WebSphere. software

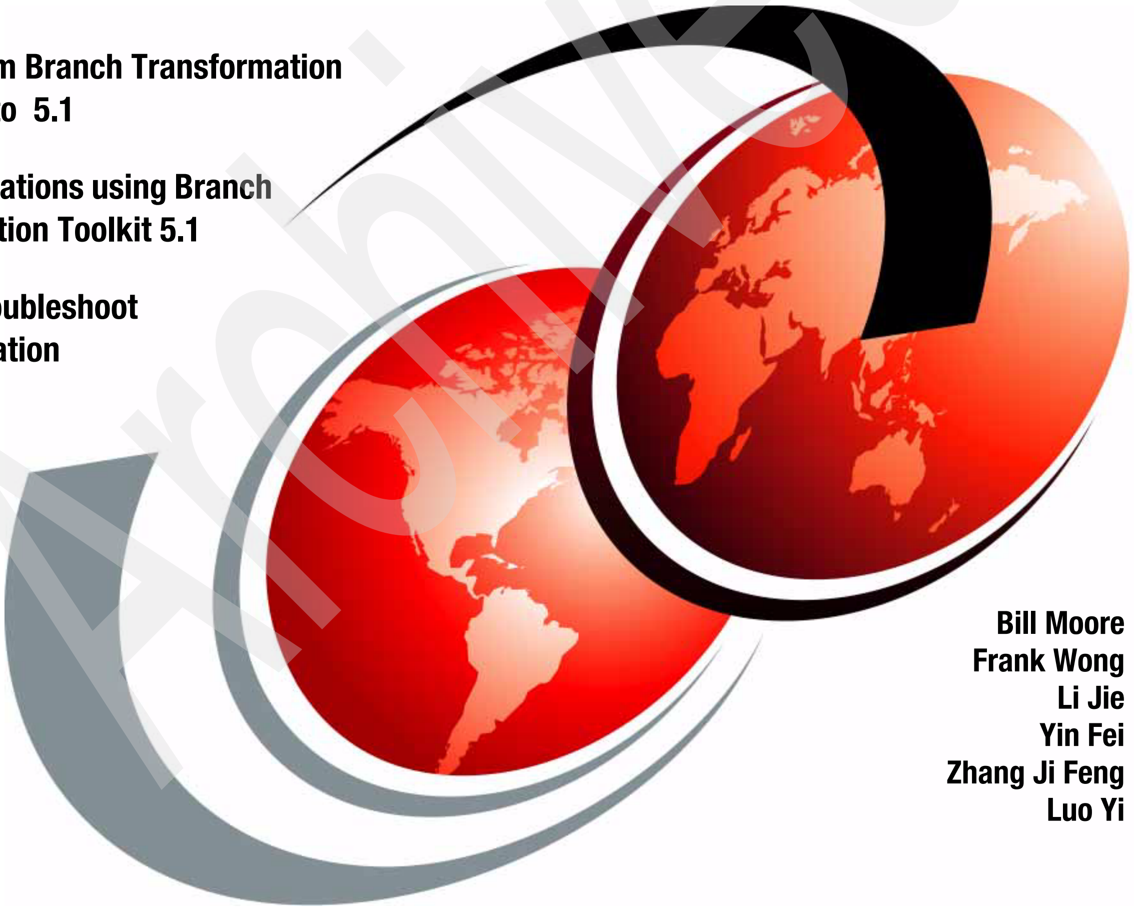
IBM

IBM Branch Transformation Toolkit 5.1
Migration and Usage Guidelines

Migrate from Branch Transformation Toolkit 4.3 to 5.1

Build applications using Branch Transformation Toolkit 5.1

Test and troubleshoot your application



Bill Moore
Frank Wong
Li Jie
Yin Fei
Zhang Ji Feng
Luo Yi

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Branch Transformation Toolkit 5.1 Migration and Usage Guidelines

June 2006

SG24-7160-00

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

First Edition (June 2006)

This edition applies to Versions 4.3 and 5.1 of IBM Branch Transformation Toolkit for WebSphere Studio.

