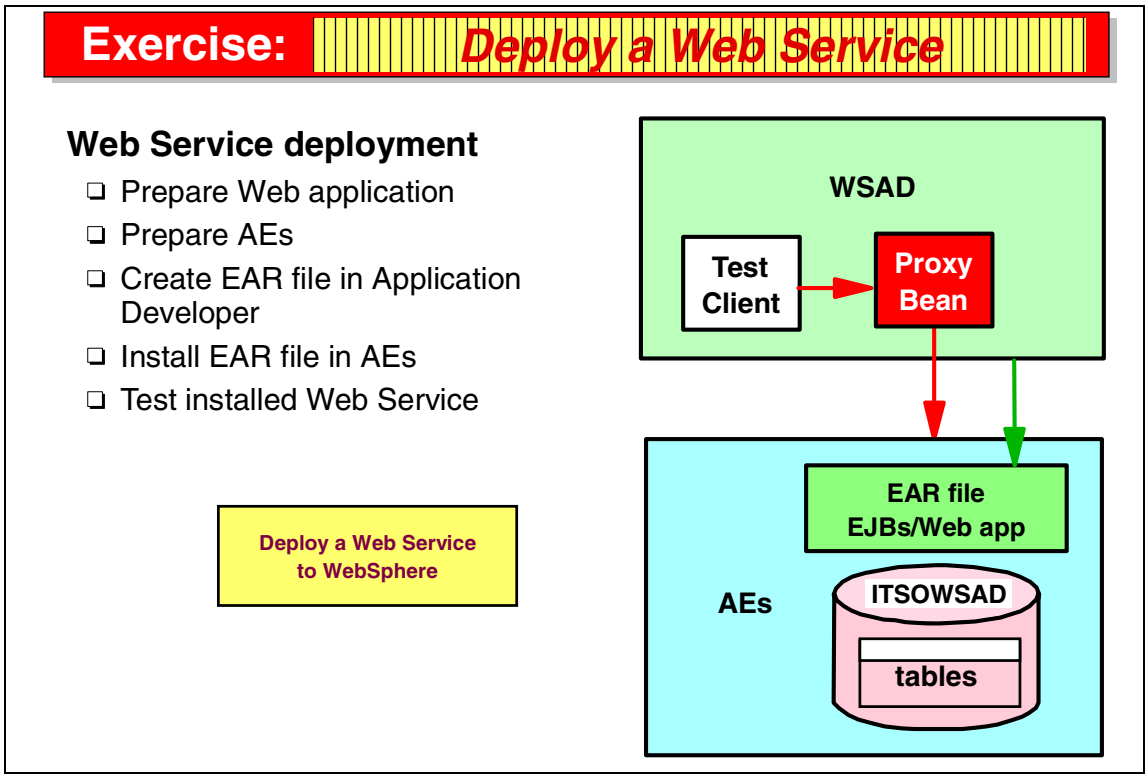
Visual 12-17 Exercise: Create a Web Service

The Web Service creation exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with an existing EJB-based application:

- Import the EJB application.
- Create a Web Service for this application.
- Test the Web Service.
- Use the TCP/IP Monitoring Server to see the HTTP traffic.

See Exercise 8, “Create a Web Service” on page 349 for the instructions for this exercise.



Visual 12-18 Exercise: Deploy a Web Service

The Web Service deployment exercise guides you through the deployment task discussed in the presentation.

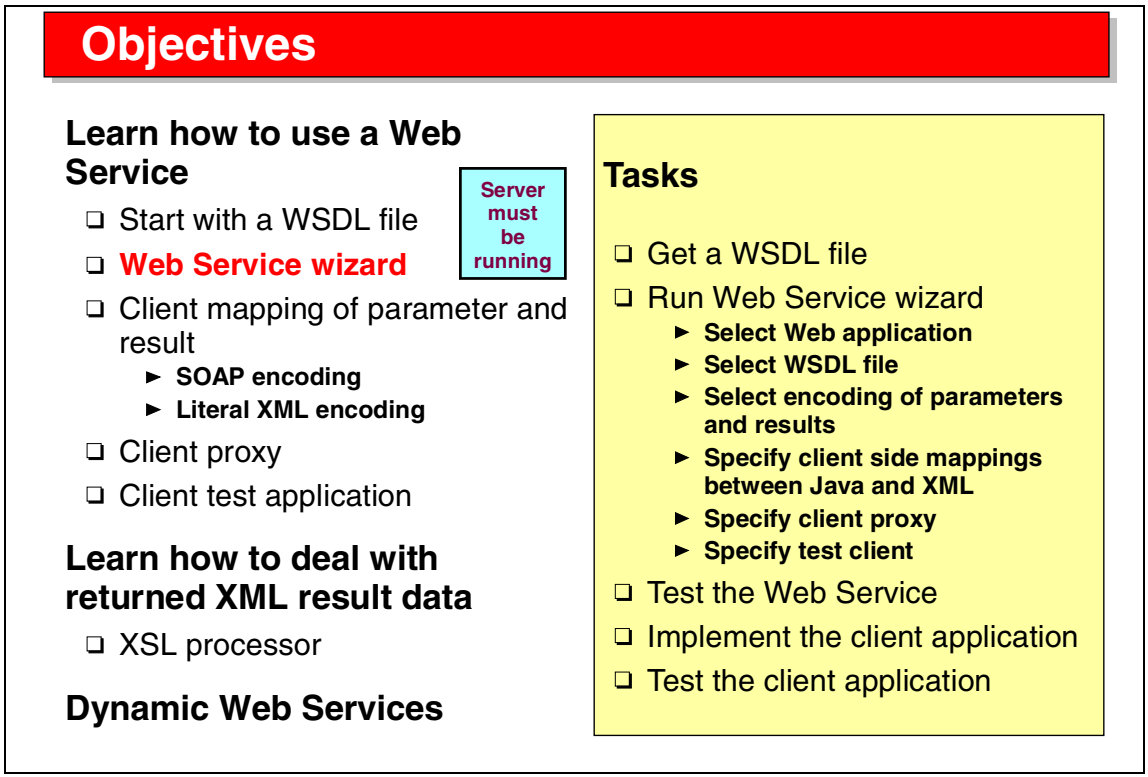
In this exercise you deploy the Web Service created in Exercise 8, “Create a Web Service” on page 349 to a WebSphere Application Server AEs.

See Exercise 9, “Deploy and test a Web Service” on page 361 for the instructions for this exercise.

13

Using Web Services

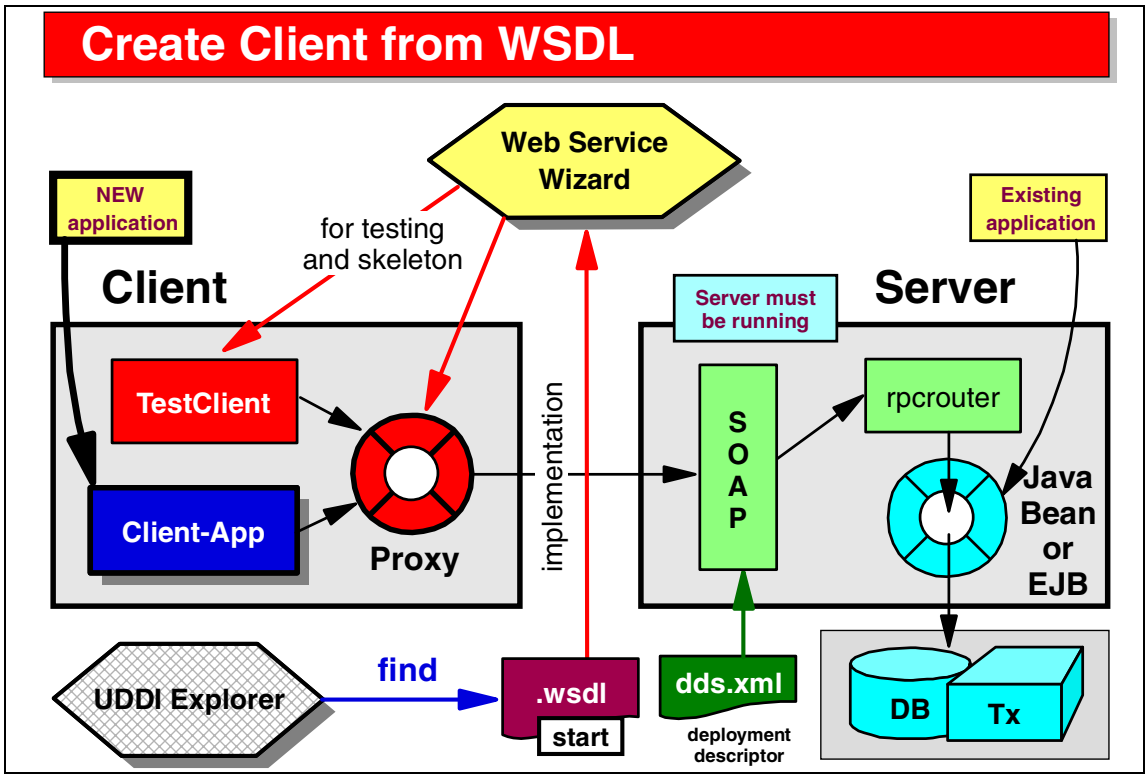
Visual 13-1 Title



Visual 13-2 Objectives

The objectives of this unit are to:

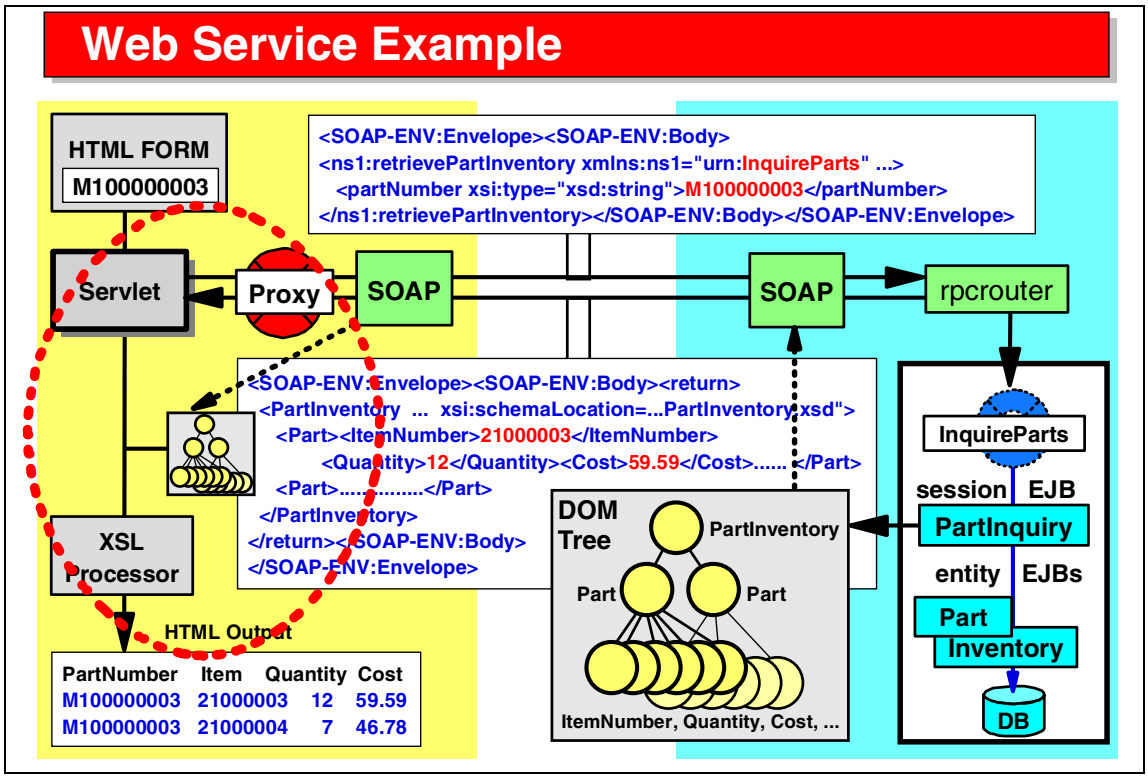
- Understand the Web Service wizard for the creation of a client application that invokes a Web Service
- Understand SOAP encoding of parameters and result
- Understand how the result of a Web Service can be translated into HTML for a browser user
- Understand dynamic Web Services where the client application interrogates the UDDI Registry to locate Web Service providers and invoke their Web Services



Visual 13-3 Create Client from WSDL

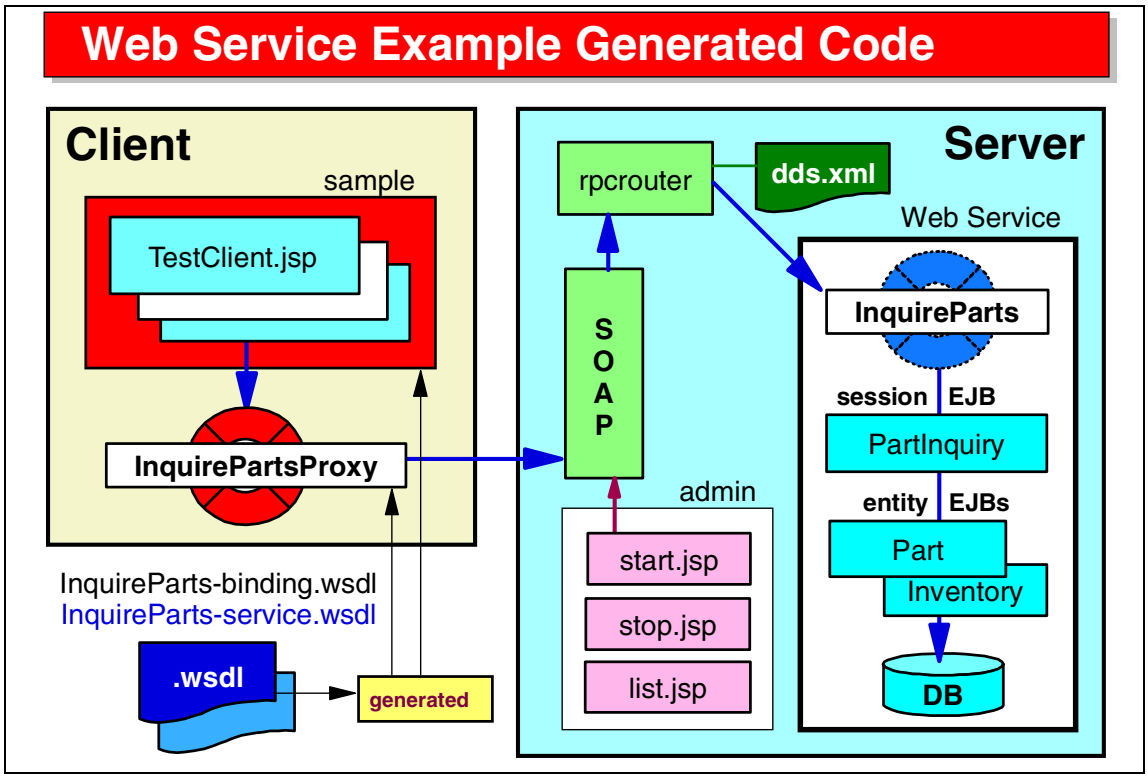
To create a client for a Web Service, the wizard is run with a WSDL implementation file as input to generate the starting skeleton code for a client application.

The proxy bean and the test client are generated, then a real client can be implemented.



Visual 13-4 Web Service Example

In the Web Service example, described in Visual 11-29 on page 247, we now look in detail at the requestor side, where the Web Service is invoked and the result is displayed in a browser.



Visual 13-5 Web Service Example Generated Code

To create a client application, the Web Service wizard is run for a WSDL implementation file as input. The wizard generates:

- The proxy bean that embeds the SOAP Call object and invokes the Web Service
- A sample test client (a set of JSPs) that can be used to test the Web Service before implementing the real client

The real client application can then be implemented. Some code generated in the test client may be useful for the real client.

Web Service Wizard

Create client proxy and sample test application

- Select Web project
- Select WSDL implementation (service) file
  - **Get WSDL file from UDDI Registry**
  - **Store WSDL file in Web project**
- Generate proxy bean
  - **Easiest way for client to connect to server using SOAP**
- Specify client side Java-XML mapping
- Generate sample test client
  - **Sample code shows how to set parameters and retrieve results**
  - **Sample method to write XML from DOM tree element**

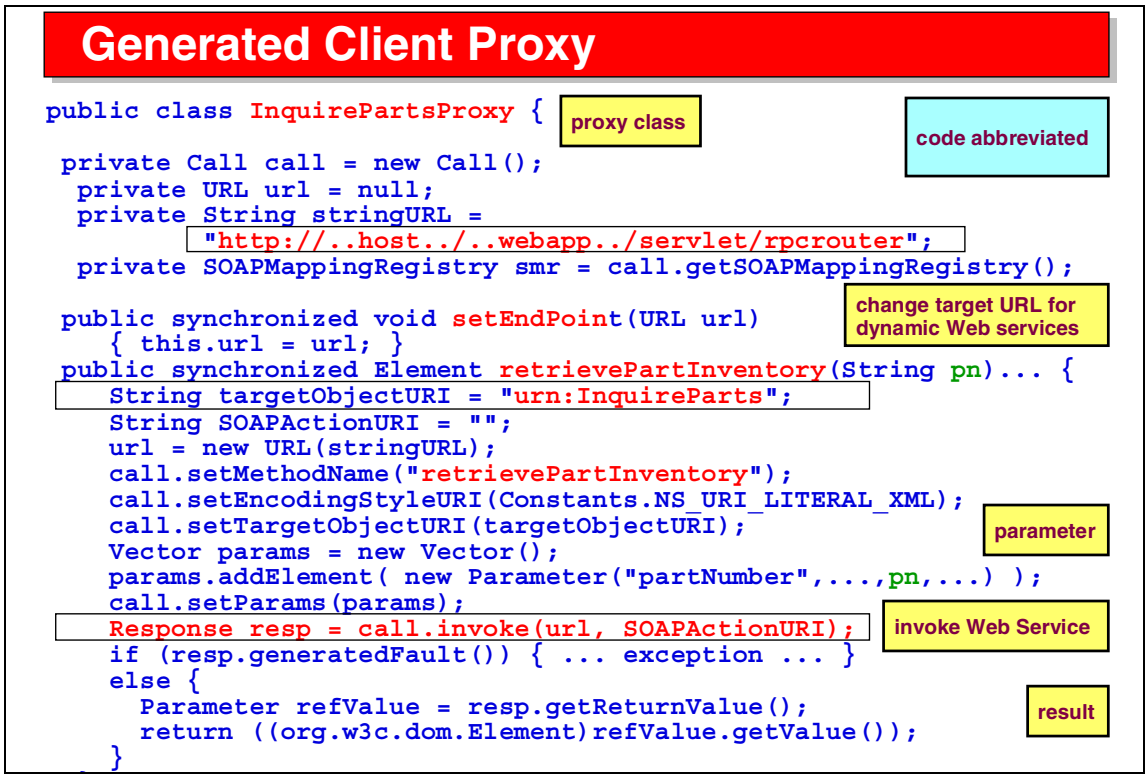
Use sample code for real client application

- Copy/paste fragments into application

Visual 13-6 Web Service Wizard

The Web Service wizard guide the user through a series of dialog panels, similar to the ones shown in Visual 12-7 on page 257 (and subsequent).

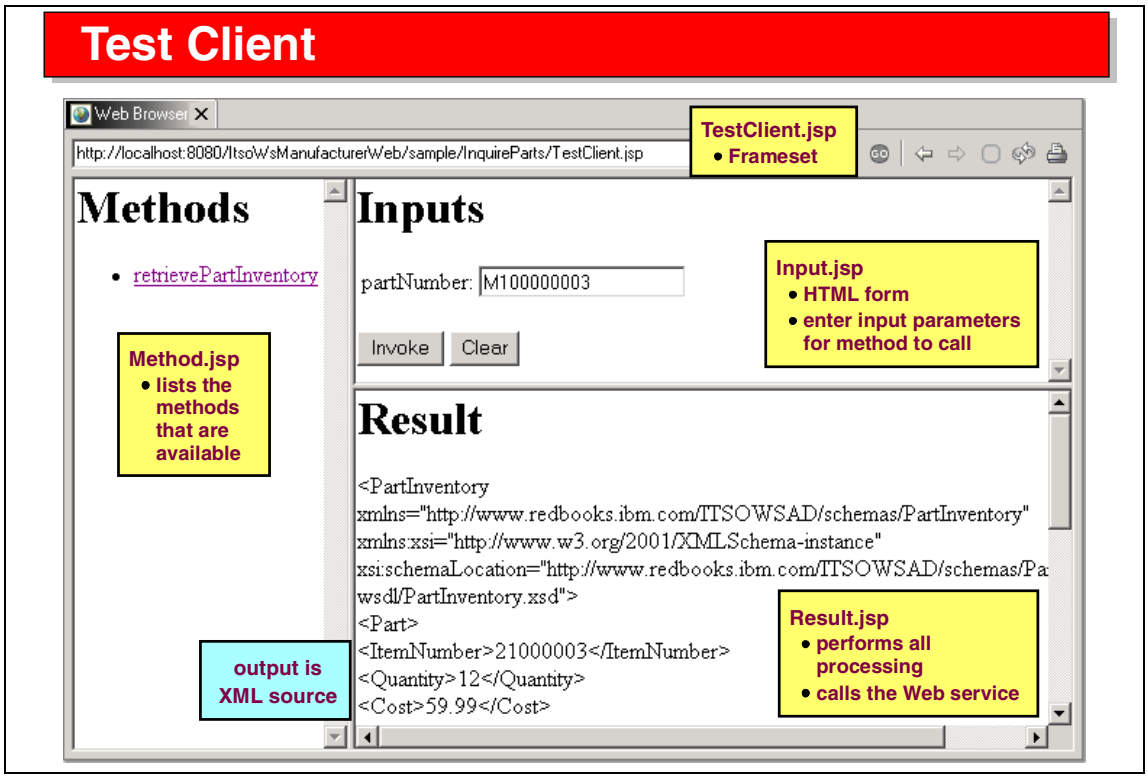
The major difference is that a WSDL implementation file is selected as input.