Contents

Notices

Trademarks

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

AIX 5L™	Extreme Blue™	Redbooks (logo) ™
AIX®	ibm.com®	Redbooks™
CICS®	IBM®	RS/6000®
ClearCase®	IMS™	S/390®
DB2 Universal Database™	LANDP®	WebSphere®
DB2®	Lotus Notes®	z/OS®
@server®	Lotus®	zSeries®
@server®	Notes®	
eServer™	Rational®	

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, J/XFS, Java, JavaBeans, JavaScript, JavaServer, JavaServer Pages, JDBC, JDK, JRE, JSP, JVM, J2EE, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redbook shows in detail the IBM Branch Transformation Toolkit for WebSphere Studio Version 5.1 and explains how to migrate from Branch Transformation Toolkit Version 4.3 to Branch Transformation Toolkit 5.1. We provide guidelines covering the architecture of the target solution, the correct use of migration tools, and migration-sizing techniques. This redbook also describes how to build new applications using Branch Transformation Toolkit V5.1. We explain both top-down and bottom-up development models, and discuss how to use WebSphere Studio development plug-ins and Branch Transformation Toolkit development plug-ins. We also describe the rich Java™ client development using the Branch Transformation Toolkit.

This book is intended for the following audiences:

- Solution architects who require an overall description of what the IBM Branch Transformation Toolkit for WebSphere Studio provides and how you can use it to build a solution.
- IT professionals and executives who require a broad understanding of the architecture of the Branch Transformation Toolkit and its implementation.
- Readers who are familiar with object-oriented software and related development techniques, and have a general knowledge of Java 2 Platform, Enterprise Edition (J2EE™) and related technologies, including network computing and Internet technologies.

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 (ITSO), Raleigh Center.

Bill Moore is a technical staff member at the IBM Extreme Blue™ lab in Raleigh, North Carolina where he provides project coordination and leadership for Extreme Blue projects.

From 2000 to 2006 Bill was an IBM WebSphere® specialist at the ITSO, Raleigh Center producing IBM Redbooks™ and other associated documentation for Application Integration & Middleware and Rational® products. He wrote extensively and taught classes on WebSphere and related topics. Before joining the ITSO, BillI was a Senior AIM Consultant at the IBM Transarc lab in Sydney, Australia. He has 21 years of application development experience on a wide range of computing platforms using many different coding languages. His current areas of expertise include J2EE, WebSphere Application Server, application development tools, object-oriented programming and design, and e-business application development.

Frank Wong is a Software Engineer at the IBM China Development Lab in Taipei, Taiwan. He worked on the project that implemented the technical support site of ibm.com® for two years. His areas of expertise include J2EE, Web services, design patterns and software development methodologies.

Li Jie is a Staff Software Engineer at the IBM China Development Lab in Beijing, People's Republic of China. His area of expertise include C/C++, Java development, software development methodology and human-computer interfaces.

Zhang Ji Feng is a Technical Manager at the Beijing Nantian Software Co., Ltd., People's Republic of China. He has five years of experience in the banking industry and has worked at Natian for five years. His areas of expertise include C/C++, J2EE, bank branch teller systems, bank core systems, design patterns and software development methodologies.

Luo Yi is a Software Engineer at the Beijing Nantian Software Co., Ltd. People's Republic of China. He has over four years of experience in Java development and J2EE technology. His areas of expertise include Java programming, WebSphere Application Server development, distributed systems, testing, and troubleshooting. He also has experience with C# development.

Yin Fei is an Advisory Software Engineer at the China Software Development Lab in Beijing, People's Republic of China. He has worked at IBM for ten years. He has experience in branch transformation solution for retail banking, include branch teller, self-service ATM and Kiosk systems, and Multi-Channel integration. Currently he is focusing on Branch Transformation Toolkit developement and services.

The authors: Yin Fei, Frank Wong, Li Jie, Luo Yi, Zhang Ji Feng

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

Yu Ling Tong
IBM People's Republic of China

Martin Leclerc
IBM Canada

Become a published author

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

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

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

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

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

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

  `ibm.com/redbooks`

- Send your comments in an email to:

  `redbook@us.ibm.com`

- Mail your comments to:

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

Part 1

Migration

In this part we describe migration from IBM Branch Transformation Toolkit for WebSphere Studio V4.3 to IBM Branch Transformation Toolkit for WebSphere Studio V5.1.

1

Introduction to IBM Branch Transformation Toolkit

This chapter describes the objectives, focus, and the intended audience of this redbook. Besides outlining the architecture and topologies of the IBM Branch Transformation Toolkit for WebSphere Studio V5.1, this chapter also provides a mapping from Branch Transformation Toolkit 4.3 to Branch Transformation Toolkit 5.1 terms and concepts.

This chapter describes the following topics:

1.1 Objectives

Financial institutions are always trying to offer and adapt their products and services to ensure that they are able to respond to future market challenges and support changing business operations in an increasingly competitive environment. The IBM Branch Transformation Toolkit for WebSphere Studio provides a set of facilities to help with each of the processes and presents them to development teams in a familiar manner. The Branch Transformation Toolkit is a pragmatic infrastructure that is designed and built in such a way that existing mission-critical systems can evolve, rather than be replaced. The toolkit architecture provides an environment for high development productivity and great flexibility to meet the challenges of the new pace of change in the technology industry and the banking industry.

IBM Branch Transformation Toolkit for WebSphere Studio 5.1 is an evolutionary offering that helps users of earlier versions move up to the full power and versatility of the IBM WebSphere Application Server and IBM WebSphere Business Integration Server Foundation. The new Branch Transformation Toolkit is based on the J2EE standard and complies with J2EE specification version 1.3. The version 5.1 toolkit provides a migration path to allow clients to reuse much of their existing toolkit application code base while taking advantage of the WebSphere Application Server J2EE environment. With version 5.1, you can bypass toolkit functionality entirely and access J2EE components directly, if desired.

Usually, users perceive migration as a form of change. However, migration is a way of life. Branch Transformation Toolkit continues to evolve in order to support ever-increasing functionalities. To use a new functionality, applications and processes have to evolve as well. Any action that is taken to reduce the cost of change helps save costs in the long run. Investing the time to do it immediately saves time and money in the future.

The object of this redbook is to introduce the tools and the manner in which they are to be used while migrating from Branch Transformation Toolkit V4.3 to Branch Transformation Toolkit V5.1. Details about our collective experience and the lessons we learned from carrying out a practical migration, are also provided. The contents of this book cover the entire migration process, that is, from strategy and analysis to architecture, operation, and deployment environment migration. This book also describes the best practices to be followed in order to create applications efficiently in the new environment provided by the Branch Transformation Toolkit.

1.2 How this book is organized

This migration and usage guide is intended for use by solution architects and developers to assist in their migration effort. The focus of this redbook is migration from Branch Transformation Toolkit version 4.3 to version 5.1, including migration tools and new development tools.

The migration path covers migration preparation, migrating operation, manual modification, testing and deployment, as shown in Figure 1-1.