Visual 13-7 Generated Client Proxy

This is an example of a generated client proxy:

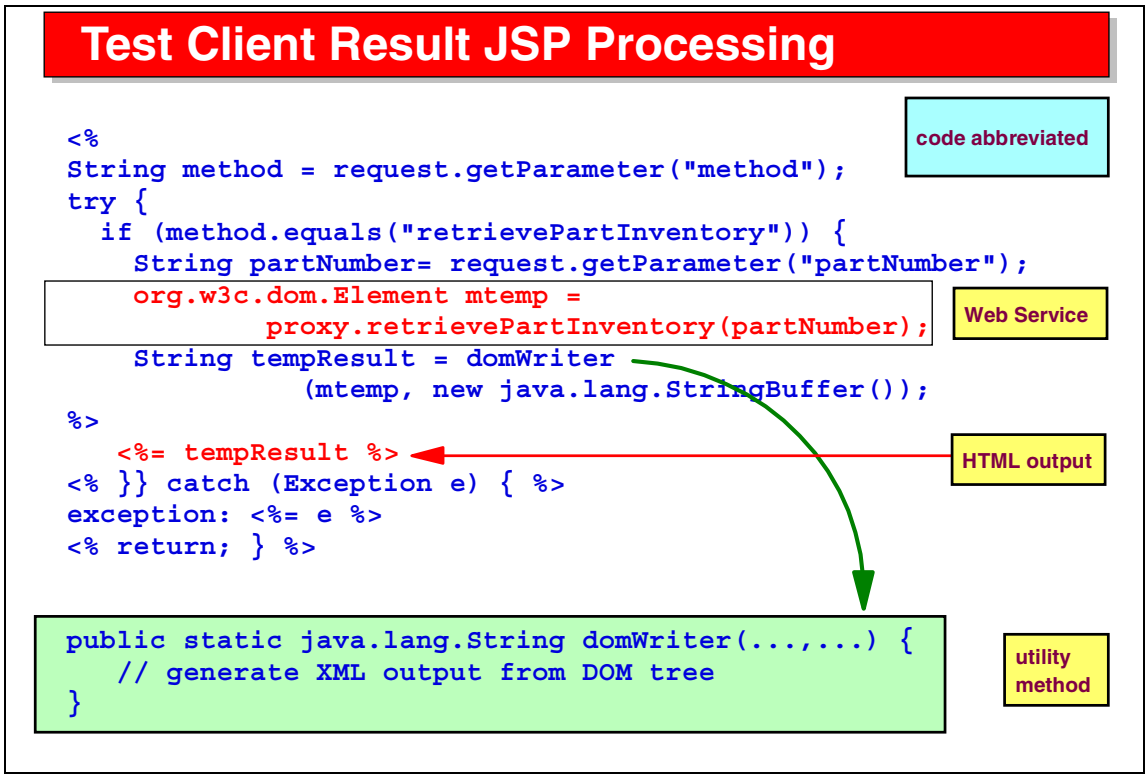
- The client proxy allocates a SOAP Call object and initializes it with the target servlet (rpcrouter).
- The proxy contains a method that the client can call to invoke the Web Service.
- In this method, the target Web Service and method within the service are set in the Call object, parameters are stored in a Vector that is also set in the Call object, and the Web Service is invoked.
- The result object is extracted from the response object of the Web Service call.



Visual 13-8 Test Client

The sample test client is composed of a set of JSPs, starting with the TestClient.jsp.

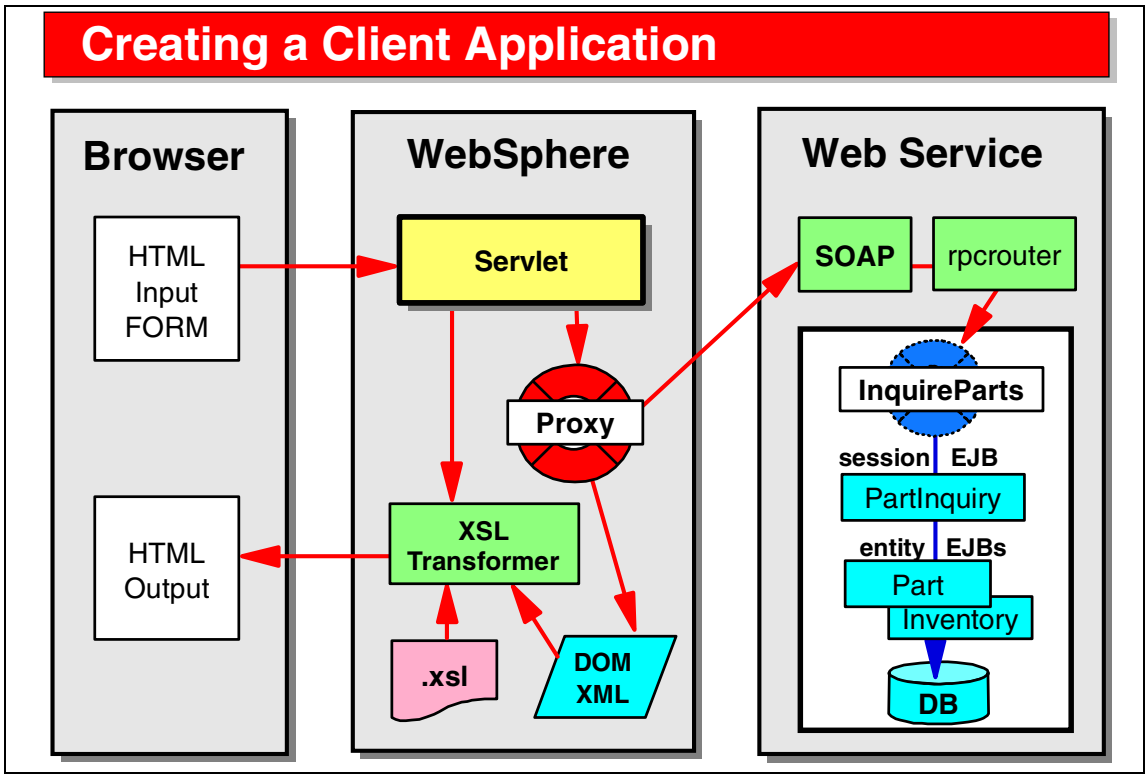
When running this JSP, you select a method, enter parameters for the method, and invoke the Web Service. The results of the Web Service are displayed after conversion to XML.



Visual 13-9 Test Client Result JSP Processing

The result JSP of the sample test client uses a domWriter method to convert the XML data object returned as the result of the Web Service call into a readable XML file.

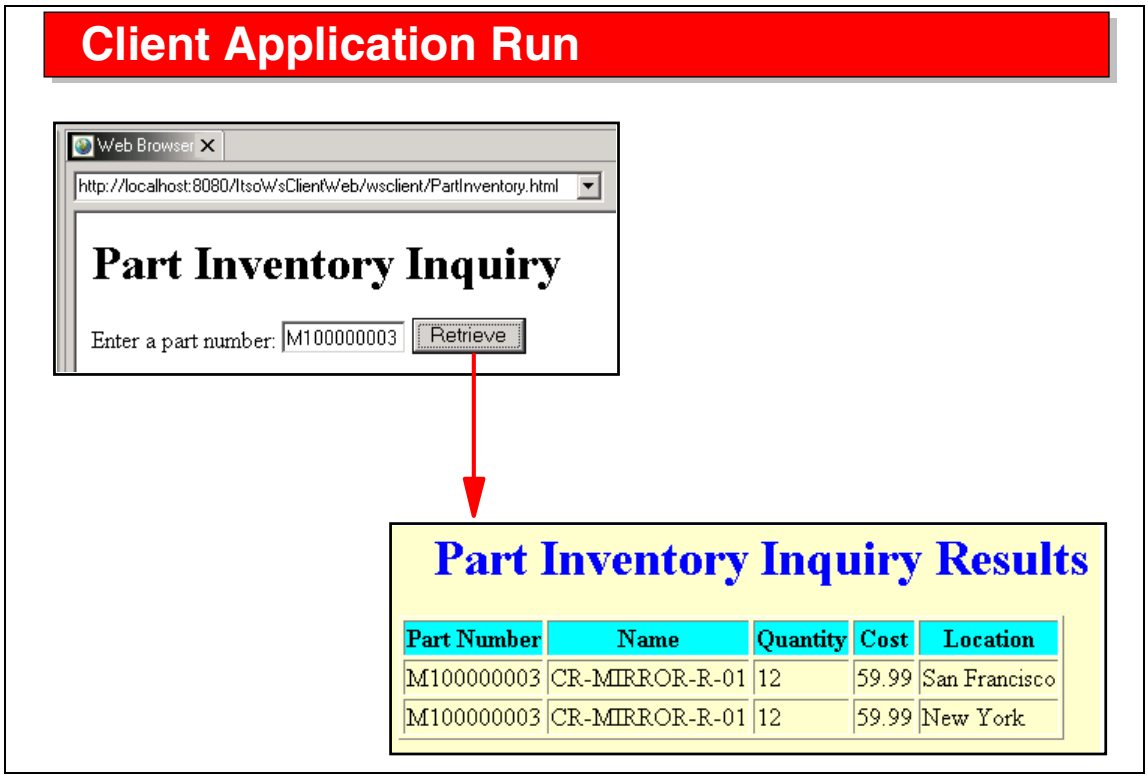
This utility method may be useful in a real client.



Visual 13-10 Creating a Client Application

In our example, the client is a Web application:

- A servlet is invoked from an HTML form.
- The servlet extracts the parameter (part number) and calls the proxy object to invoke the Web Service.
- The result of the Web Service is a XML DOM tree in memory.
- The servlet invokes the XSL transformer.
- The XSL transformer uses an XSL file to convert the XML tree into an HTML table.
- The HTML table is displayed in a browser.



Visual 13-11 Client Application Run

A sample run of the client application shows the HTML input form and the HTML result table.

Servlet Code with Proxy and XSL

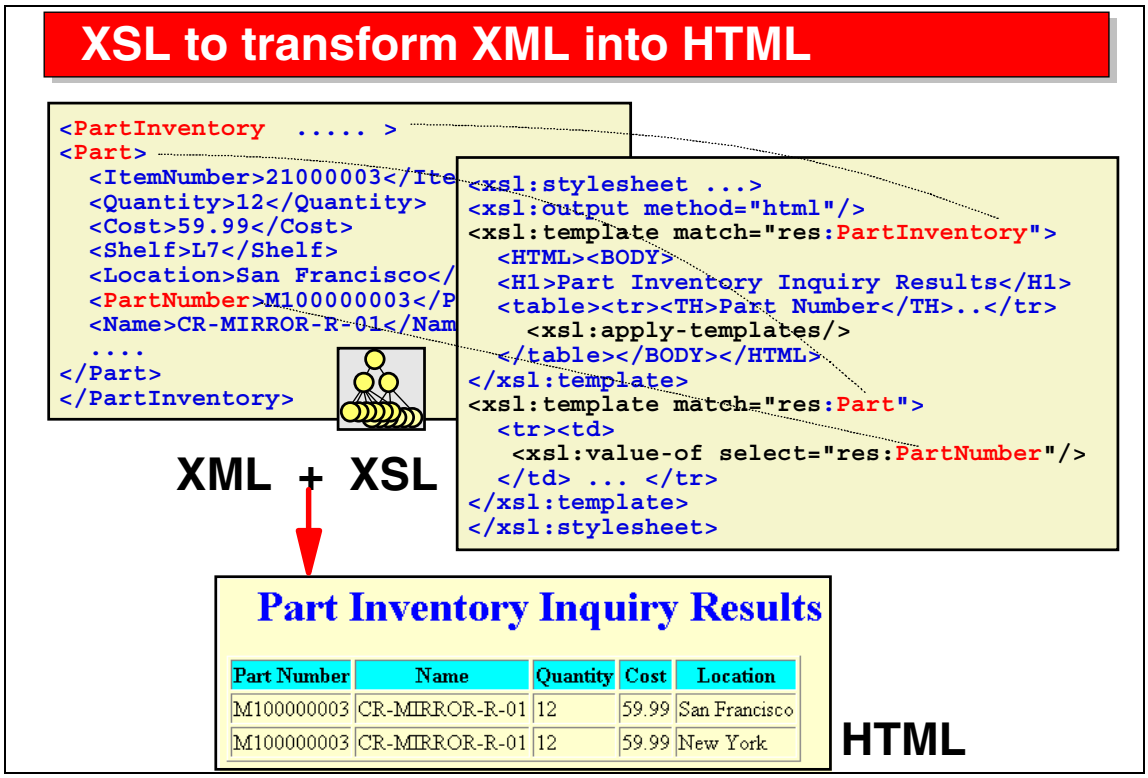
```
public class PartInventoryServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) {
    try {
      response.setContentType("text/html; charset=UTF-8");
      TransformerFactory tf =
                  TransformerFactory.newInstance();
      Source xslSource = new StreamSource( new
        URL("http://host/..webapp../xxx.xsl").openStream() );
      Transformer t = tf.newTransformer(xslSource);
      PrintWriter out = response.getWriter();
      InquirePartsProxy proxy = new InquirePartsProxy();
      String pn = (String)request.getParameter("partNumber");
      Element result = proxy.retrievePartInventory(pn);
      Source xmlSource = new DOMSource(result);
      t.transform(xmlSource, new StreamResult(out));
    } catch (Exception e) { e.printStackTrace();}
  }
```

call Web Service

Visual 13-12 Servlet Code with Proxy and XSL

The servlet of the sample application is quite simple:

- The XSL transformer is prepared using the XSL file as base.
- The Web Service is invoked using the part number parameter.
- The XML DOM tree is converted into an XML source object.
- The XSL transformer writes the output directly to the servlet's output stream.



Visual 13-13 XSL to transform XML into HTML

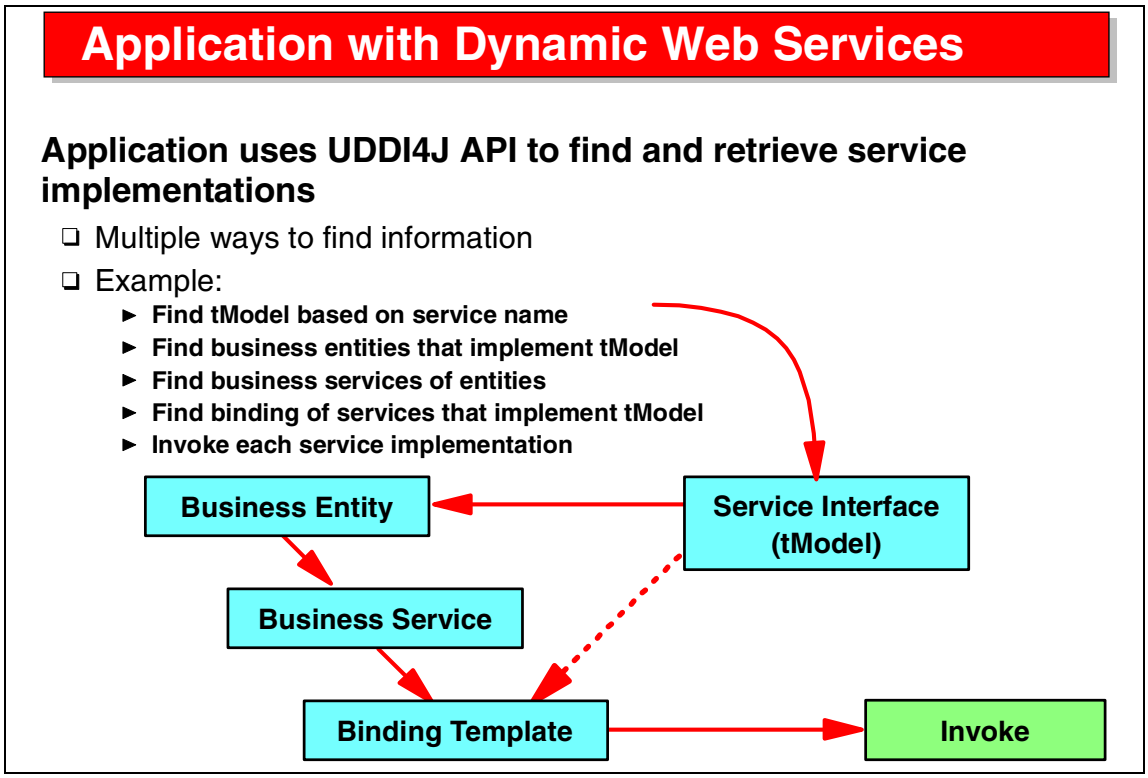
The XSL file matches the various tags of the XML result through templates:

- For the root tag, PartInventory, the HTML output is started (<HTML>), a heading is produced (<H1>, and the table is started (<TABLE>) with a table heading (<TH>).

  Processing of the other tags is then forced <xsl.apply-templates>).

  Finally, the table is closed and the HTML ended.

- For the Part tag, a new row in the table is started (<TR>) and for each detail tag (only the partNumber tag is shown) the value is placed into a table column (<TD>).



Visual 13-14 Application with Dynamic Web Services

An application with dynamic Web Services uses the UDDI Registry to find Web Service implementations.

For example, multiple manufacturers implement the PartInventory Web Service, and all their entries are stored under one business entity, the Automobile Association (this is to guarantee that all implementations are verified).

The process to find the implementations is outlined:

- Find the service interfaces (tModels) based on the service name (PartInventory).
- Find the business entity (auto parts association).
- Find all the business services for this entity.
- Find the implementations (binding templates) of these services, restricted to those that implement the given service interface.

Dynamic Web Service: Sample Code

UDDI API example to retrieve Web Services

```
inqAPI = "http://...ibm.../uddi/testregistry/inquiryapi"
pubAPI = "http://...../testregistry/protect/publishapi";
uddi = new UDDIProxy(inqAPI, updAPI);
TModelList tml = uddi.find_tModel(servicename,null,0);
Vector v1 = (tml.getTModelInfos()).getTModelInfoVector();
tmkey = ((TModelInfo)v1.elementAt(i1)).getTModelKey();
TModelBag tmb = new ....; // add Vector of tmkeys
BusinessList bl = uddi.find_business(provider, null, 0);
Vector v2 = (bl.getBusinessInfos()).getBus..InfoVector();
BusinessInfo bi = (BusinessInfo)v2.elementAt(i2);
Vector v3 = (bi.getServiceInfos()).getServiceInfoVector();
skey = ((ServiceInfo)v3.elementAt(i3)).getServiceKey();
BindingDetail bd = uddi.find_binding(null,skey,tmb,0);
Vector v4 = bd.getBindingTemplateVector();
AccessPoint ap = ((...)v4.elementAt(0)).getAccessPoint();
```

Setup and invoke proxy

skeleton code

```
proxy.setEndpoint( new URL(ap.getText()) );
result = proxy.retrievePartInventory_(pn);
```

Visual 13-15 Dynamic Web Service: Sample Code

The UDDI4J API itself uses a set of Web Services:

- A proxy object is instantiated, pointing to a UDDI Registry.
- The find_tModel method retrieves a list (actually a Vector) of tModels. The keys of these tModels are stored in a tModelBag for later use.
- The find_business method retrieves a Vector of business entities. Using the list, the services are retrieved.
- The find_binding method retrieves the implementations that point to the given service interface (using the tModelBag built earlier).
- The AccessPoint of the implementation is the address where the Web Service is running.

The AccessPoint is used to set the endpoint (the target) in the Web Service proxy object before invoking the Web Service itself.

Summary

Web Service wizard for using a Web Service

- Similar to creating a Web Service
- Test the Web Service with the test client
- Implement the client application
  - **Process XML results**
  - **XSL processor**

Dynamic Web Services

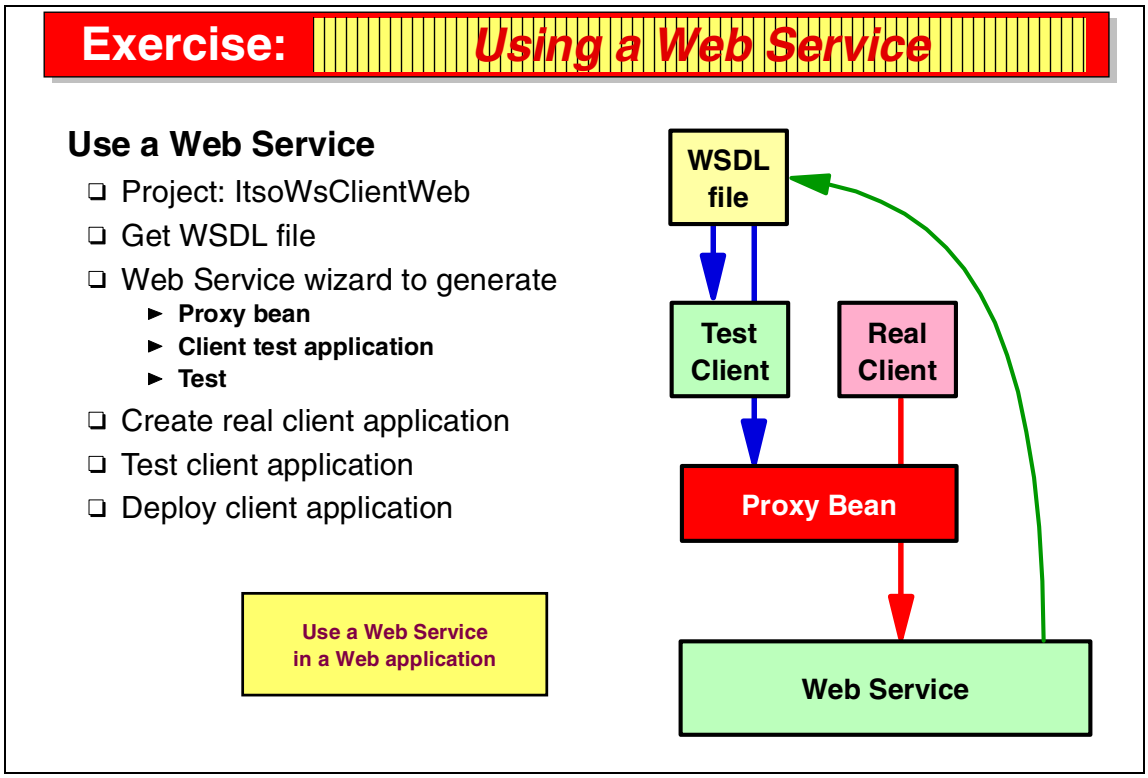
- Interact with UDDI Registry to retrieve Web Service and WSDL files
  - **Quite complicated programming**
- Dynamically set up the client proxy bean to invoke a Web Service

Visual 13-16 Summary

The Web Service wizard of the Application Developer generates the proxy object to be used for client applications that want to invoke a Web Service.

The proxy object can be used to invoke static Web Services as well as dynamic Web Services.

The invocation of the Web Service from a client application is simple when using the generated proxy object. Processing the result of a Web Service is the real challenge.



Visual 13-17 Exercise: Using a Web Service

The Web Service client exercise guides you through many of the tasks discussed in the presentation.

In this exercise you create a client Web application that invokes the Web Service created in Exercise 8, “Create a Web Service” on page 349:

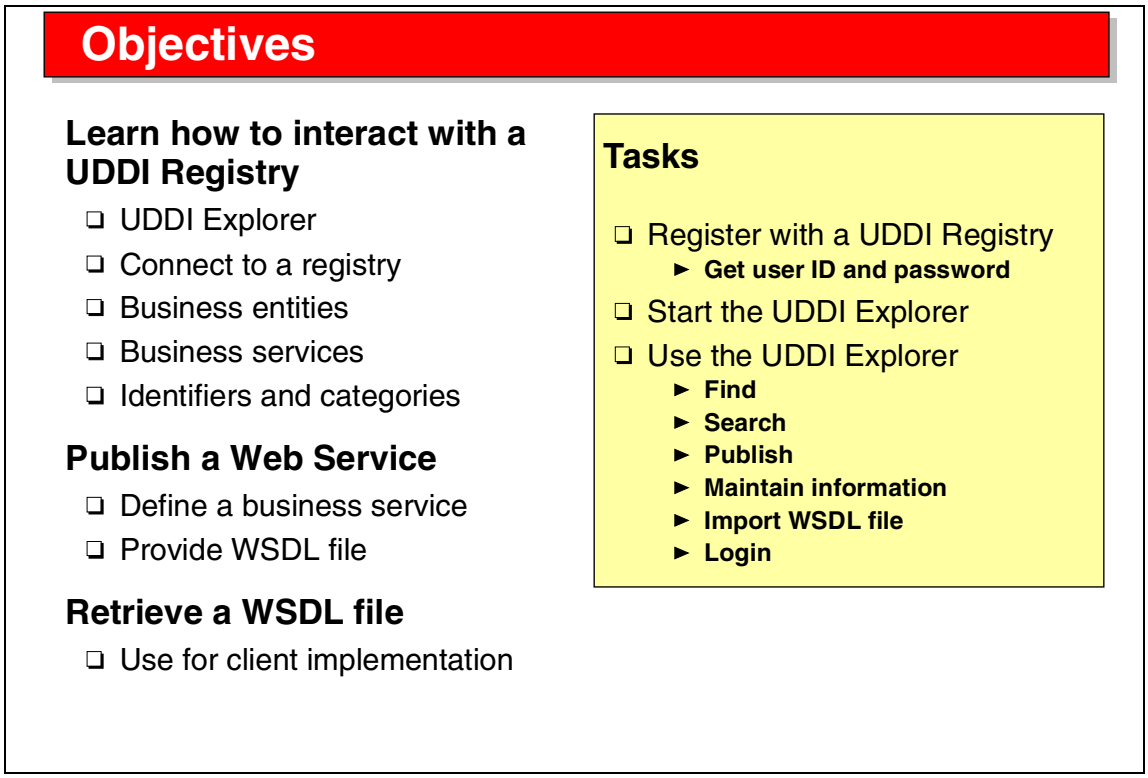
- You generate the proxy object from the WSDL implementation file.
- You create the Web application using a servlet with an XSL processor.