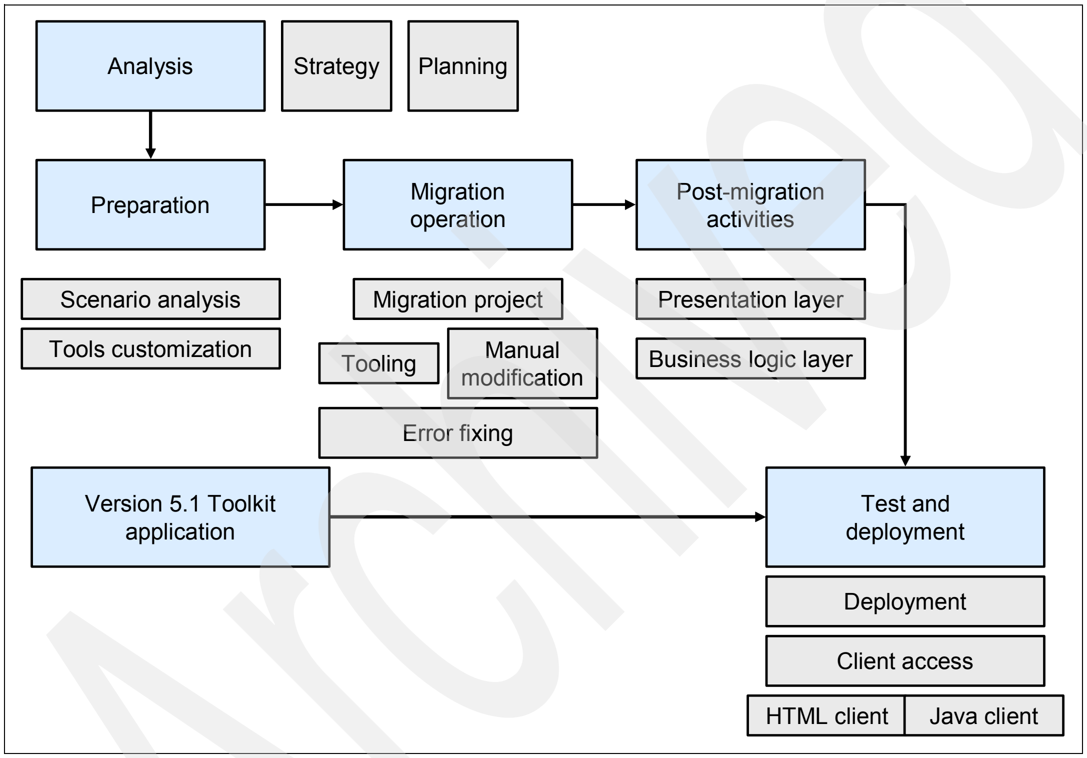


Figure 1-1 The migration path

This redbook contains the following chapters:

- Chapter 1, "Introduction to IBM Branch Transformation Toolkit" on page 3

  This chapter describes the objectives and organization of the book. The Branch Transformation Toolkit V5.1 architecture is outlined and a mapping of concept from Branch Transformation Toolkit version 4.3 to version 5.1 is provided.

This chapter includes the following topics:

– 1.1, “Objectives” on page 4
– 1.2, “How this book is organized” on page 5
– 1.3, “Branch Transformation Toolkit 5.1 architecture” on page 8
– 1.4, “Terms and definitions” on page 11
– 1.5, “Concept mappings” on page 15

- Chapter 2, “Migration strategy” on page 21

  This chapter covers the best practices in migration strategies and discusses the stages of a migration.

  This chapter includes the following topics:

  – 2.1, “Introduction to migration” on page 22
  – 2.2, “Hardware and software requirements” on page 23
  – 2.3, “Migration stages” on page 27
  – 2.4, “Things to do before migration” on page 29
  – 2.5, “Elements of migration” on page 35

- Chapter 3, “Planning a Branch Transformation Toolkit migration” on page 57

  This chapter discusses the methodology for using Branch Transformation Toolkit and details about the migration activities. The programming model and topology and customer extensions and limitations are also detailed.

  This chapter includes the following topics:

  – 3.1, “Version 5.1 End-to-end programming model” on page 58
  – 3.2, “Application packaging and topology” on page 75
  – 3.3, “Workload management decisions” on page 78
  – 3.5, “Migration considerations for custom extensions” on page 82
  – 3.6, “Limitations” on page 85

- Chapter 4, “Preparing for migration” on page 89

  This chapter outlines the work that must be completed before dealing with a migration. This comprises consists of several phases, including tool customization.

  This chapter includes the following topics:

  – 4.1, “Analysis and preparation” on page 90
  – 4.2, “Setting up the migration tools” on page 91
  – 4.3, “Customizing the migration tools” on page 96

- Chapter 5, “Migrating an application” on page 97

  This chapter talks about how to migrate an application by using the migration tools provided by the Branch Transformation Toolkit. We migrated a sample application using this method. A migration project was created before carrying out an automatic migration. The manual work needed to complete the migration is also described.

This chapter includes the following topics:

- 5.1, “Creating a new migration project” on page 98
- 5.2, “Using the migration tools” on page 102
- 5.3, “Manual modification for migration” on page 115

- Chapter 6, “Post-migration activities” on page 119

  This chapter discusses the post migration activities that are necessary to complete the migration. This includes the application architecture, operation, flow, business process, event and communication service, as well as the development and deployment environment.

  This chapter includes the following topics:

  - 6.1, “Sample project requirements” on page 120
  - 6.2, “Importing the required EARs for version 5.1” on page 124
  - 6.3, “Preparing the client and the server” on page 132
  - 6.4, “Adding and modifying code” on page 153

- Chapter 7, “Testing and deployment” on page 209

  This chapter provides details about how to test and deploy the migration application. We used two types of clients to call the services on the server side to complete the testing.

  This chapter includes the following topics:

  - 7.1, “Configuring the test environment” on page 210
  - 7.2, “Java client” on page 211
  - 7.3, “HTML client” on page 213
  - 7.4, “Application deployment” on page 214

- Chapter 8, “Building an application with Branch Transformation Toolkit V5.1” on page 225

  This chapter provides information about the sample application we constructed using Branch Transformation Toolkit V5.1.