See Exercise 10, “Using a Web Service in a client application” on page 365 for the instructions for this exercise.

14

Web Services and the UDDI Explorer

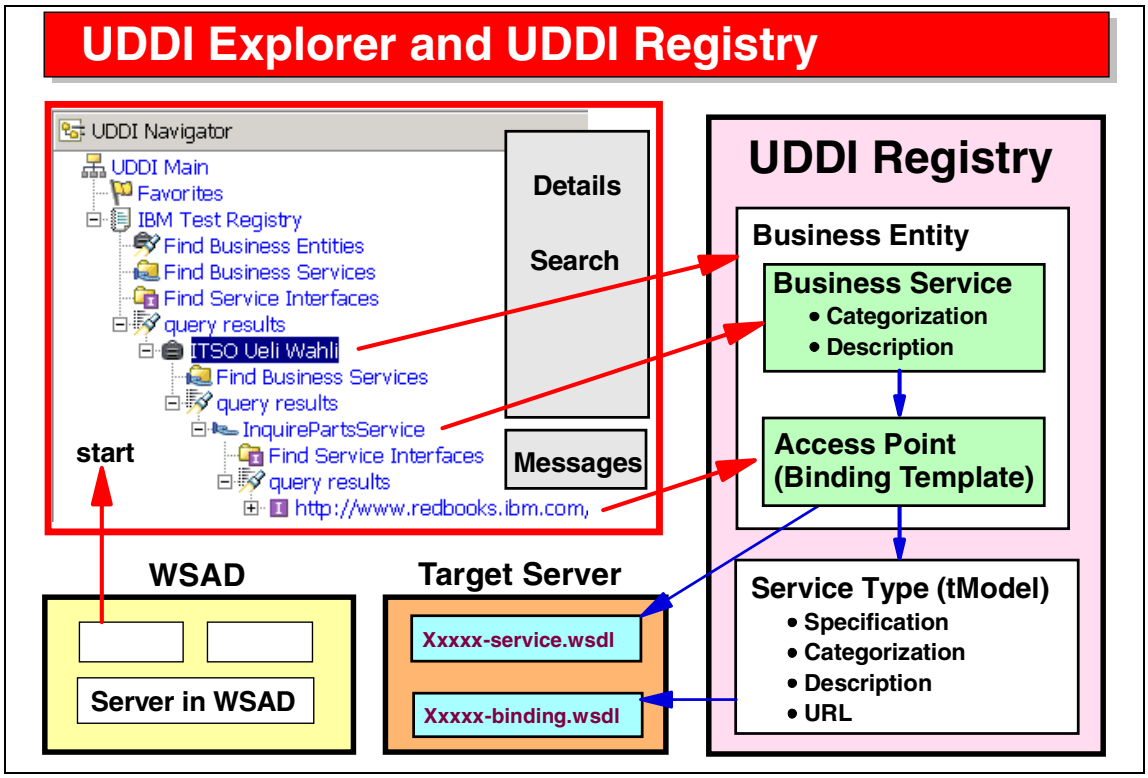
Visual 14-1 Title



Visual 14-2 Objectives

The objectives of this unit are to:

- Understand the UDDI Registry
- Understand the UDDI Explorer of the Application Developer
- Connect to a UDDI Registry and work with business entities and business services



Visual 14-3 UDDI Explorer and UDDI Registry

A UDDI Registry contains these entries:

- Business entities—companies that want to register Web Services
- Business services—a Web Service that the company registers
- Access point (called binding template)—points to an installed Web Service (the target address) and to the matching WSDL service (implementation) file
- Web Service type (called tModel)—a Web Service definition that points to the matching Web Service binding (interface) file

The Application Developer provides an internal server and a UDDI Explorer application that can be used to access and update any UDDI Registry that implements the UDDI4J API.

The UDDI Explorer enables a user to search for entries in the registry and to navigate between entries. Entries can also be created; for example, a business entity can be created and a WSDL file can be published.

UDDI Explorer

Start from Application Developer

- File -> Import or Export UDDI
  - **Select project for import**
  - **Select WSDL file for export (publish a Web Service)**
- Starts internal server on port 8090
- Starts normal browser
  `http://localhost:8090/uddiexplorer/uddiexplorer.jsp?.....`
- Browser communicates with UDDI registry through internal server

Access any UDDI registry

- Click on UDDI Main and set registry URL
- Default is IBM Test Registry
  `http://www-3.ibm.com/services/uddi/testregistry/inquiryapi`
- Production business registry
  `http://www-3.ibm.com/services/uddi/inquiryapi`
- IBM WebSphere UDDI Registry
  `http://hostname/services/uddi/inquiryapi`

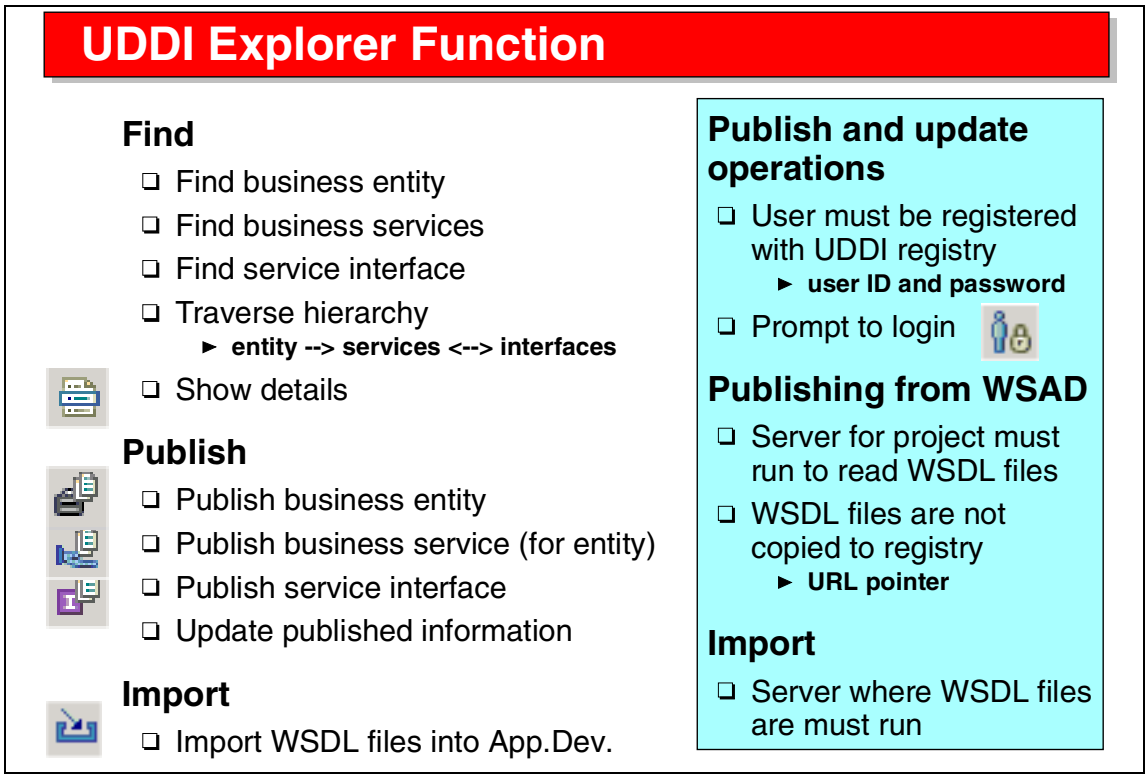
can add to Favorites

Visual 14-4 UDDI Explorer

The UDDI Explorer is started from the Application Developer using either import or export. Export would be used to publish a Web Service (WSDL file).

The UDDI Explorer is a Web application that runs in a browser. Connection to a UDDI Registry is done through an internal server.

The first action is to set up the address of a UDDI Registry, for example, the IBM Test Registry or a private registry (IBM WebSphere UDDI Registry).

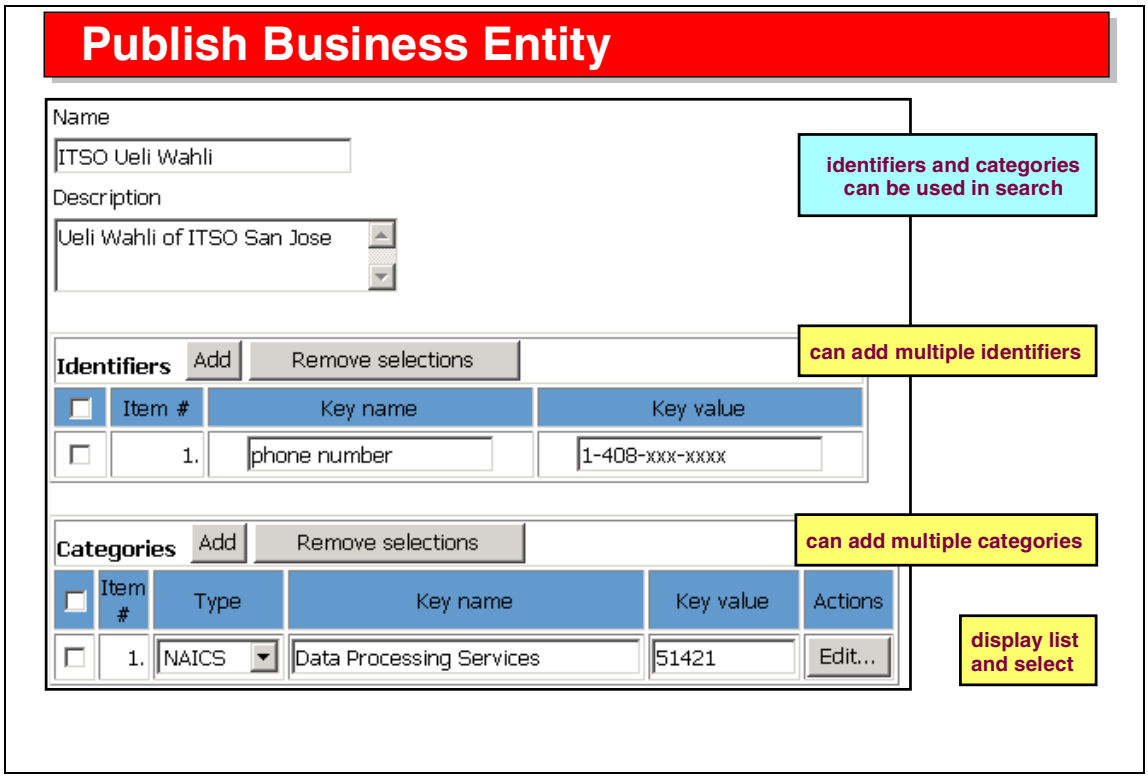


Visual 14-5 UDDI Explorer Function

The UDDI Explorer provides three main functions:

- Find—search for entries and navigate between entries
- Publish—publish business entities and business services, and update information
- Import—retrieve a WSDL file (note that the registry only points to WSDL files; the actual file is retrieved from a target server)

To publish and update information, you must be registered with the UDDI Registry and use a logon ID and password to connect.



Visual 14-6 Publish Business Entity

To publish information, you fill out forms provided by the UDDI Explorer.

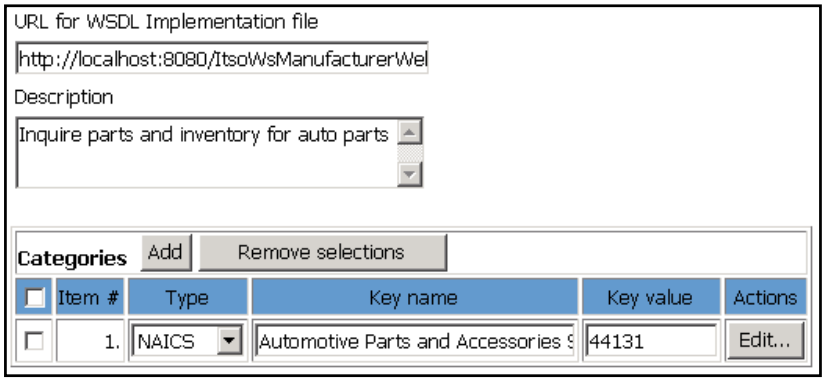
For example, you do the following to publish a business entity:

- Give it a name and description
- Assign identifiers (phone number, address)
- Assign categories according to one of the supported standards

Publish Business Service

- Select WSDL file to be published and **File -> Export -> UDDI**
- Find business entity
- Publish Web Service
  - **URL of WSDL file is pre-filled**

    http://localhost:8080/ItsoWsManufacturerWeb/wsdl/InquireParts-service.wsdl
- Select categories



Visual 14-7 Publish Business Service

To publish a Web Service, you point to the WSDL file that describes the Web Service and add a description and categories.

The server where the WSDL file is retrieved must be running.

The WSDL file is analyzed to extract the information that is stored in the registry.

Importing a WSDL File

To use a Web Service, you must have the WSDL implementation file

- Get the WSDL file from the service provider
- Get the WSDL file from the UDDI registry
  - **Find business service**
  - **Use Import action**
  - **WSDL file stored in selected project**
- UDDI registry points to URL of WSDL file
  - **Server must be running to retrieve the file**

From the WSDL file, using the Web Service wizard, generate

- Proxy bean
- JavaBean skeleton
- Client test application

Visual 14-8 Importing a WSDL File

To use a Web Service in a client application you need the WSDL implementation file. You can find the file through the UDDI Registry and retrieve it from the server where it is running.

You then use the WSDL file to generate the client proxy or the JavaBean skeleton (if you want to implement the Web Service).

Summary

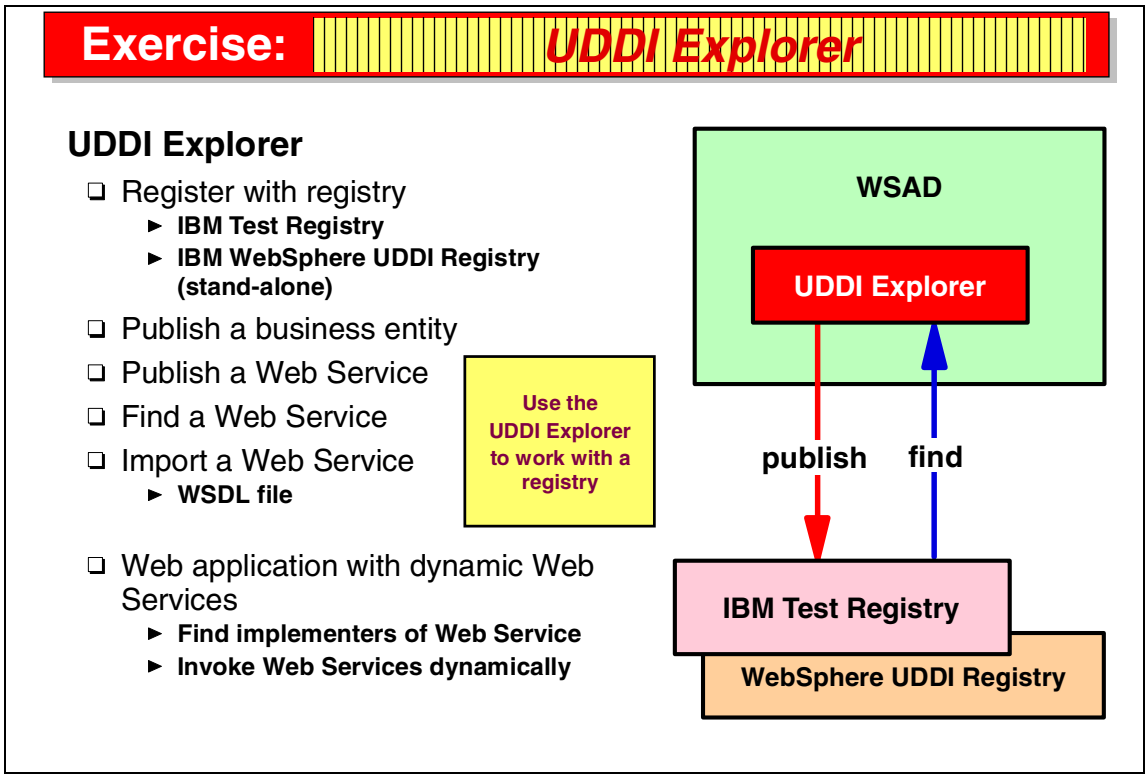
UDDI Registry can be accessed using a normal browser

Application Developer provides the UDDI Explorer for direct interaction

- Connect to a registry
- Browse a registry
- Publish to a registry
  - **Business entity**
  - **Business service**
  - **Service interface**
- Maintain information
- Retrieve WSDL files from a registry

Visual 14-9 Summary

The UDDI Explorer of the Application Developer provides the necessary facilities to automate access to a UDDI Registry and integrate the access into the application development process.



Visual 14-10 Exercise: UDDI Explorer

The UDDI Explorer exercise guides you through many of the tasks discussed in the presentation.

In this exercise you work with either the IBM Test Registry or the IBM WebSphere UDDI Registry:

- Register to get a logon ID
- Publish a business entity and a Web Service
- Find a Web Service and retrieve the WSDL file
- Optionally work with a dynamic Web Services application

See Exercise 11, “Web Service publishing in the UDDI registry” on page 369 for the instructions for this exercise.

Part 2

Exercises

In preparation for the hands-on exercises, you have to install the products and retrieve the sample code from the ITSO Web site. Then you can perform the exercises by following the instructions for each exercise.

See Appendix A, “Installation and configuration” for detailed installation instructions.

See Appendix B, “Additional material” for detailed instructions about downloading and installing the sample code from the ITSO Web site.

The instructions assume that the products and samples are installed as follows:

- DB2 in d:\SQLLIB
- WebSphere Application Server in d:\WebSphere\AppServer
- WebSphere Stusio Application Developer in d:\WSAD
- IBM WebSphere UDDI Registry Preview in d:\UDDI Registry Preview
- Sample code in c:\WS\sampcode
- Database ITSOWSAD created and loaded with sample data

Your setup may be different, so check the installation directories.

Sample data

The ITSOWSAD database has two sets of two tables, PARTS and INVENTORY:

- The dealer tables are AAPARTS and AAINVENTORY
- The manufacturer tables are MMPARTS and MMINVENTORY

AAPARTS and **MMPARTS** tables are identical:

PARTNUMBER	NAME	WEIGHT	DESCRIPTION
M100000001	CR-MIRROR-L-01	10.50	Large drivers side mirror for Cruiser
M100000002	CR-MIRROR-R-01	10.80	Large passenger side mirror for Cruiser
M100000003	CR-MIRROR-R-01	4.60	Large rear view mirror for Cruiser
W111111111	WIPER-BLADE	0.90	Wiper blade for any car
B222222222	BRAKE-CYLINDER	2.20	Brake master cylinder
X333333333	TIMING-BELT	0.60	Timing belt
T0	Team	100.00	International WSAD Groupies
T1	Olaf	100.11	German Turbo Engine
T2	Wouter	100.22	Belgium Chocolate Steering Wheel
T3	Denise	100.33	US Spark Plug
T4	Mark	100.44	British Muffler
T5	Ueli	100.55	Swiss Cheese Cylinder
L1	License	0.30	Personalized license plate

AAINVENTORY table:

ITEMNUMBER	PARTNUMBER	QUANTITY	COST	SHELF	LOCATION
21000001	M100000001	10	89.99	2A	AA - Almaden
21000002	M100000002	5	99.99	2B	AA - Almaden

MMINVENTORY table:

ITEMNUMBER	PARTNUMBER	QUANTITY	COST	SHELF	LOCATION
21000003	M100000003	10	59.99	L8	MM - San Francisco
21000004	M100000003	12	57.99	B7	MM - New York
31000005	W111111111	2	12.34	H8	MM - Los Angeles
31000006	B222222222	13	123.45	E5	MM - Frankfurt
31000007	X333333333	7	12.34	2D	MM - Santa Cruz
900	T0	1	99.00	M0	MM - San Jose
901	T1	1	11.00	M1	MM - Heidelberg
902	T2	1	22.00	M2	MM - Brussels
903	T3	1	33.00	M3	MM - Raleigh
904	T4	1	44.00	M4	MM - London
905	T5	1	55.00	M5	MM - Zurich
910	L1	1	30.00	M6	MM - California

1

Java development

What this exercise is about

In this lab we set up a Java project in WebSphere Studio Application Developer and implement simple Java applications.

User requirement

Generate a list of all the parts in the database.

What you should be able to do

At the end of this lab you should be able to:

- Start WebSphere Studio Application Developer
- Define a Java project
- Import and work with Java code
- Run and debug Java applications

Introduction

We implement a command line and a GUI application to list the parts table.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Define a Java project

2. Select *File -> New -> Project* (or use the *New* button in the toolbar). In the SmartGuide, select *Java* and *Java Project*, then click *Next*.
3. Enter **ItsoWsDealerParts** as the project name, and leave *Use default location* selected. Click *Next*.
4. Leave all the *Java Settings* for now (we use the project as source folder). Click *Finish*.
5. Select the Java Perspective. If it is not in the Perspective Bar (on the left), select *Perspective -> Open -> Java*.

Create a package and a class

6. Create a Java package (*File -> New -> Java Package* or *New* button). Enter **itso.wsad.dealer.app** as the package name, then click *Finish*. The package appears in the Packages view.
7. Select the new package and create a class (*File -> New -> Java Class* or *New* button). In the SmartGuide, check the package name (itso.wsad.dealer.app) and enter **Listing** as the class name. Leave Object as superclass. Select the *main* method under *Which method stubs would you like to create*. Click *Finish*. The Listing class appears under the package and a Java editor opens.

Complete the code

8. In the editor, under the package statement, add an import statement:

```
import java.sql.*;
```

9. Add a static field under the class:

```
// database table
static String dbtab = "aaparts";
```

10. Open the sample file **c:\ws\labscode\exjava\cmdapp\ListingMain.txt** and copy/paste the complete method code into the *main* method between the braces {}. Notice the color highlighting of the source code.
11. Save the Listing.java file (*File -> Save*, or *ctrl-s*). The program is compiled and one error is noted in the task list (method getnext() is undefined). Double-click the entry in the task list, and the bad code is highlighted in the editor.

Code assist and hover help

12. Delete the *getnext* text, place the cursor after *rs.* and press *ctrl-space* for code assist. Select the *next()* method from the list and double-click to add it into the code.
13. Save the file (*ctrl-s*), and the error disappears.
14. Place the cursor on variable references (rs, con, stmt) and watch the hover help appear. It does not work on the variable declaration. Hide the hover help by clicking the *Text Hover* button in the toolbar (third, from right to left).
15. Double-click the title bar of the edit view to make the source code occupy the complete WSAD window. Double-click again to make it small. Adjust the size of the panes by moving the borders.

Outline view

16. Select elements in the Outline view and watch the Java editor mark the selected part of the source code and position it to the top.
17. Click the various buttons in the outline toolbar to see (or not see) fields, static or public definitions.
18. Select the edited file and click the *Show Source of Selected Element Only* button in the toolbar (fourth from right). Now select elements in the outline, and the editor only displays the selected element. Reset the button for the editor to see the complete source.
19. Grab the title bar of the outline view and move it away to become a separate window.
20. Move it back over the main window. Watch the icon at the top of the moving window change to squares (separate window), folders (overlay a view), and arrows (add view above, below, left, right). Try different positions. Move it back to the right border when done experimenting.

Replace from local history

21. Select the *main* method in the outline view, and from the context menu select *Replace from Local History*.
22. A dialog opens and shows all the states of the main method that you saved. When you select a local version in the top pane, the lower right pane displays the selected version and highlights the differences to the code loaded in the Workbench (left pane). You could replace the code with an older version. Click *Cancel*.

Smart import assist

23. Select the *import declarations* in the outline, and *Delete* from the context menu. The import statement is removed from the source. Do not save the code, or you will get many errors.
24. In the editor, select *Organize Imports* from the context menu. A dialog opens to choose the types to import (because there is more than one Connection class in the Workbench). Select *java.sql.Connection* and click *Finish*.
25. Note that four import statements are added to the source. The other three types were not ambiguous. Save the code.

Extracting a method

26. We want to make a separate method to connect to the database. In the editor, select the lines:

```
try {
   Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
   con = DriverManager.getConnection("jdbc:db2:itsowsad");
} catch(Exception e) {
   System.err.print("Exception: ");
   System.err.println(e.getMessage());
}
```

27. From the context menu, select *Extract Method*. In the Refactoring SmartGuide, enter `connect` as the new method name, and select *protected* as the access modifier. Click *Next*.
28. The SmartGuide shows how the selected lines of code will be replaced by a method call, and how a new connect method is created from those lines.

```
con = connect();
protected static Connection connect() { ... }
```

29. Click *Finish* to generate the new method. An error appears in the task list because the variable is not initialized. Note that you can see the error message as hover help when placing the cursor on the red mark in the editor window.
30. Fix the code in the connect method and save the code.

```
Connection con = null;
```

Running the application

31. Notice the running man next to the class name in the Outline view. This indicates that the class is executable.

32. Click the *Run* icon in the toolbar (running man). The first time, you are prompted to select the launcher. Select *Java Application* (it is remembered for the project), then click *Next*. The Listing class is preselected, so click *Finish*.

33. The Debug Perspective opens and the Console displays an error message because the DB2 driver class is not found. If you want the Debug Perspective to open automatically, use *Window -> Preferences -> Debug***.** The Processes view shows that the application terminated.

Setting the build path

34. We have to add the **db2java.zip** file to the build path. We could add the file to the project itself, but it is better to refer to the file in the DB2 directory. In the Java Perspective, select the project and *Properties* (context).

35. Select the *Java Build Path*, and the *Library* page. Click *Add External JARs* and navigate to **d:\SQLLIB\java\db2java.zip**, and then click *Open*. The file is added to the build path. Click *OK*. The db2java.zip file appears in the Packages view with the library symbol.

    A better approach is to use a class path variable, so that there is no direct reference to a real directory, that may be different for another developer. A variable DB2JAVA may already exist, pointing to the db2java.zip file. See *Window -> Preferences -> Java -> Classpath Variables*.

36. Click *Run* again; this time the output should appear in the Console view of the Debug Perspective. Close the Debug Perspective.

Import a Java source file

37. In the Java Perspective, select the ItsoWsDealerParts project to import a GUI application. Select *File -> Import -> File system*, then click *Next*. Click *Browse* and navigate to the **c:\ws\labscode\exjava\guiapp** directory. Select the *guiapp* directory and click *OK*. Select the check box (in front of guiapp) to import all subdirectories and files. Check that the correct project is in the Folder field, and click *Finish*.

    Do *not* select the check box *Create complete folder structure* (under Options).

38. Two Java source files are imported and the package itso.wsad.dealer.parts is created under the project. Both files show errors that refer to missing packages (com.ib.ivj, com.ibm.db). This application was created in VisualAge for Java using the data access beans feature, which is missing in our project.

39. We have to import the JAR file with the supporting packages and classes (ivjdab.jar) and add it to the build path.

Select the project and *File -> Import -> File system, Next, Browse* to navigate to **c:\ws\labscode\exjava\lib**, and then click *OK*. Select the lib directory (click *lib*) to see the files that are in the directory in the right pane; it contains the ivjdab.jar file. Select the ivjdab.jar file (check box) and click **Finish**. The file appears under the project.

40. Select the project and *Properties* (context)**,** and select *Java Build Path* and the *Libraries* page. Click *Add JARs*. In the JAR Selection dialog, expand the ItsoWsDealerParts project and select the ivjdab.jar file. Click *OK*. Close the Properties dialog with *OK*.
41. Notice that the icon of the ivjdab.jar file changes to a “jar” that can be expanded to reveal the content, packages with classes.
42. All the errors have been resolved, except for a deprecated method warning in the javax.swing.JViewport class. Double-click the warning to open the GuiListing file in the source editor. Also notice the Outline view with all fields and methods.

Search

43. In the source editor error line, select the *setBackingStoreEnabled* method name, and from the context menu, select *Search -> Declarations in Hierarchy*. This is a fast way of opening the Search dialog (flashlight icon or *Edit -> Search*, entering the method name, and searching for method and limit to declarations).
44. The Search view opens with the method found in the JViewport class. Double-click on the search match and the JViewport class opens in the editor, but no source is available. Select the editor window and the outline appears. However, this does not really help to solve the problem because the documentation for JViewport is not available.
45. In the GuiListing.java editor, delete the setBackingStoreEnabled(true) call, and press *ctrl-space* after getViewport() and select the **setScrollMode(int)** method. Enter `javax.swing.JViewport.BACKINGSTORE_SCROLL_MODE` as parameter (use **ctrl-space** instead of typing):

```
getJScrollPane1().getViewport().setScrollMode
       (javax.swing.JViewport.BACKINGSTORE_SCROLL_MODE);
```

46. Close the JViewport class in the edit view. Save the GuiListing.java file.

Run GUI program

47. Click *Run* and the GuiListing applet appears. Enter a partial name (for example, **IRRO**) and click *Select*. Matching parts should be listed in the table. Close the Debug Perspective.

Debugging

48. Open the Listing.java application in the editor. Double-click in the left border on the line **“`Connection con = null;`”** to set a breakpoint.
49. Click the *Debug* icon in the toolbar. The application runs in debug mode and the Debug Perspective opens at the breakpoint. Study the different views. Click *Variables* under the Breakpoints view and you should see the dbtab variable (click the *S* icon to show or not show static fields).
50. Step through the code using the *Step over* icon in the Debug view. Watch the **con** variable appear in the Variables view. Continue stepping through the code until you pass over the line “`while (rs.next()) {`“.
51. In the Listing.java view, select the text **“**`rs.getString("name")`**“** and select *Display* in the context menu. The evaluated expression is in the Display view (top right).
52. In the **Variables** view, select the variable *rs=DB2ResultSet* and select *Inspect* from the context. Study the variable content in the Inspector view.
53. In the Debug view, click the *Resume* icon to run the program to the end.
54. Close the Debug Perspective. Close the Listing.java editor.

Type hierarchy (optional)

55. Select the *GuiListing* in the Outline view and select *Open Type Hierarchy* from the context menu.
56. The Types view opens and shows the hierarchy of the class. The bottom part of the view shows methods and fields. Select a superclass in the top part, and the fields and methods change in the bottom part.
57. In the Types view, click the *Show the supertype hierarchy* icon (second from the right). Now you see the hierarchy inversed, and you also see the interfaces that are implemented.
58. Select the Applet class and *Open Type Hierarchy* (context). Now you can browse the hierarchy from the Applet point of view. Switch between GuiListing and Applet using the arrow icons.
59. Select the *GuiListing* in the top pane, and the *init* method in the bottom pane. Then click the *Lock View and Show Members in Hierarchy* icon in the bottom pane (first on the left). In the top pane you can see in which superclass(es) the method is defined. Select the *paint* method for comparison.
60. In the GuiListing editor, find the *getSelect1* method (use the Outline view). Find the code where the PartsDbAccess class is used for the connection and SQL statement.

61. Select *PartsDbAccess* and *Edit -> Open on Selection*. This opens the editor for that class. Close the editor for that class.
62. Select *PartsDbAccess* and *Edit -> Open Type Hierarchy*. This opens the Types view for this class. Close the Types view and the editor of that class.

Rename (optional)

63. Select the *PartsDbAccess* class in the Packages view and *Rename* (context). In the Refactoring dialog, change the name to PartsAccess.java. Click *Next*.
64. The dialog shows you all the changes that will be performed (for example, the references in the GuiListing class). Click *Finish*.
65. The getSelect method in GuiListing has been updated and the file has been renamed. Run the GUI application; it should still work.

Scrapbook page (optional)

66. In the Java Perspective Packages view, select the project and create a scrap page (*File -> New -> Java -> Java Scrapbook Page*). Click *Next*. Enter **loop** as the file name and click *Finish*. A file named loop.jpage is added to the project, and the file opens in the editor.
67. Add this code to the empty editor pane (or copy/paste from **c:\ws\labscode\exjava\scrap\loop.txt**):

```
String total = "";
for (int i=1; i<11; i++) {
   System.out.println(i + " square " + i*i);
   total = total.concat(i*i +" ");
}
return total;
```

68. Select all the code and click the *Run selected code* icon (with the J) or *Run* in the context menu. The output appears in the Console view.
69. Save the loop.jpage file and close the editor.

What you did in this lab

- Defined a Java project
- Used multiple perspectives and views to work with the Java project
- Created packages, created and imported files
- Fixed code errors
- Used the search facility
- Used code refactoring (rename, method extract)
- Set the build path and run applications
- Used the debugger and a scrapbook

2

Relational Schema Center

What this exercise is about

In this lab we set up a project in WebSphere Studio Application Developer and work with database schemas, DDL, and SQL statements.

User requirement

Generate a list of all the parts in the database.

What you should be able to do

At the end of this lab you should be able to:

- Define databases, tables, columns
- Connect to a database and import existing schemas
- Create SQL statements

Introduction

We connect to the ITSOWSAD database and work with tables and SQL statements.

Exercise instructions

1. Start WebSphere Studio Application Developer if not already started.

Define a project for relational database

2. Create a new project (*File -> New -> Project*), select the **Simple** type, click *Next*. Enter **ItsoWsDealerDatabase** as project name and click *Finish*.
3. The Resource perspective is open. Close it, and open the **Data** perspective (*Perspective -> Open -> Other -> Data*). Notice the three views in the left pane: DB Explorer, Data View, Navigator.

Create a database connection and import tables

4. In the DB Explorer, select *New Connection* from the context menu (put cursor into the empty space). In the dialog enter:
   - Connection name: **ConITSOWSAD**
   - Database: **ITSOWSAD**
   - User ID and password: empty
   - Database Vendor Type: DB2 UDB V7.2
   - JDBC driver: IBM DB2 APP DRIVER
   - Host and port: empty
   - Class location: check that **d:\SQLLIB\java\db2java.zip** is selected
5. Click on *Filters*. In the dialog enter **MM%** as new filter and click *Add Filter*. Select the filter and change the predicate from NOT LIKE to *LIKE*. (Click on NOT LIKE until you get a pull-down.) Select the enabled check box (it should be already selected). This retrieves the MM tables only. Leave the check box *Exclude system schemas* selected. Click *OK* to close the filters. Click *Finish*.
6. The database, schema, and two tables appear in the DB Explorer. Double-click on a table and it expands into columns, but no editor opens. The Explorer view cannot be used for editing.
7. Select the **ITSOWSAD** database and *Import to Folder* (context). In the import to folder dialog select the *ItsoWsDealerDatabase* project. Click *OK* and then *Finish*.
8. Switch to the **Data** view and you find the database, schema, and the two MM% table definitions added. Double-click on a table to open the table editor. Go through the panels (bottom), then close the editor.
9. Switch to the **Navigator** view and study the underlying XMI files that contain the database, schema, and table definitions. You can double-click a file and the same editor as in the Data view appears.

Create a database and a table

10. In the Data view select the ItsoWsDealerDatabase project. Create a new database (*File -> New -> Other -> Database* or *New -> New Database* from context). Enter **ITSOTEST** as database name and DB2 7.1 as database vendor type. Click *Finish*.

    Note: In such a manual database you cannot execute SQL statements because no connection definition exists.

11. Select the ITSOTEST database and *New -> New Schema* (context). Enter **TEST** as schema name, click *Finish*. These steps only illustrate that DB, schemas and tables can be created - we do not use ITSOTEST further.

12. We will do all work in the **ITSOWSAD** database that was created through the connection. Expand the ITSOWSAD database and ITSO schema. Select the schema and *New -> New Table* (context). Enter **AAPARTS** as table name, click *Next*.

13. Click *Add another* to define the columns. After entering the data for each column, click *Add another* again.

    – Name: PARTNUMBER, type: CHARACTER, length: 10, not null (nullable not checked), not bit (for bit data not checked)
    – Name: NAME, type: CHARACTER, length: 30, nullable, not bit
    – Name: DESCRIPTION, type: VARCHAR, length: 100, nullable, not bit
    – Name: WEIGHT, type: DOUBLE, nullable
    – Name: IMAGE_URL, type: VARCHAR, length: 100, not null, not bit

14. Click *Next*. Enter **PARTKEY** as primary key name, select the *PARTNUMBER* column and click *>>*. Click *Next* (nothing to enter for foreign keys), click *Finish*.

15. The ITSO.AAPARTS table appears under the schema/tables. Double-click on the table and a table editor appears, where you could make changes using the different panels. Close the editor.

Generate, import, and run DDL

16. Select the ITSO.AAPARTS table and *Generate DDL* (context). The filename **AAPARTS.sql** is prefilled. Leave the default options (fully qualified names). Click *Finish*.

17. The file AAPARTS.sql is added to the project. Double-click to bring up an editor and look at the DDL. Close the editor.

18. Import a DDL file. Select *File -> Import -> File system*, click *Next*. Click *Browse* and navigate to the **c:\ws\labscode\exdata\ddlimport** directory. Select the directory and the **AAINVENTORY.sql** file in the directory. Set the

target folder by *Browse* and locate the ItsoWsDealerDatabase project. Do not select *Create complete folder structure*. Click *Finish*.

19. Select the AAINVENTORY.sql file and *Execute* (context). In the Execute dialog click *Browse* for default schema and select the ITSWSAD database and ITSO schema. Click *OK* and *Finish*.
20. The ITSO.AAINVENTORY table is added to the schema. Double-click on the table to open an editor. Go through the pages; the table includes foreign key to the AAPARTS table. Close the editor.
21. To actually run the DDL into DB2, we would have to use a DB2 command window or the DB2 Control Center. However, the tables do already exist in DB2.
22. You can also generate a DDL script for the database (or schema). It will include all the components (tables).

SQL Query Builder (optional)

23. In the **Data** view, select the *Statements* folder (under the ITSOWSAD database) and *New -> Select Statement*. Enter **PartListing** as name and click *OK*. The editor opens with SELECT *.

    Note: If the Statements folder is not visible, close the Data Perspective and reopen it; this may help. Otherwise, select *New -> Data -> SQL Statement*, click *Next*, and a wizard appears. Select *Create an SQL resource and invoke the SQL Builder*, *Use existing database model* (*Browse* to *ItsoWsDealerDatabase -> ITSOWSAD*), enter **PartListing** as statement name, and the file is created and opened in the editor.
24. In the Data view, expand the database, schema (ITSO) and tables. Drag the *MMPARTS* and *MMINVENTORY* tables into the middle (or top) pane of the editor. The tables are shown graphically in the middle pane and are listed in the top pane.
25. Select desired columns in the middle pane, for example, *partnumber* and *name* in MMPARTS, and *quantity* and *cost* in MMINVENTORY.
26. Drag the *partnumber* column from MMPARTS to the same column in MMINVENTORY to create the join.
27. In the bottom pane for Columns, select the sort type (Ascending) for the partnumber column.
28. In the bottom pane for Conditions, select the *quantity* column, operator <, and for the value type **:quantity** (host variable). Click in another field to complete the line.

29. Run the SQL statement (*SQL run icon*). In the execute dialog, click *Execute*, and specify a value for *:quantity*, for example **11**. Click *Finish* and the result rows appear. Close the dialog.
30. Save the SQL statement and close the editor. When you are prompted for the host variable value, click *Cancel* to proceed - we want to leave the host variable in the statement. The SQL statement is saved as file ITSOWSAD_PartListing.sqx.

What you did in this lab

- Connected to a database to view tables
- Imported database, schema, and tables into local storage
- Manually created database, schema, and table objects
- Generated DDL, imported DDL, and executed DDL
- Built and ran an SQL statement

3

XML development

What this exercise is about

In this lab we set up a Java project in WebSphere Studio Application Developer and work with XML files.

User requirement

Use XML for distribution of the parts inventory.

What you should be able to do

At the end of this lab you should be able to:

- Work with DTDs and XML schemas and convert one format into the other
- Work with XML files that match a DTD or an SXML schema
- Create an XSL mapping between DTDs and transform an XML file
- Generate XML output from an SQL query

Introduction

We use XML to represent parts and inventory.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Define a Java project and import files

2. Define a new **Java** project named **ItsoWsDealerXml**. Click *Next*. Select *Use source folders contained in the project* and click *Create New Folder* and create a **source** folder. For the *Build output folder* enter /ItsoWsDealerXML**/classes**, and click *Finish*.
3. Open the XML Perspective (*Perspective -> Open ->* ...)
4. Import a DTD and an XML schema. Select *File -> Import -> File system*, click *Next*, and *Browse* to the **c:\ws\labscode\exxml\schemas** directory. Click *OK*. Select the schema's directory to show the list of files. Import **Part.dtd** and **PartDef.xsd**. Set the folder to the *ItsoWsDealerXml* project. Click *Finish*.
5. The files appear in the Navigator view.

Edit DTD and XML schema

6. Edit the **Part.dtd** file. Notice the Outline view and the editor. In the editor, switch between Design and Source views. You have to use the Outline to move to different parts in the design or in the source.
7. In the Outline, select *Part.dtd* and *Add Element* from the context. Enter **Location** as name (overtype NewElement in the Design view). Select the *EMPTY* subitem under Location (in the Outline) and change it to *#PCDATA* (pull-down) in the editor (Design view).
8. Expand the *Inventory*. Select the model group **(,)** and *Add Element to Content Model* (context). Select *Location* for the name (pull-down) and select the *Optional* radio button. Save the changed DTD and close the editor.
9. Generate an XML schema from the DTD. Select the DTD and *Generate -> XML Schema* from the context menu. The resulting schema file is named **Part.xsd**. Click *Finish*.
10. Edit the Part.xsd file. Check out the Outline, Design, and Source views. Select and expand the elements in the Outline.
11. Edit the **PartDef.xsd** file. This is basically the same schema as Part.xsd, but improved. For example, the Cost is defined as decimal (not string), the Weight is defined using a *weighttype* with a constraint (maximum 100), and the quantity is defined using a *quantnum* type (range 1-20). Default values are also defined.

12. Select the PartDef.xsd file in the Navigator and *Generate -> DTD* (context). Edit the resulting **PartDef.DTD** file and notice in the Source view that the special datatype definitions could not be converted to a DTD.
13. Close the editors.

Work with XML files

14. Select the Part.dtd and *Generate -> XML File* (context). Name the output file **Part.xml** (default) and click *Next*. Select the root element (Part) and *Create required and optional content*. Click *Finish*.
15. The Part.xml file appears in the outline and editor. Enter some data for the elements, for example, partnumber **H3**, name **Headlight**, weight **5.5**, inventory item **H301**, quantity **4**, cost **34.56**, shelf **S2**. Save the file; there should be no errors.
16. Switch to the Source view. Delete the **<Quantity>4</Quantity>** line. Save the file. You get an error in the Tasks list, because quantity is required by the DTD.
17. Select the *Inventory* element in the editor (Design view) and *Add Child -> Quantity* (context). Set the value to 4 and save; the error disappears. You can also use the Validate icon (page with check mark) in the toolbar.
18. Select the *Part* and *Add -> Child -> Inventory* to add another inventory item. Notice that only required elements (quantity, cost) are added. Set some values for item (H302), quantity (5), and cost (67.89), and save the file. Close the editor.
19. Import an XML file. Select *File -> Import -> File system*, click *Next*, and *Browse* to the **c:\ws\labscode\exxml\xml** directory. Click *OK*. Select the xml directory to show the list of files. Import **PartDef.xml**. Set the folder to the *ItsoWsDealerXml* project. Click *Finish*.
20. Edit the **PartDef.xml** file. Notice in the Source and Design view that it refers to the XML schema (PartDef.xsd), not the DTD.
21. Validate the file using the toolbar icon. You get an error message box and 3 errors in the Tasks list. The PartDef.xsd file defines data types and valid values. Fix the errors in the design or source: weight 11.5, quantity 6, cost 40.00. Validate again and the errors disappear.
22. Use *File -> Save PartDef.xml As* to save the corrected file as **PartDefFixed.xml** and close the editor.

Generate an HTLM form

23. Select the Part.dtd file and *Generate -> HTML Form* (context). In the dialog select all fields (expand first), and click *Next*. For the servlet name, change *testServlet* to *PartXmlServlet*. Click *Finish*.

    Edit the resulting **Part.html** file. You can see the form that is generated. Note that the servlet is not generated for you—it is only the name used in the HTML form action. We work with HTML and Web development in later exercises. Close the editor.

XML to XML mapping

24. Import the file **Partmap.dtd** from the **c:\ws\labscode\exxml\mapping** directory into the project. This DTD is similar to the Part.dtd file.
25. Select *File -> New -> XML to XML Mapping*. Set the folder to ItsoWsDealerXml, the filename to **Partmap.xmx** and click *Next*.
26. In the Source dialog, select the **Part.dtd** (under ItsoWsDealerXml) and click **>** to copy the name to the right. Click *Next*. In the Target dialog, select **Partmap.dtd** (under ItsoWsDealerXml), click *Next*. In the Root dialog, *Part* should be preselected. Click *Finish*.
27. The editor opens with the **Partmap.xmx** file. Expand the elements on both sides to perform the mapping.
28. Drag elements from left to right: Partnumber to ID, Name to Name, ..., Inventory to Inventory, ... Cost to Cost. The mapping appears in the bottom pane. A shortcut is to select the Part element in the bottom pane and *Match by Name* from the context. (Partnumber to ID must be done manually.)
29. Select *Shelf* and *Location* (ctrl-key) and drag to *Where*. This creates a composite mapping.
30. Select *Where* in the bottom pane and *Define XSL Function* (context). In the dialog, leave String selected, and click *Next*. As function, leave *concat*. Click *Add* and enter ‘ in ‘ (including single quotes and spaces) and click *OK*. Select the entry and use *Up* or *Down* to place the constant between shelf and location. Click *Finish* and a function icon is added to the mapping line for Where. Save the code.
31. Click the *Generate XSLT script for mapping* toolbar icon (last on right). As file name leave the default, **Partmap.xsl**. Select the ItsoWsDealerXml project and click *Finish*. The generated Partmap.xsl file opens in the editor. Close the editor for both files.