  This chapter includes the following topics:

  - 8.1, “Before getting started” on page 226
  - 8.2, “Leveraging the WebSphere Studio features” on page 235
  - 8.3, “Developing an application using Branch Transformation Toolkit” on page 248
  - 8.4, “Developing a rich Java client” on page 431

1.3 Branch Transformation Toolkit 5.1 architecture

This section introduces the prime architecture of Branch Transformation Toolkit 5.1. The new version is based on the J2EE standard. The development environment for toolkit applications is integrated with WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition, which are based on the Eclipse Platform. The runtime environment for toolkit applications is based on WebSphere Application Server or WebSphere Business Integration Server Foundation.

The architecture of a Branch Transformation Toolkit 5.1 application solution is based on a logical three-tier model:

- Back-end enterprise tier
- Application server tier
- Client tier

Figure 1-2 on page 9 depicts these tiers.

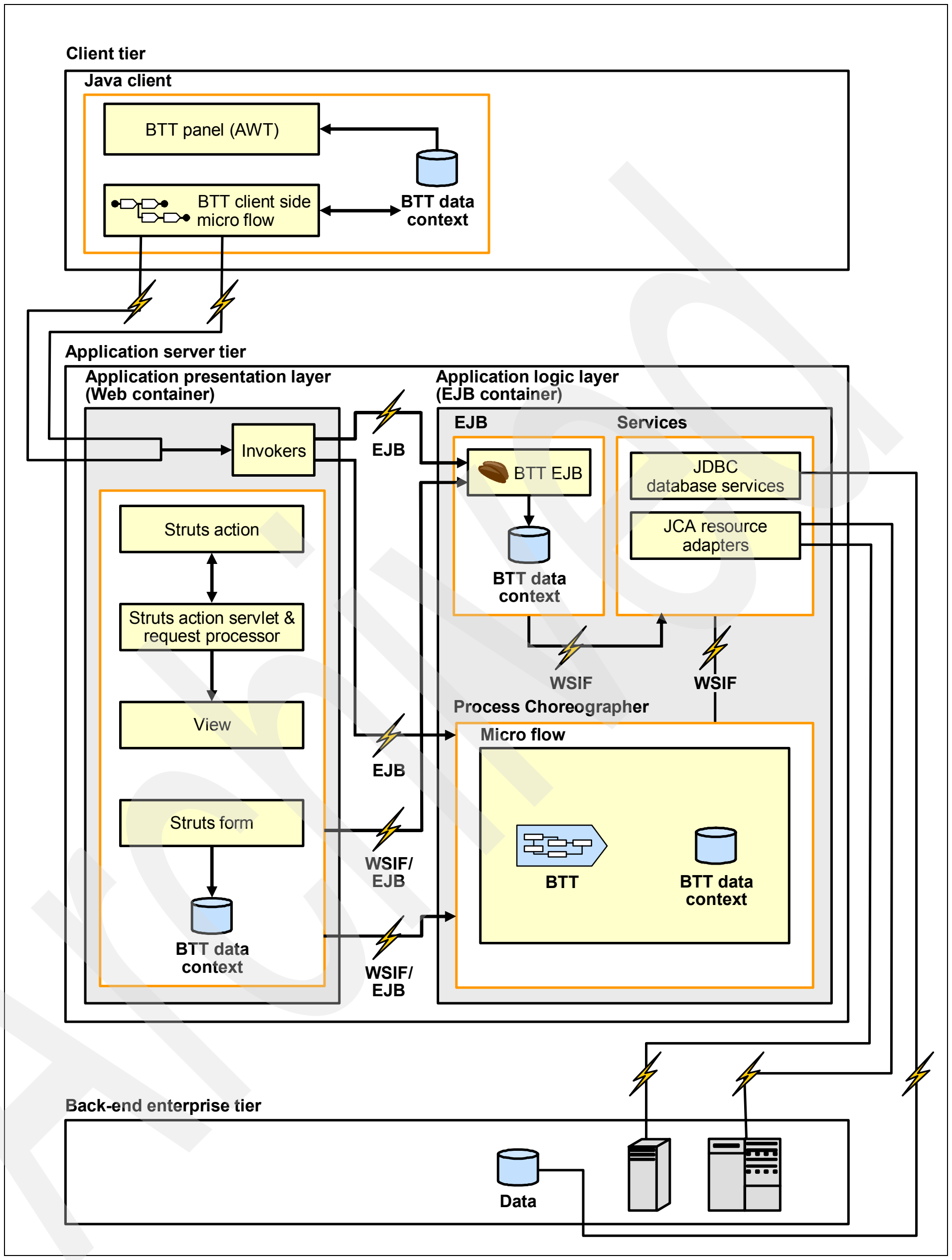


Figure 1-2 The logical tiers in Branch Transformation Toolkit architecture

The client tier is responsible for presenting an interface to the user in a client device. The Branch Transformation Toolkit 5.1 supports the following types of clients:

- Java clients running as applets in a browser or as Java applications

  For Java clients, the toolkit provides a set of graphical user interface (GUI) JavaBeans™ to build the client user interface and a mechanism to navigate the application views.

- HTML browser clients

  For HTML browser clients, the toolkit's Struts Extension framework on the application presentation layer of the application server tier manages the user interface and view navigation.

The application server tier has two parts:

- The application presentation layer

  The application presentation layer is responsible for creating a request for the invocation of business logic that is hosted within the application logic layer. In HTML browser clients, the presentation layer is also responsible for providing the view navigation. The application presentation layer converts an action carried out by a user in the user interface into a Web Services Invocation Framework (WSIF) message or Enterprise JavaBean (EJB™) method invocation, which the presentation layer then sends to the application logic layer.

- Application logic layer

  The application logic layer is responsible for performing the business process that fulfils the presentation layer request. A business process consists of a set of activities and may involve accessing and manipulating enterprise data and performing financial service procedures in the back-end enterprise tier.

  The application logic layer can use two mechanisms for performing the business process. If the application server tier is running on WebSphere Business Integration Server Foundation, the application logic layer can use the Business Process Choreographer and work area features. If the application server tier is running on WebSphere Application Server, the application logic layer can use a Single Action EJB. For more details about Business Process Choreographer and Single Action EJB, refer to 1.4, "Terms and definitions" on page 11.