Translating an XML file

32. In the Navigator view, select the files **Part.xml** and **Partmap.xsl** (ctrl-key) and *Apply XSL -> As XML* (context).
33. The XSL Trace Editor opens and you can see the result XML. Notice in the Result XML pane that the second inventory created the <where> tag with just the word ‘in’ because no shelf and location were present.
34. Use the arrow buttons to step through the code and watch the input, XSL, and output lines being highlighted.
35. Save the result XML file (*File -> Save XSL Trace Editor As*) and enter **Partmapresult.xml** as the file name.

SQL to XML mapping (optional)

36. In the Navigator view create a new folder named **sqlxml** in the ItsoWsDealerXml project.
37. Switch to the Data perspective and the **ItsoWsDealerDatabase** project. In the DB Explorer view look at the *ConITSOWSAD* connection. If no database content is displayed under the connection, select *Reconnect* from the context menu. (A connection is lost if you shut down WSAD.)
38. In the Data view, select the **PartListing** SQL statement you built in the RBD lab, and select *Generate new XML* from the context menu.
39. Select *Primary keys as attributes*, *XML Schema*, *Generate query template file* (PartListing.xst), and the *sqlxml* folder in the *ItsoWsDealerXml* project as output folder. Click *Finish*.
40. If you used a host variable, you are prompted for input. Enter **10**, for example.
41. Switch to the Navigator view and the *sqlxml* folder. Open the various files:
    - PartListing.xml contains the XML output
    - PartListing.html shows the output as an HTML table
    - PartListing.xsd is the underlying XML schema
    - PartListing.xsl is the XSL transformation file
    - PartListing.xst is the template with the SQL statement and the options
42. Close all files.

What you did in this lab

- Imported DTD, XML schema, and XML files
- Worked with DTDs, XML schemas, and XML files
- Generated and validated XML files
- Transformed an XML file from one DTD to another
- Generated XML and HTML output from an SQL query

4

Web development

What this exercise is about

In this lab we set up a Web project in WebSphere Studio Application Developer and implement simple Web applications.

User requirement

Generate a list of selected parts in the database. Query the database for parts with low inventory.

What you should be able to do

At the end of this lab you should be able to:

- Create and test Web applications
- Work with the Page Designer
- Configure and run the WebSphere test environment
- Use the Database Wizard to generate a Web application

Introduction

We use WSAD to implement Web applications using servlets, HTML, and JSPs.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Define a Web project

2. Define a new project (*New -> Web -> Web Project*) with the name **ItsoWsDealerWeb**. Use the default location. For the EAR project name, enter **ItsoWsDealerEAR**. For the context root, leave ItsoWsDealerWeb. Click *Next* twice. Leave the default source location (ItsoWsDealerWeb/source) and the default output folder (ItsoWsDealerWeb/webApplication/WEB-INF/classes). Click *Finish*. This creates both the Web and the EAR project.
3. Open the Web Perspective and expand the ItsoWsDealerWeb project. You should find the source folder (for Java servlets), the webApplication folder with the WEB-INF folder, and the web.xml deployment descriptor file.

Import a Web application

4. Select the *webApplication* folder and create a subfolder named **images**. Make sure to have it under webApplication!
5. Select the *images* folder and *File -> Import -> File system*, *Browse* to **c:\ws\labscode\exweb\images** (click *OK*), and select all the GIF files in that directory. Click *Finish* to import all the images.
6. Select the *webApplication* folder and *File -> Import -> File system*, *Browse* to **c:\ws\labscode\exweb\web** (click *OK*), and select the **PartList.html** and **PartList.jsp** file in that directory. Click *Finish* to import the files.

   Note: The PartList.hmtl and .jsp files must be directly in the *webApplication* folder, not in a *web* subfolder (do not import the web folder, only the files contained in it).
7. Notice the warning and error in the Tasks list; we will fix these later. Double-click on each file to open the Page Designer. Both should display the ITSO image on the top. Close the editors.
8. Select the *source* folder and *New -> Other -> Java -> Java Package*. Click *Next* and enter **itso.wsad.dealer.web** as package name and click *Finish*. The folder structure appears under the source folder.
9. Select the itso\wsad\dealer*web* folder and click the *Create a Java Servlet Class* icon (or *New -> Web -> Servlet*). The package should be preselected. Enter **PartList** as the servlet name, *HttpServlet* as superclass, select *init*, *doPost*, *doGet* methods. Click *Next*.

10. Select *Add to web.xml* (preselected). We use the default display name and URL (PartList). The servlet will be in the deployment web.xml descriptor. Click *Finish*.
11. The Java editor opens for **PartList.java**. Study the skeleton code.

Complete the code

12. Open the file **c:\ws\labscode\exweb\servlet\PartList.txt** in Notepad. We use this code to complete the servlet:
    - Replace *import javax.servlet.http.HttpServlet* with the list of import statement from PartList.txt
    - Complete the empty body of *doPost* with: doGet(request, response);
    - Complete the empty body of *doGet* with the code from PartList.txt
    - Add the new method *getResults* after the doGet method

    Make sure all the brackets match.
13. Save the PartList.java file, there should be no errors. Study the code. The *doGet* method gets the parameter from the HTML file (partialName), prepares a result object (Vector partListResult), calls getResults, stores the Vector in the request block, and calls the JSP (PartList.jsp).

    The *getResults* method connects to the database, executes the SELECT statement, and stores the values of the columns of one row in an array that is added to the result Vector. The JSP can retrieve all the values from the Vector.
14. Open the HTML file (**PartList.html**). Select the form (dotted line) and Attributes (context). Enter **PartList** as the Action, replacing the XXXXX. The HTML file now invokes the PartList servlet. Save and close the editor.
15. Open the JSP file (**PartList.jsp**). Select the Source view and study the code. The results are retrieved from the *partListResult* bean and placed into a table. However, the declaration of the bean is missing:
    - Delete the text “-- the bean from the servlet goes here --”.
    - Switch to the Design view. The cursor should be in front of the first JSP tag icon, after the title. Select *JSP -> Insert Bean* (there is also a small bean icon you can use).
    - Enter **partListResult** as ID, **java.util.Vector** as Type, and **request** as Scope (it has to match what the servlet stores in the request block). Click *OK* and switch to the Source view to see the <jsp:useBean> tag.

    Save the JSP. The error in the Tasks list disappears.
16. Close all the editors.

Preparing a server for testing

17. Open the Server Perspective (*Perspective -> Open -> Other -> Server*).
18. Create a new Server project (*New -> Server -> Server Project*), click *Next*, enter **ItsoWsServer** as name, click *Finish*.
19. Select the ItsoWsServer project and *New -> Server Instance and Configuration*. Enter **ItsoWsDealer** as server name. For instance type, expand WebSphere Servers and select *WebSphere v4.0 Test Environment*. Click *Next*, leave port 8080, click *Finish*.
20. Expand the ItsoWsServer project. The configuration and instance also appears in the Server Configuration view (bottom left) and in the Servers view (bottom right).
21. Edit the configuration properties: either double-click the **server-cfg.xml** file in the Navigator view, or double-click the **ItsoWsDealer** configuration in the Server Configuration view. Go through the pages.
22. The only change we have to perform is in the **Datasource** view. Select the *Db2JdbcDriver* and click *Edit*. Check that the class path points to the db2java.zip file in the DB2 installation directory, for example **D:\SQLLIB\java\db2java.zip** (or wherever DB2 is installed). Click *OK*, save and close the configuration.
23. Select the ItsoWsDealer configuration and *Add Project -> ItsoWsDealerEAR*. We associate our project to this server.
24. Select *Window -> Preferences* and select *Server*. The first check box determines if the server runs in normal or debug mode when you select *Run on Server*. For now we leave debug mode; we can always start the server manually in normal or debug mode.

Test the Web application

25. Start the server in the Servers view (use the start icon, or the context menu). Watch the Console view until the server is open for e-business. As part of the messages you should see that the Web module is loaded, because we associated the project with this server.
26. Expand the ItsoWsDealerWeb project, select the PartList.html file, and *Run on Server* (context). A Web browser view opens with the HTML file. Enter **IRRO** as partial name, click *Retrieve*. The servlet is invoked (see Console), then wait for the JSP to compile and display the results.
27. Go back in the browser and enter other values (L, X, T).

Enhancement (optional)

28. Display the image belonging to a result row. Edit the **PartList.jsp** file. Change the display of the last column (row[4]) to:

```
<TD> <img src="images/<%= row[4] %>"> </TD>
```

    Save the JSP and rerun the sample in the browser view.

29. Close the browser.

Using the Database wizard

30. In the Web Perspective, select the *source* folder, and create a new Java package named **itso.wsad.dealer.dbapp** (*New -> Other -> Java ...*).

31. Click the *Create Web pages that access database fields* icon (or *New -> Web -> Database Web Pages*) to start the Database wizard.

32. For the destination folder, select *ItsoWsDealerWeb/webApplication* (click *Browse*). For the package name, select *itso.wsad.dealer.dbapp*. For Web Pages, select both input and details form. For Model, select *View Bean* to generate servlet and JSP. Select store results in request. Click *Next*.

33. Skip the panel Choose an existing select statement (click *Next*). On Specify SQL statement information, select *Be guided through creating an SQL statement*. Select *Use existing database model* and click *Browse* to locate the ItsoWsDealerDatabase project (expand) and select the ITSOWSAD database and click *OK*. Click *Next*.

34. Construct the SQL statement:

    – Select the tables MMINVENTORY and MMPARTS, click >.

    – Columns: MMPARTS: partnumber, name, description, image_url
    INVENTORY: itemnumber, quantity, cost, shelf, location

    – JOIN: join the tables on part number (drag the partnumber field)

    – Condition: column quantity, operator **<**, value **:quantity** (use drop-downs for column and operator, type :quantity into the field)

    – ORDER: mmparts.partnumber, ASC

35. Click *Next*. The SQL statement is complete. You can *Execute* it with 20 as value. Click *Next*.

36. Connection: select *Use data source connection* and enter **jdbc/ITSOWSAD** as JNDI name. Click *Next*.

37. Input form: set label to Quantity, set size and max length to **4**. Click *Next*.

38. Result table: Select only *part number, name, quantity, cost*, deselect others. Change the labels for the four columns to Partnumber, Name, Quantity, Cost (Use the *Enter* key, or click on another property). Click *Next*.

39. Details: select all columns. Optionally change the labels. Click *Next*.
40. Specify Front Controller and View Bean Choices: take the defaults to have a controller servlet and view beans generated. Click *Next*.
41. Select **Inventory** as prefix. Click *Finish*. The generated servlets (three files) appear in the source package folder and the HTML (one file) and JSPs (two files) in the webApplication folder.

Configure data source and test

42. In the Server Perspective, edit the **ItsoWsDealer** configuration. In the Datasource view, click *Add* for data sources. Enter **ITSOWSAD** as name, **jdbc/ITSOWSAD** as JNDI name, and **ITSOWSAD** as database name. Click *OK*. Save and close the configuration.
43. Start or restart the **ItsoWsDealer** server. Wait until it is ready.
44. Select the InventoryInputForm.html file and *Run on Server* (context). The browser view displays the input form. Enter a value (20) and click *Submit*. It takes a while but the result rows should display. Select a row and click *Details*.
45. Optionally, tailor the HTML and JSP layouts using the Page Designer. For example, display the GIF image instead of the image_url text. (Use a technique similar to step 28.)
46. Stop the ItsoWsDealer server.

Export Web application as WAR file

47. In the Web Perspective, select the ItsoWsDealerWeb project and *Export WAR* (context). Specify an output directory and file (d:\itsowsdealerweb.war) and click *Finish*.

    If you get an error message, rebuild the project (Web perspective, *Rebuild Project* from context) and validate it (*Validate Project* from context).

Using the Database wizard and generate JSPs (optional)

48. Redo the database example, but choose **Tag library** for the model. This generates a solution using JSPs and no servlets.

    Note: Use = as comparison operator. Smaller < and greater > result in JSP parser/compiler errors, although the code does work.

Debugging JSPs (optional)

49. Set some breakpoints in the **PartList.jsp** file (open the JSP source view, select a line with a JSP tag, and in the line prefix select *Add breakpoint* from context).
50. Start the ItsoWsDealer server in debug mode.
51. Select the PartList.html file and *Run on Server*.
52. Enter a partial partname (RR) and step through the JSP code in the Debug view. Study variable values in the Variables view.
53. Stop the server when done. Remove the breakpoints in the JSP and close it.

What you did in this lab

- Defined a Web project with an EAR project
- Imported an existing Web application and used the Page Designer
- Configured a WebSphere test server and ran the Web application
- Used the Database Wizard to create HTML, JSPs, and servlets
- Configured a DataSource and ran the generated application
- Exported a Web application to a WAR file

5

EJB development

What this exercise is about

In this lab we set up a Java project in WebSphere Studio Application Developer and implement an entity and a session enterprise bean.

User requirement

Update stock in the inventory.

What you should be able to do

At the end of this lab you should be able to:

- Define entity beans and map to a database table
- Define session beans with business methods
- Configure a server to test enterprise beans
- Use the EJB test client to verify the functionality

Introduction

We implement an entity bean to interact with the inventory table. Then we implement a session bean with business methods to update the stock through the entity bean.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Define an EJB project

2. Define a new project (*New -> EJB -> EJB Project*) with the name **ItsoWsDealerEJB**. Use the default location. For the EAR project name, select **ItsoWsDealerEAR**. Click *Next* twice. Preselected is *Use source folders contained in the project* (ItsoWsDealerEJB/ejbModule). Leave the default output folder (ItsoWsDealerEJB/bin). Click *Finish*. This creates the EJB project and attaches it to the existing EAR project.
3. The J2EE perspective is opened automatically. You can expand the enterprise applications, Web modules, EJB modules, server configurations and instances, and databases.

Create an entity bean

4. Create a new Java package named **itso.wsad.dealer.ejb** under the project folder ItsoWsDealerEJB/ejbModule (from the Navigator or J2EE view).
5. In the J2EE view, click the *Create an enterprise bean* icon in the toolbar (or *New -> EJB -> Enterprise Java Bean*). Select *CMP* as type, enter *ItsoWsDealerEJB* as project and **Inventory** as bean name,*ejbModule* as source folder, and **itso.wsad.dealer.ejb** as package. Click *Next*. The bean class becomes itso.wsad.dealer.ejb.InventoryBean.
6. Leave the defaults for the interface names and key class. For persistence fields, click *Add*. Enter the following fields:
   - **itemNumber**, java.lang.Long, key field, click *Add*
     (do not use “long”, it must be java.lang.Long)
   - **partNumber**, String, access with getter/setter, promote to remote interface, make getter read-only, click *Add*
   - **quantity**, int, getter/setter, promote, read-only, click *Add*
   - **cost**, **java.math.BigDecimal**, getter/setter, promote, read-only, click *Add*
   - **shelf**, String, getter/setter, promote,read-only, click *Add*
   - **location**, String, getter/setter, promote, read-only, click *Add*
   - Click *Close*.
7. Select *Use the single key attribute type for the key class*. Click *Next*.
8. For import statements, click *Add Package*, and enter/select *javax.ejb*, click *OK*. Select *Add Type* and *java.math.BigDecimal*, click *OK*.

   Note: Do not select com.ibm.math.BigDecimal.

9. Click *Finish* to create the Java code for the bean, key, home and remote interface. The inventory bean with CMP fields and classes appears in the J2EE view under the EJB Modules. The Java source code for the classes is there as well.

Editing the bean

10. Double-click on Inventory (not the InventoryBean.java); no editor opens. You can only edit the whole module. Double-click on the **ItsoWsDealerEJB** JAR with beans. This opens the ejb-jar.xml file (the EJB JAR). Go through the different pages, then in the *Bean* page, select the Inventory bean. You could make updates here. Close the editor.

Complete the bean with create and business methods

11. Several columns are mandatory in the database and we must make sure that the values are not null when an instance is created. Open an editor (double-click) for the InventoryBean (Java code). The code changes are in the file **c:\ws\labscode\exejb\bean\InventoryBean.txt** for copy/paste.
12. Find the **ejbCreate** method. Change the code to:

```
public java.lang.Long ejbCreate(java.lang.Long itemNumber,
      String partNumber, int quantity, BigDecimal cost)
      throws javax.ejb.CreateException {
   _initLinks();
   this.itemNumber = itemNumber;
   this.partNumber = partNumber;
   this.quantity = quantity;
   this.cost = cost;
   this.shelf = null;
   this.location = null;
   return null;
}
```

13. Find the **ejbPostCreate** method. The parameters must match ejbCreate:

```
public void ejbPostCreate(java.lang.Long itemNumber, String partNumber,
    int quantity, BigDecimal cost) throws javax.ejb.CreateException {
}
```

14. Add business methods to remove and add stock at the edn of the class:

```
public int removeStock(int amount) throws InsufficientStockException {
   if (quantity < amount) throw
      new InsufficientStockException("Insufficient stock");
   quantity -= amount;
   return quantity;
}
public int addStock(int amount) {
```

```
        quantity += amount;
        return quantity;
    }
```

15. Save the code. You get errors because the InsufficientStockException class is missing and because the home does not match the ejbCreate method.
16. Import the InsufficientStockException class. Select *File -> Import -> File system*, click *Browse* to locate **c:\ws\labscode\exejb\bean**, select the InsufficientStockException.java file. Import into the folder *ItsoWsDealerEJB\ejbModule\itso\wsad\dealer\ejb*. Some errors disappear.

    Note that you cannot see the InsufficientStockException class in the J2EE view; you have to select the Navigator view to see non-EJB classes.

Home and remote interface

17. With the InventoryBean in the editor, select the two methods *removeStock* and *addStock* in the Outline view, and *Enterprise Bean -> Promote to Remote Interface* (context). This adds the methods to the remote interface; open the **Inventory** interface and check that the methods were added. Also a small *R* icon is added to the method in the outline view.
18. Select the *ejbCreate* method in the Outline view and *Enterprise Bean -> Promote to Home Interface* (context), where a matching create method is added. Open the **InventoryHome** interface. Delete the old create method (with one parameter) and add the import statement:

    ```
    import java.math.BigDecimal;
    ```

19. Save the code. If errors still show in the Tasks view, make a dummy change to the InventoryBean class and save again. The errors disappear. Close the editors.

Create the mapping to the database table

20. Switch to the Data Perspective. In the DBExplorer view, reconnect the **ConITSOWSAD** connection (context menu). (This connection was created in step 4 in Exercise 2, “Relational Schema Center”.)
21. Select the ITSOWSAD database and *Import to Folder* (context). Select the **ItsoWsDealerEJB** project, click *OK*. The ejbModule\META-INF\Schema folder is automatically selected. Click *Finish*. In the Confirm target dialog, click *Yes*, to import the schema.
22. Switch to the J2EE Perspective, Navigator view, and you can see the imported schema in the ejbModule\META-INF folder.

23. Select the ItsoWsDealerEJB project and click the *Create an EJB to RDB mapping* icon in the toolbar (or *Generate -> EJB to RDB Mapping* from the context).
24. In the create mapping dialog, select *Meet In The Middle*. Select *Open mapping editor after completion*. Click *Next*. Select *Match by Name*, click *Finish*.
25. The mapping editor opens with the **Map.mapxmi** file. Expand the Inventory EJB (left) and MMINVENTORY table (right). Nothing is matched because the bean name (Inventory) does not match the table name (MMINVENTORY).
26. Map the **Inventory** bean to the **MMINVENTORY** table by drag/drop:
    - Drag the Inventory bean to the MMINVENTORY table
    - Drag bean attributes to matching columns (or columns to attributes)
27. Save the file and close the editor. The mapping file is only visible in the Navigator view in the META-INF directory.

Generate deployed code

28. In the J2EE view, select the ItsoWsDealerEJB module and *Generate -> Deploy and RMIC Code*. Select the Inventory bean and click *Finish*. The generated classes are visible in the Navigator view.

Bind the container to a DataSource

29. In the J2EE view, select the ItsoWsDealerEJB module and *Open With -> EJB Extension Editor* (context). Go through the pages, and select the *Binding* tab.
30. Select the *ItsoWsDealerEJB* and set the JNDI name for the DataSource to **jdbc/ITSOWSAD**, and user ID and password to **db2admin** (or the userID used to install DB2). We could also set the JNDI name for the Inventory EJB itself; for an empty name, a default name of InventoryHome will be used.

    Note: You must have completed step 42 in Chapter 4, “Web development” to define the jdbc/ITSOWSAD data source for the ItsoWsDealer server.
31. Save and close the editor.

Testing the inventory bean

32. Switch to the ServerPperspective and start the **ItsoWsDealer** server. (The EAR file is already attached to that server.) Wait for the server to be ready.
33. In the Navigator, select the ItsoWsDealerEJB project and *Run on Server* (context). This opens the universal test client (UTC) in a browser view.

34. Select the JNDI Explorer. You can enter a JNDI name, or select from the list. Click on the *InventoryHome*.
35. Expand *EJB References -> Inventory -> InventoryHome*. Click on the *findByPrimaryKey* method. Enter 21000003 as key value, click *Invoke*. The EJB object is added to the bottom pane. Click *Work with Object* and the Inventory object is added to the EJB References (left).
36. Invoke some get methods of the *Inventory* object. The result is shown in the bottom pane. After executing the *getCost* method, click on *Work with Object* and the BigDecimal value is added to Object References (left).
37. Invoke the *addStock* and *removeStock* methods. Try to remove more than the quantity and you get the InsufficientStockException.
38. Invoke the *create* method on the home. First set the BigDecimal value (last). Click on the constructor icon (right), select the BigDecimal(String) constructor, enter 22.22 as value, and *Invoke and Return*. Then set the other parameters as 33000007, M100000002, and 7. Note that the partNumber (M100000002) must exist in the database.
39. Once BigDecimal objects have been added to the references, you can also select values from the pull-down instead of using a constructor.
40. Close the test client browser and stop the server.

Creating a session bean (optional)

41. In the J2EE view, create a new enterprise bean named **StockUpdate**, of type Session. Click on *Packages*, select the itso.wsad.dealer.ejb package, then overtype ejb with session to make it **itso.wsad.dealer.session**. Click *Finish*.
42. Edit the **StockUpdateBean** to complete the code with business methods. Open the file **c:\ws\labscode\exejb\bean\StockUpdateBean.txt** and use it to copy/paste:
    - import statements
    - a variable: private InventoryHome inventoryHome = null;
    - in ejbCreate: inventoryHome = getHome();
    - new methods: addStock, removeStock, moveStock, getHome
43. Save the class. Promote the addStock, removeStock, moveStock methods: select the methods in the Outline view and *Enterprise Bean -> Promote to Remote Interface* (context). Close the editor.
44. Edit the **StockUpdate** interface. Several errors:
    - add: import javax.ejb.*;
    - add: import itso.wsad.dealer.ejb.*;

    Save and close the editor.

45. Generate the deployed code. In the J2EE view, select the EJB project and *Generate -> Deploy and RMIC Code*. Select only the StockUpdate bean (we did not change the Inventory bean).