The back-end enterprise tier consists of enterprise level databases and older systems that provide existing business logic and services. The application logic layer communicates with these databases and systems through J2EE Connector Architecture (JCA) connectors, database services, and formatters. The connectors, database services, and formatters form an interface so that toolkit applications do not impose any changes directly on these databases and systems.

As a result of being written in Java, the design and portability of the product allows the application tier servers to exist at either the branch level, that is, one server per branch, the regional level, that is, one server per a group of branches, or at a centralized level, that is, a single server for the entire financial institution. These options provide the flexibility to achieve the right balance between the number of servers and network bandwidth, with no changes to application logic.

Branch Transformation Toolkit 5.1 supports multiple channels for application delivery. For example, a typical configuration for bank tellers is a grouping of client workstations with physically attached financial devices, with each one having an HyperText Markup Language (HTML) browser or a Java client desktop. The client has access to the application presentation layer on server side through HTTP or Hypertext Transfer Protocol Secure (HTTPS) protocol. The server side provides services for the client workstation and access to the business processors provided by the application logic layer.

Internet banking users access financial services through a Web browser running on a device connected to the Internet. The HTTP protocol connects the client side to the server side, which is in the application presentation layer. The application is usually located at a central site behind a firewall. In most cases, the client views are HTML pages. However, an HTTP protocol can use other technologies and presentation options such as eXtensible Markup Language (XML) messages if the client device supports them.

Toolkit applications can also appear in kiosks or ATMs that run Internet technologies such as a Web browser and Java. The client usually supports the financial devices present in the terminal such as magnetic stripe reader (MSR), chip card reader, receipt printer, passbook printer, bar code reader, or touch-screen display.

1.4 Terms and definitions

This section introduces the terms and definitions used in this book, as a reference. Some of them are newly defined in Branch Transformation Toolkit 5.1 because this is the first time that they have been used in toolkit architecture, especially on the server side. The terms and concepts can be grouped according to their architectural location, that is, either client side or server side. Figure 1-3 shows some terms and definitions within the boundary of toolkit architecture.