46. In the J2EE perspective and view, select the ItsoWsDealerEJB module and *Open With -> EJB Editor* (context). Go to the EJB Reference page. Select the StockUpdate bean, click *Add*. The session bean uses the local reference java:comp/env/ejb/Inventory in the getHome method to find the Inventory bean. We have to specify this reference for later binding to a real JNDI name. In the dialog enter:

    – ejb/Inventory (for the name)
    – Entity (for type)
    – itso.wsad.dealer.ejb.InventoryHome (for home)
    – itso.wsad.dealer.ejb.Inventory (for remote)
    – Inventory (for link).

    Save and close.

47. Select the ItsoWsDealerEJB module again and *Open With -> EJB Extension Editor* (context). On the Bindings page, the JNDI name for the data source should still be jdbc/ITSOWSAD. Expand the ItsoWsDealerEJB, select each bean, and set the JNDI name to the names **itso/wsad/dealer/Inventory** and **itso/wsad/dealer/StockUpdate**. (Do not set data source JNDI names for the beans.)

    Set the reference under StockUpdate to **itso/wsad/dealer/Inventory** so that the session bean can find the inventory entity bean in the getHome method. Save and close the extension editor.

Test the session bean (optional)

48. Switch to the Server Perspective. Restart the ItsoWsDealer server.

49. Select the ItsoWsDealerEJB project and *Run on Server* (context).

50. Use the JNDI Explorer to locate the **itso/wsad/dealer/StockUpdate** home. Invoke the *create* method, then work with the object. Invoke the business methods for item numbers 21000003, 21000004, 33000007. Check the results using the InventoryHome.

Add a servlet and HTML file

This section requires that the ItsoWsDealerWeb project was created and the images were imported (see steps 2 through 5 in Exercise 4, “Web development”).

51. Switch to the Web Perspective, and click the *Create a Java Servlet Class* icon in the toolbar:

- For the folder *Browse* to **/ItsoWsDealerWeb/source**
- For package enter: **itso.wsad.dealer.ejbweb**
- For servlet name enter: **InventoryControl**
- For superclass select HttpServlet
- For methods: init, doGet, doPost
- Click *Next*.

52. Select *Add to web.xml* (leave the default names) and click *Finish*.
53. Edit the servlet InventoryControl.java.
54. Replace the code with the code from **c:\ws\labscode\exejb\servlet\InventoryControl.txt** or **c:\ws\labscode\exejb\servlet\InventoryControlNoSession.txt** and *Save*.

    Note: If you did not implement the session bean, use the file **c:\ws\labscode\exejb\servlet\InventoryControlNoSession.txt** to replace the servlet code.

55. The references to the EJB package are not resolved. Open the properties of the ItsoWsDealerWeb project (select *Properties* from the context), and select the Java Build Path. On the projects page mark the *ItsoWsDealerEJB* project and click *OK*. The errors should be fixed.
56. Select the webApplication folder and *File -> Import -> File system*. Click *Browse* to locate **c:\ws\labscode\exejb\servlet** and select the **InventoryControl.html** file. Click *Finish*.
57. Open the file and study the form and the possible actions, and then study the InventoryControl servlet to understand the processing. Notice the init method where the homes of the EJBs are acquired. Note that the homes are retrieved by global JNDI names, but the code for local JNDI names is in comments.

    To use local JNDI names you have to edit the **web.xml** file, go to the References page, and add two references:

    - **ejb/Inventory** (Entity, itso.wsad.dealer.ejb.InventoryHome, itso.wsad.dealer.ejb.Inventory, itso/wsad/dealer/Inventory)
    - **ejb/StockUpdate** (Session, itso.wsad.dealer.session.StockUpdateHome, itso.wsad.dealer.session.StockUpdate, itso/wsad/dealer/StockUpdate)

    The binding information between local and global JNDI names is stored in the ibm-web-bnd.xmi file.

58. Close the editors.
59. ISelect the **ItsoWsDealerWeb** project and *Edit Module Dependencies* (context menu). Select the ItsoWsDealerEJB.jar file and click *Finish*. This

makes sure that the Web application can access the EJBs in the server where the application is deployed.

Run the servlet application

60. Select the InventoryControl.html file and *Run on Server* (context). The ItsoWsDealer server should start (if it not already running). You can also start the ItsoWsDealer server first and then run the HTML file.
61. Test the different actions.
62. Stop the server.

What you did in this lab

- Defined an EJB project as part of an EAR project
- Defined an entity bean and mapped it to an existing database
- Defined a session bean that uses the entity bean
- Defined business methods
- Tested the EJBs in the built-in server
- Imported and run a servlet application that uses the EJBs.

6

Test and deploy using WebSphere AEs

What this exercise is about

In this lab we test an application on a real WebSphere Application Server Single Server (AEs).

User requirement

Run the application in a real WebSphere environment.

What you should be able to do

At the end of this lab you should be able to:

- Define a server in WSAD to publish to AEs
- Test a Web application and EJBs on AEs
- Setup and configure WebSphere Application Server AEs
- Deploy an application and install it in AEs

Introduction

We prepare AEs and WSAD for test and deployment.

Exercise instructions

1. Start WebSphere Studio Application Developer.
2. Make a copy of the WebSphere AEs server configuration file: **d:\WebSphere\AppServer\config\server-cfg.xml** ==> server-cfgSAVE.xml
3. Check in Services that the IBM Agent Controller is started.

Prepare Web application dependency

4. In the Web Perspective, select the **ItsoWsDealerWeb** project and *Edit Module Dependencies* (context menu). Check that the ItsoWsDealerEJB.jar file is selected (if not, selected the file) and click *Finish*. This makes sure that the Web application can access the EJBs in the server where the application is deployed.

Configure a server for remote testing in WebSphere AEs

5. In the Server Perspective, click the *Create server instance and configuration* in the toolbar (or *New -> Server -> Server Instance and Configuration*).
6. Enter **ItsoWsAEs** as server name, **ItsoWsServer** as project folder, select *WebSphere v4.0 Remote Server* as instance type, click *Next*.
7. Enter **127.0.0.1** (or **localhost**) as host address, **d:\WebSphere\AppServer** as WebSphere installation directory, and select *Use default WebSphere deployment directory*. This sets the deployment directory as **d:\WebSphere\AppServer**. Click *Next*.
8. Select *Create a new file transfer instance*, and *Copy File Transfer Mechanism*, click *Next*.
9. Enter **ItsoWsServer** as project folder, ***your-host-name*** (or **localhost**) as remote file transfer name, and **d:\WebSphere\AppServer** as remote target directory. Click *Next*.
10. Change the port from 8080 to **9080** to match AEs. Click *Finish*.
11. You should find the new configuration and instance in the Server Configuration View, and the your-host-name.rft file in the Navigator view.
12. Double-click the ItsoWsAEs instance to open the editor (you can make changes here). Close the editor.
13. Double-click the ItsoWsAEs configuration to open the editor:
    - On the General page, select *Enable administration client* (you can open the Administrative Console while testing).
    - On the EJB page, *Enable the EJB test client* should be preselected.

– On the Datasource page, edit the Db2JdbcDriver and set the class path to **D:/sqllib/java/db2java.zip**. Click *OK*. Then click *Add* for a new data source and enter **ITSOWSAD** (name), j**dbc/ITSOWSAD** (JNDI name) **ITSOWSAD** (database name). Click *OK*.

– Save the changes and close the editor.

14. Select the ItsoWsAEs configuration, and *Add Project -> ItsoWsDealerEAR* (context). This will publish the application when we start the server.

Test the applications in the remote AEs server

15. In the Servers view, select the *ItsoWsAEs* server, and *Start* (context or icon). Watch the console; you should see the EJB and the Web module loaded without errors, and the message:

    ```
    Server Default Server open for e-business
    ```

    Note that the IBM Agent Controller must be running on the machine to start a server from WSAD.

16. Select the ItsoWsDealerWeb project and *Properties*, and in *Server Preference*, change the default server from ItsoWsDealer to **ItsoWsAEs**. This enables launching the Web browser from files.

17. Select the **PartList.html** file and *Run on Server*. The browser opens with http://127.0.0.1:9080/ItsoWsDealerWeb/PartList.html and you can test the application. Select the **InventoryInputForm.html** file and test it, as well.

18. Enter **http://127.0.0.1:9080/UTC** to start the EJB test client. Test the EJBs. (You could also change the default server launcher for the ItsoWsDealerEJB project and then *Run on Server* for the project.) Select the InventoryControl.html file and test the servlet with EJB access.

19. Start the Administrator Console with **http://127.0.0.1:9090/admin**. Login with your user ID, click *Submit*. You can browse the configuration.

20. Close the browser and stop the ItsoWsAEs server from the Servers view.

21. Change the default server launcher for the ItsoWsDealerWeb and ItsoWsDealerEJB projects back to the ItsoWsDealer server (in the Properties Server Preference menu).

Prepare WebSphere AEs for deployment of applications

22. When we started the ItsoWsAEs server from WSAD, it changed the original AEs server configuration file d:\WebSphere\AppServer\config\server-cfg.xml, and we have to restore the original. WSAD created a **wasTools_bkup** subdirectory with the original configuration file.

Copy wasTools_bkup\server-cfg_bk_xxxxxx.xml to the config directory, delete the server-cfg.xml file, and rename the server-cfg_bk_xxxxxx.xml as server-cfg.xml.

You could also copy from the saved file server-cfgSAVE.xml.

23. Start WebSphere AEs, either:
    - From a command window: **startserver**
    - From *Start -> Programs -> IBM WebSphere -> Application Server V4.0 AES -> Start Application Server*
24. Wait for the message:

```
WSPL0057I: The server Default Server is open for e-business
```

25. Start the Administrative Console from *Start -> Programs -> IBM WebSphere -> Application Server V4.0 AES -> Administrator's Console*.
26. Login with your user ID, click *Submit*.
27. Expand *Resources -> JDBC Drivers*, select Db2JdbcDriver. Set the server class path to **d:/SQLLIB/java/db2java.zip** (or where DB2 is installed) and click *OK*.
28. Expand Db2JdbcDrivers, select *Data Sources*. Click *New* to define a new data source. Enter **ITSOWSAD** as name, **jdbc/ITSOWSAD** as JNDI name, ITSO Data Source as description, **ITSOWSAD** as database name, **db2admin** as user ID and password (or the userID of DB2 installation). Click *OK* at the bottom.
29. At the top of the panel you should see the message Configuration needs to be saved. Select the message (or select *Save* in the menu bar). Click *OK* in the save panel.
30. Expand *Nodes -> yournode -> Enterprise Applications*, and you should see some samples that are preinstalled. Expand *Application Servers -> Default Server -> Web Container*, select *HTTP Transports*, and you should see three ports: 9080 is the default built-in HTTP Server, 9090 is the Administrative Console (look at your browser address field: http://localhost:9090/admin), and 9443 is for SSL. (We defined the server in WSAD that uses port 9080 to run directly with the HTTP server of AEs.)
31. Expand the *EJB Container* and you should see some installed sample EJBs.

Deploying an enterprise application to AEs

32. In WSAD, export the EAR file by selecting *File -> Export -> EAR file*, click *Next*. Select the **ItsoWsDealerEAR** resource, and **d:\itsowsdealer.ear** as target directory and file. Click *Finish*.
33. In the AEs Administrator Console, *select Enterprise Applications*. Click *Install*.

34. For the path, click *Browse* to locate the **d:\itsowsdealer.ear** file you exported. Click *Next*.

35. For EJB JNDI names, the names we set in the extension editor are displayed:

    ```
    itso/wsad/dealer/Inventory
    itso/wsad/dealer/StockUpdate (if the session bean was created)
    ```

    Click *Next*.

36. If you defined the session bean (Exercise 5, “EJB development” on page 329): for EJB Reference mapping, the reference from the session bean to the entity bean is filled properly.

    If you defined the references from the servlet (step 57 in Exercise 5, “EJB development” on page 329), then the global JNDI names are filled, as well. Click *Next*.

37. Database is DB2UDBWIN_72, and the datasource JNDI name should be jdbc/ITSOWSAD with db2admin (or your own user ID) as user ID. There is no need to set JNDI names for individual EJBs; click *Next*.

38. Click *Next* again; the defaults are fine.

39. Remove the check mark from *Re-deploy*. WSAD has deployed everything already. Click *Next*, review and click *Finish*.

40. If the IBM HTTP server is used instead of the built-in server, then the plug-in must be regenerated and the HTTP Server stopped and started.

41. Save the configuration and exit the console.

42. Perform **stopserver** and **startserver** to activate the enterprise application.

43. Open a regular browser window and enter:

    ```
    http://yourhostname:9080/ItsoWsDealerWeb/PartList.html
    http://yourhostname:9080/ItsoWsDealerWeb/InventoryInputForm.html
    http://yourhostname:9080/ItsoWsDealerWeb/InventoryControl.html
    ```

Installing the universal test client in AEs (optional)

44. Note that you cannot test the EJBs using the universal test client, because the test client is not installed in WebSphere AEs.

45. You also have to set the module visibility for the Default Server so that the test client can see other Web applications. Start the server (if not started). Use the administrative console (expand Node to the Default Server). Change the module visibility value from APPLICATION to COMPATIBILITY and save the configuration. Exit the administrative console.

46. Stop the server (stopserver command).

47. To install the universal test client in WebSphere AEs, copy the **IBMUTC.ear** directory:

```
from: WSAD\plugins\com.ibm.etools.websphere.tools\IBMUTC
to:   d:\WebSphere\AppServer\installableApps
```

and run the command:

```
seappinstall -install d:\websphere\appserver\installableApps\IBMUTC.ear
-expandDir d:\websphere\appserver\installedApps\IBMUTC.ear -ejbDeploy false
-interactive false
```

48. Start the server (startserver command), and http://localhost:9080/UTC should bring up the test client.

Stop the AEs server

49. To enable further testing in WSAD, stop AEs (stopserver command).

What you did in this lab

- Defined a server in WSAD to publish and test with WebSphere Application Server Single Server (AEs)
- Tested an application from WSAD in AEs
- Configured AEs for deployment
- Deployed and installed an application from WSAD into AEs

7

Profiling an application

What this exercise is about

In this lab we trace a Web application using the profiling tool.

User requirement

Analyze where most time is spent in a Web application.

What you should be able to do

At the end of this lab you should be able to:

- Configure a server for tracing
- Start and stop a trace
- Analyze the trace results

Introduction

We trace the dealer Web application and analyze where the most time is spent.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Configure server instance

2. In the Server Perspective, select the **ItsoWsDealer** instance and edit it (double-click). Select *Enable profile server process*.
3. Disable the just-in-time compiler: On the Environment page, click *Add* to define a new variable. Enter **java.compiler** as name, and **NONE** as value. (A value of jitc enables the JIT compiler.)
4. Save and close the editor.

Agent Controller

5. In the Windows Services list, make sure that the IBM Agent Controller is started.

Start the server

6. Start the **ItsoWsDealer** server in the Server Control Panel view. Check the Console for message similar to these:

```
WebSphere AEs 4.0.1 a0130.02 running with process name localhost/
   Default Server and process id 1932
Host Operating System is Windows 2000, version 5.0
Java version = J2RE 1.3.0 IBM build cn130-20010609 (JIT disabled),
     Java Compiler = NONE
```

   Take note of the process ID, as you will need it later.

Configure the host

7. Open the **Profiling** Perspective. In the toolbar, find the *Profile* icon (stop watch) and click the down-arrow and select *Attach -> Java Process*.
8. In the Attach to Java Process dialog, find the process (javaw) with the same ID as noted in the Console log (select *Show all processes* if you cannot see the javaw process with the same ID). Select the process and click **>>** to move the process to the target area. Click *Next*, leave the default names, click *Next*, leave the default filters, click *Finish*.
9. The javaw process shows up in the Monitors view in the ProfileProject with a Profiling object under it. Select the *Profiling* object and *Properties* (context).

Here you could configure which packages should be excluded or included in the trace. We will use the defaults, so click *Cancel*.

Trace an application

10. Select the *Profiling* object and *Start Monitoring* (context).
11. In the Server Perspective, open the *Properties* of the **ItsoDealerWeb** project and check that the server preference is ItsoWsDealer. Expand the project, select the **PartList.html** file, and *Run on Server* (context). Enter **RR** as partial name and click *Retrieve*. Be patient, and wait for the result.
12. Click *Back to the previous page* (left arrow) in the browser, enter **L** as partial name, and click *Retrieve* again. This time, the result arrives much faster.

Trace analysis

13. In the Profiling Perspective, select the *Profiling* object and *Refresh* (context). In the Class Statistics view, click the Update View icon. This action should display:
    - base time in the class
    - cumulative time in the class (includes calls to other classes)
14. Switch to the Execution Flow view (use the icons in the toolbar) to graphically see where the time was spent. Use the zoom buttons to magnify interesting areas of the graph.
15. Switch to the Method Statistics view to see the time spent in each method.
16. Switch to the Heap view to see the objects that exist in the JVM. This view can be helpful to diagnose memory leaks.
17. Switch to the Object References view (you may have to open it) to see objects and the references they hold. References can be the reason that objects are not garbage-collected.

Close down

18. Select the *Profiling* object and *Stop Monitoring* (context).
19. Close the Profiling Perspective (you can optionally save the data as files under the ProfileProject).
20. In the Server Perspective, close the Web browser and stop the ItsoWsDealer server. (You may get an internal error.)
21. Edit the ItsoWsDealer server instance, remove the check mark for *Enable profile server process*, and remove the **java.compiler** variable on the

Environment page (or set the value from NONE to **jitc**). Save and close the editor.

What you did in this lab

- Configured a server for tracing
- Started and stopped a trace
- Analyzed the trace to understand the execution flow and time spent in classes and methods

8

Create a Web Service

What this exercise is about

In this lab we create a Web Service based on a session EJB that uses two entity EJBs to query the parts and inventory tables for available parts.

User requirement

Provide a callable service to clients looking for parts.

What you should be able to do

At the end of this lab you should be able to:

- Create a Web Service from a JavaBean
- Understand the Web Service wizard
- Understand the generated WSDL files, the proxy bean and the sample client
- Test a Web Service with the sample client and proxy bean

Introduction

We use an access bean that wraps the session EJB. The Web Service wizard takes a JavaBean as input. The session bean and the entity EJBs are provided as a base for this lab.

Exercise instructions

1. Start WebSphere Studio Application Developer.

   Make sure that no internal or external (AEs) server is running—stop all servers that are running.

Import an EJB project

2. Create a new EJB project named **ItsoWsManufacturerEJB** which points to a new **ItsoWsManufacturerEAR** project. Click *Next* twice. For Java build settings, on the Library page, click *Add Variable*, click *Browse* and select the **XERCES** variable (WSAD\plugins\org.apache.xerces\xerces.jar). This is the XML parser for Java which is used by the session bean. Leave the Path Extension empty, click *OK* and *Finish*.
3. Select *File -> Import -> EJB JAR file*. Click *Next*, then *Browse* to locate the **c:\ws\labscode\exwscreate\import\itsowsmanufacturerejb.jar** file. For the EJB project, select ItsoWsManufacturerEJB (the EAR project is then preselected as ItsoWsManufacturerEAR). Also select *Overwrite existing resources without warning*. Click *Finish*.

   Note: If you are interested how this EJB JAR file was created, see “Addendum: how the EJB JAR file was created” on page 357 in this exercise.
4. In the J2EE Perspective and view, select the **ItsoWsManufacturerEJB** module. There are two entity EJBs (MmPart, MmInventory - mapping to the MM tables), and one session EJB (PartInquiry).
5. Open the ItsoWsManufacturerEJB module in the extension editor (context). On the Relationship page you can see the 1:m relationship between Part and Inventory.

   To retrieve the inventory items for a part, a getStocks method is generated (role name stocks). Similarly, there is a getThePart method to get the part for an inventory item. You can also see the relationships when expanding the EJBs in the J2EE view.

   On the Binding page you can see the JNDI names of the EJBs specified as itso/wsad/manu/Xxxxx.
6. Open the **PartInquiry** session bean. This session bean retrieves the inventory items for a part and returns a Vector of PartInventory JavaBeans. Open the PartInquiryBean Java file and browse the code. You should find:
   - private variable for the MmPartHome
   - getPartHome method to find the home for the MmPart EJB

- retrievePartInventory method that returns the Vector of PartInventory beans by finding the MmPart bean for the given part number, retrieving all the MmInventory beans for the given part, constructing a PartInventory bean for each one, and adding it to the Vector. This method is the public business method promoted to the remote interface.
- retrievePartInventoryArray method that returns the same result as array of PartInventory beans, by converting the Vector to an array. (This method will not be used in the lab exercise.)

7. In the Navigator view you can also find the **itso.wsad.manu.beans** package with two JavaBeans. One bean is the **PartInventory** bean discussed in the session bean. This bean contains properties for all the attributes of a part and an inventory item.

   The other bean is called **InquireParts** and is the base for our Web Service. It returns the result Vector of the session bean as an XML element. This bean contains:

   - Two private fields, for the PartInquiry session bean home and for an XML document builder
   - getPartInquiryHome, to instantiate the partInquiryHome field
   - getDocumentBuilder, to instantiate the document builder field
   - newElement, which creates a name/value pair for an XML element
   - populatePart, which creates the XML tree for a PartInventory bean
   - retrievePartInventory, which is the public business method that locates and calls the session bean and converts the Vector of PartInventory beans to an XML element by calling populatePart to create the XML tree for each item.

Define a server configuration and instance

8. In the Server Perspective, select the **ItsoWsServer** project and *New -> Server Instance and Configuration*. Enter **ItsoWsManufacturer** as server name and select *WebSphere v4.0 Test Environment* as instance type. Click *Finish*.
9. Edit the configuration properties: double-click the **ItsoWsManufacturer** configuration in the Server Configuration view. In the **Datasource** view, select the *Db2JdbcDriver* and click *Edit*. Check that the class path points to the correct db2java.zip file.

   Click *Add* for data sources. Enter **ITSOWSAD** as name, **jdbc/ITSOWSAD** as JNDI name, and **ITSOWSAD** as database name. Click *OK*. Save and close the configuration.

10. Select the configuration and *Add Project -> ItsoWsManufacturerEAR*.

Create a Web project for the Web Service

11. In the Web Perspective, create a new **ItsoWsManufacturerWeb** Web project, which points to the **ItsoWsManufacturerEAR** project. Click *Next*. Under Module Dependencies, select the ItsoWsManufacturerEJB.jar file. Click *Next*. For Java build settings, on the Library page, click *Add Variable* and select the XERCES variable (d:\WSAD\plugins\org.apache.xerces\xerces.jar).

    Check that the **ivjejb35.jar** file is in the list of JAR files on the Library page. If not, click *Add External Jars* and select the file d:\WSAD**plugins\com.ibm.etools.websphere.runtime\lib\ivjejb35.jar**.

    Click *Finish*.

Copy the server JavaBean from the EJB project

12. In the Web Perspective, select the *source* folder of the new Web project, and create a Java package named **itso.wsad.manu.server**.
13. Select the **InquireParts** in the ItsoWsManufacturerEJB project, ejbModule, itso.wsad.manu.beans folder, and *Copy* from the context menu. Select the ItsoWsManufacturerWeb project, source, itso.wsad.manu.server package as destination.
14. Open the copied InquireParts.java file, and change the package name to itso.wsad.manu.server. Save and close the changed file.

Create the Web Service from the JavaBean

15. In the Web Perspective, select the itso.wsad.manu.server.**InquireParts** bean and *New -> Other -> Web Service -> Web Service*, click *Next*. The Web Service type (JavaBean) and the project (ItsoWsManufacturerWeb) should be preselected. For defaults, select *Start Web service in Web project*, *Generate a proxy*, and *Generate a sample*. Select *Create folders when necessary*. Click *Next*.
16. On the JavaBean selection panel, the InquireParts bean is preselected, click *Next*.
17. On the identity panel, set the Web Service URI to **urn:InquireParts**, and the scope to *Application*. Do not select any of the check boxes. Change the ISD filename to webApplication/WEB-INF/**isd/InquireParts.isd**.

    The WSDL file names are set to:

    ```
    webApplication/wsdl/InquireParts-service.wsdl
    webApplication/wsdl/InquireParts-binding.wsdl
    ```

Change the WSDL schema to:

```
webApplication/wsdl/InquireParts.xsd
```

Click *Next*.

18. On the Java Beans Methods panel, the **retrievePartInventory** method is preselected. Select the method name, leave input encoding as *SOAP* (parameters), and output encoding as *Literal XML* (our output is XML). Select *Show server Java to XML type mappings*. Click *Next*.
19. On the Java to XML mapping panel:
    - Select the String encoding, but leave the default JavaBean mapping.
    - Select the Element encoding, but leave the *Show and use the default DOM Element mapping*.
    - Click *Next*.
20. On the proxy generation panel, select *SOAP Bindings* (preselected). The folder is correct (/ItsoWsManufacturerWeb/source). Change the class to:

    ```
    itso.wsad.manu.proxy.InquirePartsProxy
    ```

    Select *Show mappings* and click *Next*.
21. On the XML to Java mappings panel, select each mapping, but do not change anything. Click *Next*. In the SOAP binding mapping panel leave the defaults, and click *Next*.
22. On the test client panel, do not select *Launch the test client* (this would launch the universal test client). Click *Next*.
23. On the sample generation panel, select *Generate a sample*, but do not select *Launch the sample*. Leave the target JSP folder as set, and click *Next*.
24. On the publication panel do not select *Launch the UDDI Explorer*.
25. Click *Finish*.
26. Be patient; it takes a while to generate all the code. Also, the ItsoWsManfufacturer server is started.

Generated files

27. In the Web Perspective, expand the ItsoWsManufacturerWeb project, webApplication.
28. The **wsdl** directory contains two generated WSDL files:
    - InquireParts-service.wsdl: implementation - how to invoke
    - InquireParts-binding.wsdl: interface - how to connect

29. For publishing of the Web Service to a UDDI registry, unique names must be assigned to services; we will use:

    `http://www.redbooks.ibm.com/ITSOWSAD/definitions/InquirePartsRemoteInterface`

    Open the **InquireParts-service.wsdl** file. In the Design view, change the values of <definitions ... **xmlns:binding** and import **namespace**:

    ```
    from: http://www.inquireparts.com/definitions/InquirePartsRemoteInterface
    to  : http://www.redbooks.ibm.com/ITSOWSAD/definitions/InquirePartsRemoteInterface
    ```

30. Open the **InquireParts-binding.wsdl** file. In the Design view, change the values of <definitions ... **targetNamespace**, and **xmlns:tns**:

    ```
    from: http://www.inquireparts.com/definitions/InquirePartsRemoteInterface
    to  : http://www.redbooks.ibm.com/ITSOWSAD/definitions/InquirePartsRemoteInterface
    ```

    Change the values of **xmlns:xsd1**, and import **namespace**:

    ```
    from: http://www.inquireparts.com/schemas/InquirePartsRemoteInterface
    to  : http://www.redbooks.ibm.com/ITSOWSAD/schemas/PartInventory
    ```

    Check that the value of <import ... **location** is:

    `http://localhost:8080/ItsoWsManufacturerWeb/wsdl/InquireParts.xsd`

31. Save the WSDL files.

32. The generated file ..\wsdl**InquireParts.xsd** is incomplete and does not contain all the XML elements, because the wizard does not know what the session bean generates as XML output (DOM tree).

    Import the correct schema file. Select the **wsdl** folder and *File -> Import -> File system*, *Browse* to locate **c:\ws\labscode\exwscreate\import** and select only the **InquireParts.xsd** file. Select *Overwrite existing resources* and click *Finish*. This schema defines the XML structure generated by the session bean.

33. Open the **InquireParts.isd** file (in WEB-INF\isd). It is only used to build the dds.xml file. This file defines the JavaBean session bean) and the method to be invoked (retrievePartInventory).

34. Open the **dds.xml** file (in webApplication). It is a concatenation of all .isd files.

35. Open the **soap.xml** file. It instantiates the XMLDrivenConfigManager class that looks for the dds.xml file.

36. Close all the files.

37. Open the **web.xml** file (in WEB-INF). On the Servlets panel you can find two servlets that were added to the Web application: rpcrouter, and messagerouter. The rpcrouter is used in our case, as defined in the InquireParts-service.wsdl file <soap:address> tag. The servlets are implemented in the **soapcfg.jar** file which you can find in the **lib** folder. Close the web.xml file.

View deployed Web Service

38. A Web application to view and configure the Web Service has been generated into the webApplication**admin** folder. Expand the admin folder.
39. Select the **index.html** file, and *Run on Server* (context). Click on *List all services* and the InquireParts service should be listed. The other actions can be used to start and stop Web Services (all the started initially).
40. Click on *urn:InquireParts* to see the details of the service.

Client proxy

41. The client proxy code was generated into source\itso\wsad\manu\proxy as **InquirePartsProxy** file. Open the file and browse the code:
    - A variable defines the URL of the Web Service.
    - The retrievePartInventory method invokes the Web Service using the *Call* class from the org.apache.soap.rpc package. You can see the input parameter and the handling of the response. Close the editor.

Sample client

42. The sample client Web application was generated into the webApplication**sample\InquireParts** folder. There are four files:
    - TestClient.jsp is the frameset for the other three files.
    - Method.jsp lists the available Web Services.
    - Input.jsp provides a form for the input parameter.
    - Results.jsp creates the client proxy and executes the Web Service using the input data. It also contains a *domWriter* method that generates an XML file from the result tree.
43. To run the sample client, select the **TestClient.jsp** and *Run on Server* (context). Be patient; wait until the frames appear in the browser.
44. Click the *retrievePartInventory* method, and the input form should appear.
45. Enter **M100000003** as part number and click *Invoke*. Be patient; wait until the XML output appears in the output frame.
46. Use the **admin** application to stop the Web Service, then try to invoke it from the TestClient. You should get an error message in the output.
47. Start the Web Service again and rerun the TestClient. It should work again.
48. Stop the ItsoWsManufacturer server (or do the optional exercise).

Monitoring a Web Service (optional)

49. In the Server Perspective, create a new server instance and configuration (toolbar icon or *File -> New ->Server Instance and Configuratio*n).
50. Enter **ItsoWsMonitor** as server name, select *TCP/IP Monitoring Server* as instance type, and click *Finish*.
51. Start the ItsoWsManufacturer server (if you stopped it).
52. Start the ItsoWsMonitor server. The console should display:

```
Monitoring server started
localhost:8081 -> localhost:8080
```

53. Add the TCP/IP Monitor view to the Server Perspective using *Perspective -> Show View -> TCP/IP Monitor*.
54. Select the TestClient.jsp (in the sample Web application) and *Run on Server*. You are prompted for the server; select the *TCP/IP monitoring server*.
55. Run the application (with partNumber M100000003), then maximize the TCP/IP Monitor view and click on each request in the left pane to see the input and output streams. The XML response is not easy to see in this format.
56. To see the SOAP requests we have to modify the client proxy. Edit the **InquirePartsProxy** file (in ItsoWsManufacturerWeb\source).
57. Change the URL and save the code:

```
from: http://localhost:8080/ItsoWsManufacturerWeb/servlet/rpcrouter
to  : http://localhost:8081/ItsoWsManufacturerWeb/servlet/rpcrouter
```

58. Rerun the application, then switch to the TCP/IP Monitor view. The last request (ItsoWsManufacturerWeb/servlet/rpcrouter) shows the SOAP request input and output and you can see the XML in a good format.
59. Change the **InquirePartsProxy** file back to port **8080**, save, and close the editor.
60. Close the TCP/IP Monitor view.
61. Stop the ItsoWsMonitor server.
62. Stop the ItsoWsManufacturer server.

What you did in this lab

- Created a Web Service based on a session enterprise bean
- Used the Web Services administrative application
- Tested the Web Service using the generated sample client

Addendum: how the EJB JAR file was created

1. Create a new EJB project named **ItsoWsManufacturerEJB** which points to a new **ItsoWsManufacturerEAR** project. Click *Next*. For Java build settings, on the Library page, click *Add Variable*, click *Browse* and select the **XERCES** variable (WSAD\plugins\org.apache.xerces\xerces.jar). This is the XML parser for Java which is used by the session bean. Click *Finish*.
2. Create packages **itso.wsad.manu.ejb** and **itso.wsad.manu.facade** under the ejbModule folder.

Create CMP entity beans with association

3. Create a new CMP entity bean named **MmPart** in itso.wsad.manu.ejb, with fields: partNumber (String, key), name (String), description (String), weight (double), imageUrl (string). Use single key attribute as the key class.
4. Create a new CMP entity bean named **MmInventory** in itso.wsad.manu.ejb, with fields: itemNumber (java.lang.Long, key), quantity (int), cost (java.math.BigDecimal), shelf (String), location (string). Use single key attribute as the key class.
5. Create an association: select the EJB module and *Open With -> EJB Extension Editor*. On the relationships page, click *Add* and define: **InventoryForPart**, between MmPart and MmInventory, roles **thePart** and **stocks**, navigable, 1..1 and 0..m (1:m). Click *Apply*. Save and close the extension editor.

Complete the code

6. Edit **MmPartBean**.java. In the ejbCreate method, before the return, add:

```
this.imageUrl = "";
```

7. Add method to return the key value:

```
public java.lang.String getPartNumber() { return partNumber; }
```

8. Select the getPartNumber method in the Outline, and *Enterprise Bean -> Promote to Remote Interface*. Save the code.
9. Edit **MmInventoryBean**.java. Replace the ejbCreate method with:

```
public java.lang.Long ejbCreate(java.lang.Long itemNumber,
                                itso.wsad.manu.ejb.MmPart aThePart)
                                   throws javax.ejb.CreateException {
   _initLinks();
   this.itemNumber = itemNumber;
   this.quantity = 1;
   this.cost = new java.math.BigDecimal(0.0);
   try { this.setThePart(aThePart); }
   catch (java.rmi.RemoteException ex)
```

```
            { throw new javax.ejb.CreateException("Create inventory failed
               for part "+aThePart); }
         return null;
      }
```

10. Replace the ejbPostCreate method with:

```
public void ejbPostCreate(java.lang.Long itemNumber,
                          itso.wsad.manu.ejb.MmPart aThePart)
                                   throws javax.ejb.CreateException {}
```

11. Add a method to return the key value:

```
public java.lang.Long getItemNumber() { return itemNumber; }
```

12. Save the code. Select the ejbCreate method in the Outline, and *Enterprise Bean -> Promote to Home Interface*. Select the getItemNumber method in the Outline, and *Enterprise Bean -> Promote to Remote Interface*.
13. Edit **MmInventoryHome**.java. Delete the create method with only one parameter. Save and close.
14. You may have to save the MmInventoryBean again to get rid of the error messages.

Import beans for the Web Service

15. Create a new **itso.wsad.manu.beans** package under the ejbModule folder (Navigator view). Import the two files PartInventory.java and InquireParts.java from **c:\ws\labscode\exwscreate\beans**. The PartInventory bean will be used in the session bean, and the InquireParts bean will be used for the Web Service. (The second file will have errors for now.)

Create session bean

16. Create a session bean named **PartInquiry** in itso.wsad.manu.facade. Click *Finish*.
17. Edit **PartInquiryBean**.java. Replace the code with the content of the file **c:\ws\labscode\exwscreate\ejbcode\PartInquiryBean.java**. This code has the business methods. Save the code.
18. Select the retrievePartInventory and retrievePartInventoryArray methods in the Outline, and *Enterprise Bean -> Promote to Remote Interface*.

Set EJB references

19. Select the EJB module and *Open With -> EJB Editor*. On the References page select the **PartInquiry** bean, click *Add*, and create a new reference: ejb/MmPart, Entity, itso.wsad.manu.ejb.MmPartHome, itso.....MmPart, MmPart (identical to the reference under MmInventory). Save and close.

20. Select the EJB module and *Open With -> EJB Extension Editor*. On the Bindings page: For the ItsoWsManufacturerEJB module, the data source JNDI name must be **jdbc/ITSOWSAD** with **db2admin** as user ID/password.
    - Set the JNDI names for the beans to **itso/wsad/manu/MmPart**, **itso/wsad/manu/MmInventory** and **itso/wsad/manu/PartInquiry**.
    - Set the JNDI names for the references to the same values.
21. Save and close.

Create mapping

22. Select the EJB module and *Generate -> EJB to RDB Mapping*. Select *Meet In The Middle* and *Open mapping editor after completion*, click *Next*.
23. Database connection: Select *Use existing connection* **ConITSOWSAD**, click *Next*. Select tables MMINVENTORY and MMPARTS, click *Next*. Select *Match By Name*, click *Finish* and the mapping editor opens.
24. Complete the mapping by expanding the beans (left) and tables (right) and dragging first the beans (from left) to the matching tables (right), and then the attributes to the columns.

    To map the association: drag the attribute **thePart:MmPart** in the Inventory bean to the **ITEMPART:MMPARTS** column in the INVENTORY table. (This maps both directions of the association.)
25. Save the mapping and close the editor.

Generate deployed code

26. Generate the deployed code by selecting the EJB module and *Generate -> Deploy and RMIC Code*. Select all beans and click *Finish*. The generated classes are visible in the Navigator view.

    You will get an error in EJSJDBCPersisterCMPMmInventoryBean.java. Open the file (double-click the error in the Tasks view) and change the references to **inkey.partNumber** to **inkey**.

Export EJB JAR

27. Create the JAR file by selecting *File -> Export -> EJB JAR file*, click *Next*. Select the ItsoWsManufacturerEJB module and enter the name of an output file (c:\ws\labscode\excreate\import\itsowsmanufacturerejb.jar). Select *Export source files* and click *Finish*.

9

Deploy and test a Web Service

What this exercise is about

In this lab we deploy the Web Service and the sample client to WebSphere Application Server AEs.

User requirement

Deploy the Web Service to a production machine.

What you should be able to do

At the end of this lab you should be able to:

- Configure ports in WebSphere AEs
- Deploy an enterprise application using the SEAppInstall command
- Deploy a Web Service and administer the Web Service
- Run the sample test client

Introduction

We configure AEs for deployment and deploy the enterprise application with the Web Service.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Prepare the Web application

2. In the Web Perspective, expand the **ItsoWsManufacturerWeb** project - *source - itso - wsad - manu - proxy*, and edit the InquirePartsProxy.java file. Notice how the rpcrouter is invoked:

   ```
   http://localhost:8080/ItsoWsManufacturerWeb/servlet/rpcrouter
   ```

3. WebSphere AEs by default listens to port 9080, or port 80 when using the HTTP server. For deployment we can either change the code here, or we configure AEs to also listen on port 8080 (which is what we will do).
4. Close the editor without changing anything.
5. For deployment we require an EAR file. Select *File -> Export -> EAR file*. For resources, select the **ItsoWsManufacturerEAR** project. For the export location, Browse to **c:\ws\labscode\exwsdeploy\EARfiles**, and enter **itsowsmanufacturer.ear** as file name. Click *Finish* to create the EAR file.

Prepare WebSphere AEs for port 8080

6. Make sure that no server runs in WSAD.
7. Check that the original WebSphere AEs configuration file was restored (d:\WebSphere\AppServer\config\server-cfg.xml) if you used remote testing in Exercise 6, “Test and deploy using WebSphere AEs” on page 339.

   (Otherwise, copy wasTools_bkup\server-cfg_bk_xxxxxx.xml to the config directory, delete the server-cfg.xml file, and rename the server-cfg_bk_xxxxxx.xml as server-cfg.xml.)
8. Start WebSphere AEs (startserver command).
9. Open the Administrator’s Console with **http://localhost:9090/admin** (or *Start -> Programs -> IBM WebSphere -> Application Server V4.0 AES -> Administrator’s Console*). Login with your user ID, then click *Submit*.
10. Expand *Nodes -> yournode -> Application Servers -> Default Server -> Web Container -> HTTP Transports*. You should see ports 9080, 9443, and 9090. Click on *HTTP Transports*. Click *New*. Enter ***** as host name, **8080** as port, and click *OK*.
11. Expand *Virtual Hosts -> default_host* and click on *Aliases*. Click *New*. Enter ***** as host name, **8080** as port, and click *OK*.
12. Save the configuration and *Exit* the Console.

13. Stop WebSphere AEs (stopserver command).

Install the EAR file with EJBs and Web applications

14. To install the EAR file, use this command:

```
SEAppInstall
  -install c:\ws\labscode\exwsdeploy\earfiles\itsowsmanufacturer.ear
  -expandDir d:\websphere\appserver\installedApps\itsowsmanufacturer.ear
  -ejbDeploy false -interactive false
```

    You can use the **c:\ws\labscode\exwsdeploy\installManu.bat** file to run this command—but be sure to check the directory names, which may be different than for your installation.

15. Start WebSphere AEs (startserver command).
16. Open the Administrator's Console with **http://localhost:9090/admin** and login with your user ID, then click *Submit*.
17. Expand *Nodes -> yournode -> Enterprise Applications* and you should see the installed ItsoWsManufacturerEAR application.

Testing the deployed Web Service

18. To test the Web Services administration client, open a browser and enter:

```
http://localhost:9080/ItsoWsManufacturerWeb/admin/index.html
```

19. List the services, stop the service, and start the service.
20. To test the Web Service, enter:

```
http://localhost:9080/ItsoWsManufacturerWeb/sample/InquireParts/TestClient.jsp
```

21. Click the *retrievePartInventory* method and enter **M100000003** as part number and click *Invoke.* The output should appear as an XML file.
22. Use the admin application to stop the Web Service, then run the test client again. Restart the Web Service.
23. Stop WebSphere AEs (stopserver command).

What you did in this lab

- Configured AEs with an additional port
- Exported an enterprise application with a Web Service and installed it in WebSphere AEs
- Tested the deployed Web Service using the test client

10

Using a Web Service in a client application

What this exercise is about

In this lab we create a client application that uses the Web Service. In the client, we transform the XML result into HTML using an XSL style sheet.

User requirement

Create a client application to query the manufacturer parts database using the Web Service.

What you should be able to do

At the end of this lab you should be able to:

- Use the Web Service wizard to create a client proxy and sample client application
- Create a real client application that invokes the Web Service

Introduction

We implement an HTML page that invokes a servlet that invokes the Web Service. The resulting XML file is transformed into HTML.

Exercise instructions

1. Start WebSphere Studio Application Developer.

Define a Web project for the client

2. In the Web Perspective, define a new Web project named **ItsoWsClientWeb**, pointing to a new **ItsoWsClientEAR** project. Click *Finish*.
3. Web Service clients can only be created from WSDL files in their own project. In the Web Perspective, create a **wsdl** folder under webApplication.
4. Expand the ItsoWsManufacturerWeb project -> webApplication -> wsdl and select the **InquireParts-service.wsdl** file. Select copy from the context menu, and point to ItsoWsClientWeb\webApplication\wsdl for the destination folder.
5. In the Server Perspective (Configuration view), add the ItsoWsClientEAR to the ItsoWsManufacturer server (*Add Project* in context).

Start the server

6. When creating a client with the Web Service wizard, certain files are retrieved from the server. In the Server Perspective, start the **ItsoWsManufacturer** server. (Note that an external WebSphere AEs server must be stopped.)

Generate the Web Service proxy and sample client

7. In the Web Perspective, select the ItsoWsClientWeb project and *New -> Other -> Web Services -> Web Service client*. The Web project should be preselected, so click *Next*.
8. Web Service file selection: the **InquireParts-service.wsdl** should be preselected, so click *Next*.
9. Web Service proxy generation: select *SOAP Binding*, the folder is set to source, for the class enter **itso.wsad.wsclient.proxy.InquirePartsProxy**. Select *Show mappings*, then click *Next*.
10. XML to Java mappings: leave the defaults, click *Next*.
11. SOAP binding mapping configuration: leave defaults, click *Next*.
12. Test client: do not select *Launch the test client*, click *Next*.
13. Sample generation: select *Generate a sample*, but do not select *Launch the sample*. Click *Finish*.

14. Study the generated classes. The folder source\itso\wsad\wsclient\proxy contains the **InquirePartsProxy** file. This class contains the retrievePartInventory method for the client to invoke the Web Service.

Test the sample client

15. Open the *Properties* (context) of the ItsoWsClientWeb project. Notice in the Java Build Path the JAR files that were added to the Library page.

    Select *Server Preferences* and make the **ItsoWsManufacturer** server the preferred instance (click *Apply*). Close the properties dialog.

16. Select the **TestClient.jsp** (in webApplication\sample\InquireParts) and *Run on Server* (context). Select *Open Web browser* (not the TCP/IP Monitor).

17. Select the **retrievePartInventory** method and run it with a part number of M100000003. It should work.

Build the client application

18. For the client, we want to translate the returned XML document into HTML using the XALAN style sheet processor.

19. Open the properties of the ItsoWsClientWeb project, and in the Java Build Path -> Libraries page, click *Add External JAR file* and add the WSAD\plugins\com.ibm.etools.xalanrt**xalan.jar** file. Click *OK*.

20. Create a new servlet (toolbar icon or *New -> Web -> Servlet*). Select ItsoWsClientWeb/source as folder, enter **itso.wsad.wsclient.servlet** as package, **PartInventoryServlet** as name, javax.servlet.http.**HttpServlet** as superclass, generate doGet and doPost methods. Click *Next*.

21. Select *Add to web.xml* (preselected) and click *Finish*.

22. The new servlet is open in the editor. Copy/paste the code from **c:\ws\labscode\exwsuse\wsservlet\PartInventoryServlet.txt** to complete the servlet source code. Save and close the servlet.

23. Import the HTML and XSL files. Select the webApplication folder and *File -> Import -> File system*, click *Browse* to **c:\ws\labscode\exwsuse\wsclient** and select the folder.

    For the target location, specify ItsoWsClientWeb/webApplication**/wsclient** so that a subfolder is created. Click *Finish*.

Test the client application

24. Restart the ItsoWsManufacturer server. This loads the ItsoWsClientWeb project, as well as the ItsoWsManufacturerWeb and EJB projects.

25. Select the **PartInventory.html** file (in wsclient) and *Run on Server*. Execute the part inquiry with part number M100000003. The servlet is invoked, and it uses the proxy to invoke the Web Service, which invokes the session bean (through the access bean), which uses the entity beans to access the database. The resulting XML file is transformed into HTML using the XSL style sheet.
26. Stop the ItsoWsManufacturer server.

Deploy the client application (optional)

27. Create an EAR file for the ItsoWsClientEAR project (**itsowsclient.ear** into c:\ws\labscode\exwsuse\earfiles).
28. Run SEAppInstall to install the EAR file (use the **installClient.bat** file).
29. Start the AES server.
30. Open a browser with

    `http://localhost:9080/ItsoWsClientWeb/wsclient/PartInventory.html`
31. Enter a part number and run the application
32. Stop the AEs server.

What you did in this lab

- Defined a Web project for the client application
- Used the Web Service wizard to create the base code for the client
- Implemented a client application consisting of an HTML page, a servlet, and an XSL style sheet
- Tested the client application
- Deployed the client application in WebSphere Application Server AEs.

11

Web Service publishing in the UDDI registry

What this exercise is about

In this lab we publish a Web Service to a UDDI registry and retrieve the definition of a Web Service from the registry.

User requirement

Publish a Web Service to make it available and searchable on the Web.

What you should be able to do

At the end of this lab you should be able to:

- Work with the UDDI registry to create a business entity
- Publish a Web Service
- Find published Web Services
- Retrieve the WSDL files of a Web Service

Introduction

We use the IBM Test Registry or the WebSphere UDDI Registry to publish and find Web Services.

Which UDDI registry to use

If you have an Internet connection, you can use the IBM Test Registry. Otherwise, you can install the IBM WebSphere UDDI Registry and work with a registry on your own machine.

Use the URLs listed below to connect to the UDDI registry.

IBM Test Registry

From an external browser:

```
http://www-3.ibm.com/services/uddi/testregistry/index.html
```

From a program (UDDI Explorer) using the UDDI API:

```
http://www-3.ibm.com/services/uddi/testregistry/inquiryapi
https://www-3.ibm.com/services/uddi/testregistry/protect/publishapi
```

IBM WebSphere UDDI Registry

For the beta code available February 2002, the URLs are listed here. For later code, check the product documentation.

From an external browser:

```
http://localhost:9080/uddiguibeta
```

From a program (UDDI Explorer) using the UDDI API:

```
http://localhost:9080/uddibeta/inquiryapi
http://localhost:9080/uddibeta/publishapi
```

Attention: Some of the exercise instructions do not work with the beta code of the WebSphere UDDI Registry.

IBM WebSphere UDDI Registry Preview

The IBM WebSphere UDDI Registry Preview is the predecessor for the beta code, but it is no longer available for the Windows platform.

From an external browser:

```
http://hostname/services/uddi/home.jsp
```

From a program (UDDI Explorer) using the UDDI API:

```
http://hostname/services/uddi/inquiryapi
http://hostname/services/uddi/publishapi
```

Exercise instructions

1. Start WebSphere Studio Application Developer.

Register a user ID and password

2. Connect to the registry using an external browser.
3. You have to register to use the registry and get a user ID and password. Once you have registered, you can login to perform actions. However, we will use WSAD to connect to the registry to perform the actions.

   Note that you can only define one business entity with one user ID in the test registry.
4. Close the browser.

Connecting to the registry

5. In the Web Perspective, select the **ItsoWsDealerWeb** project, and *File -> Import -> UDDI*. Click *Next*, then click *Finish*. Be patient; this starts an internal server and opens a browser with the URL.

   ```
   http://localhost:xxxx/uddiexplorer/uddiexplorer.jsp
   ```
6. For the IBM Test Registry:
   - Click on *IBM Test Registry*, and a number of toolbar icons appear in the right-hand pane.

   For the WebSphere UDDI Registry:
   - Click on *UDDI Main*. Enter **WebSphereUDDI Registry** as name, and the inquiry API URL listed under “Which UDDI registry to use” on page 370.
   - Click *Go*, and the WebSphere UDDI Registry should appear in the Navigator.
   - Click on *WebSphere UDDI Registry*, and a number of toolbar icons appear in the right-hand pane.
   - Click on the *Add to Favorites* icon (the flag at top right) to add this registry to the Favorites, and you never have to enter the URL again.

Creating a business entity in the registry

7. Click on the *Publish Business Entity* icon. This opens a login form where you enter the user ID and password you obtained.

   For the WebSphere UDDI Registry, you also have to enter the publish URL.

Click *Go*.

8. The form to define a business entity opens up. Enter:
   - Name: your name
   - Description: anything
   - Identifiers: click *Add*, and enter **phone number** as key name and any number as value (for example).
   - Categories: click *Add*, select *NAICS*, click *Browse* and select a suitable category, for example, *Retail Trade -> Motor Vehicles and Parts Dealers -> Automotive Parts, Accessories ... -> Automotive Parts and Accessories Stores (44131*). (You may skip this for the WebSphere UDDI Registry.)

   Click *Go*, and the business entity is created.

Publishing a Web Service to the registry

9. To publish a Web Service, the server that runs the Web Service must be started. Start the **ItsoWsManufacturer** server in the Server Perspective.

   Note: If the UDDI browser is not open, you can select the ItsoWsManufacturerWeb project and *File -> Export -> UDDI*, select the **InquireParts-service.wsdl** file, and click *Finish*.

10. Select the business entity in the Navigator. You can also use the search facility (*Find Business Entity*) to locate the business entity.

11. Click the *Publish Business Service* icon. Login if necessary. A form appears. In the implementation URL enter:

    ```
    http://localhost:8080/ItsoWsManufacturerWeb/wsdl/InquireParts-service.wsdl
    ```

    Note: This is prefilled if you started the browser with *File -> Export -> UDDI*.

12. For the description enter anything (for example, ITSO workshop manufacturer parts inventory Web Service).

13. For the categories, click *Add* and locate the 44131 entry. (You may skip this for the WebSphere UDDI Registry.)

14. Click *Go*. The **InquirePartsService** is added to the business entity, and also an interface is added with the name (URL):

    ```
    http://www.redbooks.ibm.com/ITSOWSAD/definitions/InquirePartsRemoteInterface
    ```

    When you click on this interface, you can see the bindings.wsdl file listed.

15. Close the UDDI Explorer. Leave the ItsoWsManufacturer server running.

Finding a Web Service in the registry

16. In the Web Perspective, select the **ItsoWsDealerWeb** project and *File -> Import -> UDDI*, click *Next*, then click *Finish*. The UDDI browser opens.

17. Select the IBM Test Registry or the WebSphere UDDI Registry.
18. Expand the registry entry, and click on *Find Business Entity*.
19. Enter the first letters of your business entity and click *Go*. You should find your business entity in this way.
20. Expand the business entity and click *Find Business Services*. Click *Go* and the InquirePartsService should be found.
21. Expand the InquirePartsService and click *Find Service Interfaces*. Click *Go* and you should see your interface.

Importing a Web Service from the test registry

22. To import the WSDL files, the **ItsoWsManufacturer** server must be running.
23. Select the **InquirePartsService** and click the *Import to Workbench* icon. On the import panel, select the ItsoWsDealerWeb project, then click *Go*.
24. Select the interface (**http://www.redbooks.ibm.com/...**) and click the *Import to Workbench* icon. On the import panel, select the ItsoWsDealerWeb project, then click *Go*.
25. In WSAD you should now see both WSDL files in the ItsoWsDealerWeb project, webApplication folder.
26. With these files we could now implement client applications, as done in the previous exercise.
27. Close the UDDI Explorer and stop the ItsoWsManufacturer server.

Application with dynamic Web Services (optional)

28. In the Web Perspective, select the **ItsoWsClientWeb** project and open the *Properties* (context). In the Java Build Path, Library page, add two variables (click *Add Variables*, click *Browse*, click *New* to define the variables):
    - UDDI4J, pointing to

      `WSAD/plugins/com.ibm.etools.websphere.runtime/lib/uddi4j.jar`
    - MAILJAR, pointing to

      `WSAD/plugins/com.ibm.etools.servletengine/lib/mail.jar`

    Click *OK* to close the properties.
29. Add two new folders to the project:
    - **wsdynamic**, under source\itso\wsad
    - **wsdynamic**, under webApplication

30. Create a new servlet named **DynamicServlet** in itso.wsad.wsdynamic (subclass of HttpServlet) and add it to the web.xml file.
31. Replace the code of the servlet with the code from **c:\labscode\exwsuddi\dynamic\DynamicServlet.java**, and save the file.
32. Import the files **UddiServiceImplementer.java** and **UddiTestList.java** from i**c:\labscode\exwsuddi\dynamic** into the itso.wsad.wsdynamic folder.
33. Import the files **DynamicPartInventory.html** and **DynamicPartInventory.xsl** from **c:\labscode\exwsuddi\dynamic** into the webApplication\wsdynamic folder.
34. Study the **UddiServiceImplementer** code. This helper class accesses the UDDI Registry starting from a given provider and service name. It finds the tModels from the service name, finds the business entities from the provider name, follows to their services, and finally finds the implementers of the given service. The access points of the implementers are returned.

    Attention: You have to use the correct API URLs to connect to either the IBM Test Registry or the WebSphere UDDI Registry. Activate the correct code at the beginning of the getImplementers method.
35. Study the **DynamicServlet** code. The servlet is similar to the PartInventoryServlet. However, it uses the UddiServiceImplementer helper to get the access points, then each access point is passed into the PartInventoryProxy to invoke the Web Service. The resulting XML file of each access is converted into HTML by an XSL style sheet.
36. The HTML and XSL files are basically the same as for the fixed Web Service. The only real change is that the XSL only produces a part of the final HTML file. The start and end of the HTML are produced by the servlet.

Test the dynamic Web Services (optional)

37. If you are using the IBM Registry Preview, make sure it is started.
38. In the Java Perspective, select the **UddiTestList** program and run it. This is a stand-alone program that uses the UddiServiceImplementer to list the services. It should find your service.
39. With the **IBM Test Registry**:
    - Start the ItsoWsManufacturer server in the Server Perspective. Select the DynamicPartInventory.html file and *Run on Server*.
    - Enter a part number (M100000003) and click *Retrieve*. It should all work.
    - Stop the ItsoWsManufacturer server.

With the **WebSphere UDDI Registry** product, it is difficult to test the servlet because of port conflicts between WebSphere AEs and the internal ItsoWsManufacturer server. Therefore, we can only use WebSphere AEs:

- You must have completed Exercise 9, “Deploy and test a Web Service” and installed the ItsoWsManufacturerEAR in WebSphere AEs.
- Start WebSphere AEs (where the WebSphere UDDI Registry is running).
- Export an EAR file for the ItsoWsClientEAR project (itsowsclient.ear)
- Uninstall the ItsoWsClientEAR application in AEs:

```
seappinstall -uninstall ItsoWsClientEAR -delete true
```

- Install the new ItsoWsClientEAR application in AEs:

```
seappinstall -install itsowsclient.ear
      -expandDir d:\websphere\appserver\installedApps\itsowsclient.ear
      -ejbDeploy false -interactive false
```

- Start WebSphere AEs (where the WebSphere UDDI Registry is running).
- Start the servlet using:

```
http://localhost:8080/ItsoWsClientWeb/wsdynamic/DynamicPartInventory.html
```

- Enter a part number (M100000003) and click *Retrieve*. It should all work.
- Stop WebSphere AEs.

What you did in this lab

- Created a business entity in the UDDI registry
- Published a Web Service to the registry
- Searched the registry for Web Services
- Retrieved implementation and bindings files for a Web Service through the registry
- Optionally worked with an application using dynamic Web Services

Part 3

Appendixes

A

Installation and configuration

In this appendix we describe how to install the products to perform the exercises. See “System requirements for downloading the Web material” on page 388 for hardware prerequisites.

Windows NT or Windows 2000

Install Windows NT 4.0 workstation (or server) with Fixpack 6a, or as an alternative, Windows 2000 Professional with service pack 1 or 2.

Browser

You must have a recent Internet browser installed as well:

- Netscape 4.7
- Internet Explorer 5.0.1

DB2 Version 7.2 Enterprise Edition (or 7.1 Fixpack 3)

Installation procedure for DB2 Enterprise Edition:

- Select DB2 Enterprise Edition.
- Select DB2 Application Development Client.
- Do not select DB2 Administration Client (optional).
- Select custom install with:
  - Communication protocols
  - Administration/Configuration tools
  - Getting Started

  Other components are optional, not used in workshop.
- Directory: **c:\SQLLIB** (or d:\SQLLIB).
- Create the DB2 instance:
  - User ID/password: **db2admin**/**db2admin** (use same value)
- Install.
- Cancel the product registration, which starts afterwards (you can also remove the registration from the Startup folder).

Fixpack

- Stop all DB2 services.
- Install **FixPack 3 for DB2 Version 7.1**.
- Install latest **FixPack for DB2 Version 7.2**.

Create sample database

- Run *First Steps* after reboot and create the SAMPLE database (only DB2 UDB Sample, not other samples).

Change to JDBC 2.0

Access to DB2 using DataSource only works with JDBC 2.0.

- Stop all DB2 processes from the Services list.
- Change to JDBC 2.0 by running:

```
c:\SQLLIB\java12\usejdbc2.bat
```

- Restart DB2 processes.

WebSphere Application Server Advanced Version 4

Install the Application Server **Single Server** (**AEs**) **Version 4.01** to run the examples in a real environment.

Installation procedure:

- Run SETUP.EXE.
- Language English.
- Select *Typical* install; this also installs HTTP Server and JDK 1.3.
- User ID: use your Windows NT user ID (with admin authority).
- Directory: **c:\WebSphere\AppServer**, **c:\IBM HTTTP Server** (or d:\...).
- Install.
- Restart the system.
- Install Fixpacks (not required for exercises).

Verification

- Edit c:\IBM HTTP Server\conf**http.conf**. This file should contain these lines at the very end:

```
LoadModule ibm_app_server_http_module
                   D:/WebSphere/AppServer/bin/mod_ibm_app_server_http.dll
Alias /IBMWebAS/  "D:/WebSphere/AppServer/web/"
Alias /WSsamples  "D:/WebSphere/AppServer/WSsamples/"
WebSpherePluginConfig D:\WebSphere\AppServer\config\plugin-cfg.xml
```

  If these lines are missing, add them, and stop and start the HTTP server.

- Run First Steps (should be started automatically); otherwise, *Program -> IBM WebSphere -> Application Server V4.0 AEs -> First Steps*.
  - Start the application server (wait for the command window to complete).
  - Launch Administrative Console:
    - Login with your system user ID.
    - Expand Node - Enterprise applications (you should see the sample applications).
    - Config -> Save.
    - Exit.
    - Close browser.
  - Close First Steps.
  - Stop the application server in a command window: **stopserver**.

WebSphere Studio Application Developer

Install **Version 4.0.2** of the Application Developer. The instructions for the exercises have been updated for Version 4.0.2, but the samples also work on Version 4.0. (The content and sequence of some dialogs and SmartGuides has changed a little bit.)

Installation procedure:

- Run SETUP.EXE.
- Accept the license.
- Change directory to **c:\WSAD** (or d:\WSAD).
- Select Java Developer.
- Select CVS as team repository.
- Install.

Verification

- Start WSAD: *Programs -> IBM WebSphere Studio Application Developer -> IBM WebSphere Studio Application Developer*.
- You should see the Welcome panel.
- Stop WSAD using *File -> Exit*.

WebSphere UDDI Registry

Download the UDDI registry code from:

```
http://www7b.boulder.ibm.com/wsdd/downloads/UDDIregistry.html
```

You have to register to download the code.

The IBM WebSphere UDDI Registry can be used instead of the IBM Test Registry for Web Services publishing. This is especially useful for testing on a machine that is not connected to the Internet.

The IBM WebSphere UDDI Registry is not a complete product at the time of the writing of this redbook (February 2002):

- The beta code for Windows does not work with the Application Developer for publishing of business services.
- Later versions may provide the required functionality.
- You can perform the UDDI Explorer exercise using the IBM UDDI Test Registry.

Installing the WebSphere UDDI Registry

- Follow the instructions that come with the product.
- Install the product into WebSphere Application Server AEs.

ITSO workshop sample code

The sample code is available from the Redbooks Web site at:

```
ftp://www.redbooks.ibm.com/redbooks/SG246407/
```

Download the **sg246407code.zip** file and expand to the C drive. Select the option to use the folder names. This creates a directory structure under:

```
c:\ws\labscode\ex.....
```

Create DB2 database for exercise

The exercises are based on a DB2 database named **ITSOWSAD**.

- Open a DB2 command window.
- Go to the directory: `c:\ws\labscode\setup`
- Run the commands:

```
db2 -tf itsowsad.ddl            (define database and tables)
db2 -tf itsowsad.sql            (load sample data into tables)
```

- The database is created with four tables and sample data:

```
itso.aaparts, itso.aainventory          <=== dealer
itso.mmparts, itso.mminventory          <=== manufacturer
```

- The sample data is listed in Part 2, “Exercises”.

Cloning of machines

You can install one machine and then copy the whole drive to another machine, but you should then run disconnected, because all machines have the same machine name.

If you change the machine name to run connected:

- Change the Computer Name (Control Panel -> Network).
- Reboot.
- HTTP Server:
  - Edit c:\IBM HTTP Server\conf\httpd.conf:

    ```
    ServerName xxxxxx.yyyyy.com (xxxx=new computer name)
    ```
- WebSphere AEs:
  - Edit c:\WebSphere\AppServer\bin\SetupCmdline.bat

    ```
    SET COMPUTERNAME=xxxxxx
    ```
  - Edit the files:
    - c:\WebSphere\AppServer\config\server-cfg.xml
    - c:\WebSphere\AppServer\config\server-cfg.xml~
    - c:\WebSphere\AppServer\config\template-server-cfg.xml~
    - c:\WebSphere\AppServer\config\admin-server-cfg.xml

    and change the line:

    ```
    <nodes xmi:id="Node_1" name="xxxxxx">
    ```

Performing the exercises

After the products are installed, you can perform the exercises.

Sample code

The instructions refer to **c:\ws\labscode\xxxxx**, and that is the location where you should have installed the sample code.

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246407

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-6407.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg246407code.zip	Sample code for exercises
sg246407solution.zip	Solutions of exercises

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space	3 GB
Operating System	Windows NT or Windows 2000
Processor	550 MHz or better
Memory	385 MB, recommended 512 MB

How to use the Web material

Unzip the contents of the Web material **sg246407code.zip** file onto your hard drive. This creates a folder structure **c:\ws\labscode\exxxxx**, where *exxxxx* refers to an exercise:

exjava	Java development
exdata	Relational data center
exxml	XML development
exweb	Web development
exejb	EJB development
exdeploy	Deployment of Web and EJB applications
exwscreate	Web Service creation
exwsdeploy	Web Service deployment
exwsuse	Web Service usage
exwsuddi	Web Services and UDDI
setup	Setup of DB2 database and tables

The **sg246407solution.zip** file contains a similar folder structure with directories **c:\ws\labssolutions\exxxxx** that contain ZIP, WAR, or EAR files with the project content created in Application Developer.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 391.

- *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292
- *IBM WebSphere V4.0 Advanced Edition Handbook*, SG24-6176
- *WebSphere Version 4 Application Development Handbook*, SG24-6134
- *Programming J2EE APIs with WebSphere Advanced*, SG24-6124
- *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.1*, SG24-6284
- *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144
- *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server*, SG24-5754
- *Programming with VisualAge for Java Version 3.5*, SG24-5264

- *Version 3.5 Self Study Guide: VisualAge for Java and WebSphere Studio*, SG24-6136
- *How about Version 3.5? VisualAge for Java and WebSphere Studio Provide Great New Function*, SG24-6131
- *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, SG24-5755
- *Migrating WebLogic Applications to WebSphere Advanced Edition*, SG24-5956
- *Self-Service Applications Using IBM WebSphere V4.0 and IBM MQSeries Integrator*, SG24-6160
- *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, SG24-5864
- *The XML Files: Using XML for Business-to-Business and Business-to-Consumer Applications*, SG24-6104
- *The XML Files: Using XML and XSL with IBM WebSphere V3.0*, SG24-5479

Referenced Web sites

These Web sites are also relevant as further information sources:

- Apache SOAP:

  `http://www.apache.org/soap/`
- The Apache XML project:

  `http://www.apache.org/xerces-j/, http://www.apache.org/xalan-j/`
- IBM developerWorks, Web Services zone:

  `http://www.ibm.com/developerworks/webservices/`
- IBM UDDI Business and Test Registries:

  `http://www.ibm.com/services/uddi`
- UDDI Homepage:

  `http://www.uddi.org`
- World Wide Web Consortium (W3C), XML homepage:

  `http://www.w3c.org/XML`
- Web Services Toolkit Version 2.3 on IBM alphaWorks:

  `http://www.alphaworks.ibm.com/tech/webservicestoolkit`
- XMethods Web Service repository:

  `http://www.xmethods.com/`

How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

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:

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

Abbreviations and acronyms

AAT	application assembly tool
ACL	access control list
API	application programming interface
BLOB	binary large object
BMP	bean-managed persistence
CCF	Common Connector Framework
CICS	Customer Information Control System
CMP	container-managed persistence
CORBA	Component Object Request Broker Architecture
DBMS	database management system
DCOM	Distributed Component Object Model
DDL	data definition language
DLL	dynamic link library
DML	data manipulation language
DOM	document object model
DTD	document type description
EAB	Enterprise Access Builder
EAI	Enterprise Application Registration
EAR	enterprise archive
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
EJS	Enterprise Java Server
FTP	File Transfer Protocol
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	integrated development environment
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IMS	Information Management System
ITSO	International Technical Support Organization
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JAF	Java Activation Framework
JAR	Java archive
JDBC	Java Database Connectivity
JDK	Java Developer's Kit
JFC	Java Foundation Classes
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface
JSDK	Java Servlet Development Kit
JSP	JavaServer Page
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MFS	message format services
MVC	model-view-controller
OLT	object level trace
OMG	Object Management Group
OO	object oriented

OTS	object transaction service
RAD	rapid application development
RDBMS	relational database management system
RMI	Remote Method Invocation
SAX	Simple API for XML
SCCI	source control control interface
SCM	software configuration management
SCMS	source code management systems
SDK	Software Development Kit
SMR	Service Mapping Registry
SOAP	Simple Object Access Protocol (a.k.a. Service Oriented Architecture Protocol)
SPB	Stored Procedure Builder
SQL	structured query language
SRP	Service Registry Proxy
SSL	secure socket layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UCM	Unified Change Management
UDB	Universal Database
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
UOW	unit of work
URL	uniform resource locator
VCE	visual composition editor
VXML	voice extensible markup language
WAR	Web application archive
WAS	WebSphere Application Server
WML	Wireless Markup Language
WS	Web Service
WSBCC	WebSphere Business Components Composer
WSDL	Web Service Description Language
WSTK	Web Service Development Kit
WTE	WebSphere Test Environment
WWW	World Wide Web
XMI	XML metadata interchange
XML	eXtensible Markup Language
XSD	XML schema definition

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

Self-Study Guide: WebSphere Studio Application Developer and Web Services

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Self-Study Guide:
WebSphere Studio Application Developer and Web Services

Teach yourself WebSphere Studio Application Developer

Learn about Web Services

Create and use Web Services by example

This IBM Redbook is a self-study guide for the new application development tool WebSphere Studio Application Developer and for Web services.

WebSphere Studio Application Developer is the new IBM tool for Java development for client and server applications. It provides a Java integrated development environment (IDE) that is designed to provide rapid development for J2EE-based applications. It is well integrated with WebSphere Application Server Version 4 and provides a built-in single server that can be used for testing of J2EE applications.

Web Services are a new breed of Web applications. Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web Services perform callable functions that can be anything from a simple request to complicated business processes. Once a Web Service is deployed and registered, other applications can discover and invoke the deployed service. The foundation for Web Services are the simple object access protocol (SOAP), the Web Services description language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) registry.

This redbook consists of two parts: a presentation guide and an exercise guide. The presentation guide explains the new tool and Web services. The exercise guide provides detailed instructions to perform exercises using WebSphere Studio Application Developer. The sample code used for the exercises is available for download at the Redbooks Internet site. The sample code also includes the solutions that can be loaded and studied.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6407-00 ISBN 0738424196