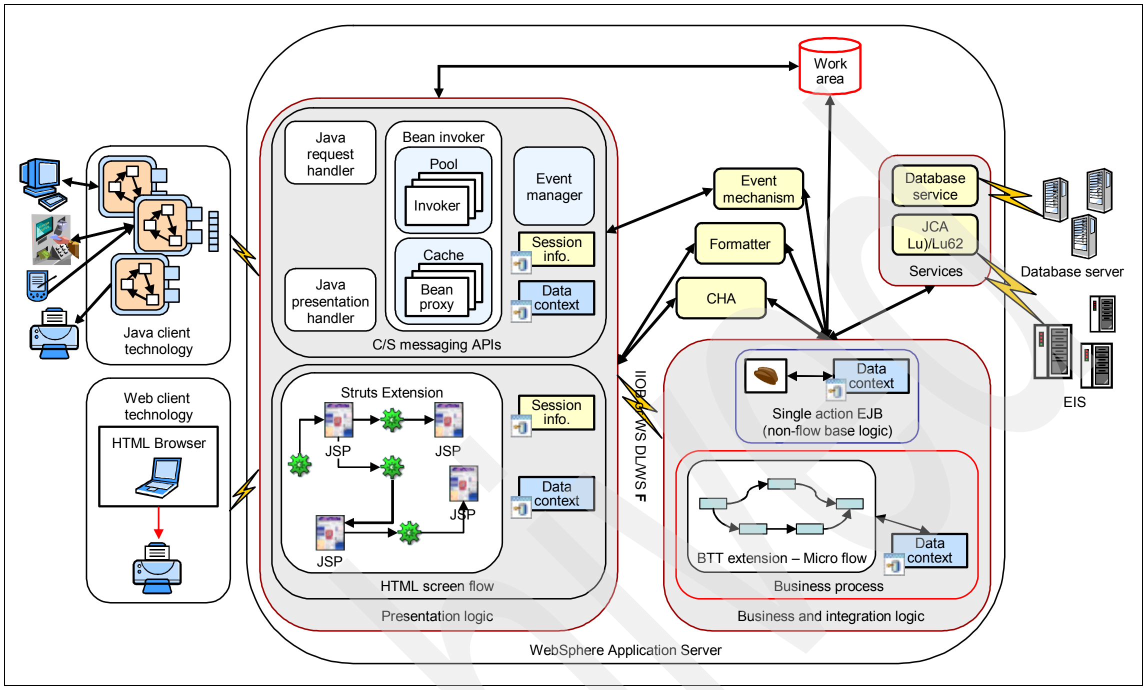


Figure 1-3 Toolkit terms and concepts

1.4.1 Client-side terms and concepts

The following terms and concepts apply to the toolkit client-side architecture:

- Contexts

  A *context* is the container for the data elements needed by a business entity such as a user or branch. Contexts have a hierarchy to enable these business entities to share common data. For example, in a branch, each teller will have a context that contains data about the teller, but they will all share the branch context that will contain data about the branch. In this relationship, the branch context is the parent and each teller context a child.

- Formatters

  A *formatter* transforms a string into data in a context or data in a context to a string. This enables an application to move data into and out of the context hierarchy and to create messages to send to a host, financial device, or service in a format understood by the message's destination. The toolkit provides an extensive set of the most commonly needed formatters for financial service applications, including Extended Binary Coded Decimal Interchange Code (EBCDIC), date, numeric, packed, binary, and other formatters.

- Events

  An *event* is how components within the application presentation layer communicate with each other. A notifier is the sender of an event. A handler, as the receiver of that event, is responsible for consuming the event or propagating it to other handlers. An event manager acts as the event controller between notifiers and handlers in order to manage both local and remote events.

- Client operation

  A *client* operation is what Java clients use to launch business processes in the application logic layer. An invoker maps the client operation to the business process.

1.4.2 Server side terms and concepts

The Branch Transformation Toolkit components that run on application servers can be divided into two categories: components running in the Web container of the application presentation layer and components running in the EJB container of the application logic layer. The Web container and the EJB container share some toolkit common components. The corresponding terms are divided as follows:

Shared components across containers

These terms and concepts describe the components shared by the Web and EJB containers:

- Common Hierarchical Area (CHA)

  The *Common Hierarchical Area* holds data within a context hierarchy for the business process component when it performs a business process. This is a distributed component that allows the data to exist anywhere. It also enables nontoolkit applications to store general global session data. The application uses the CHA application programming interface (API) to move data into and out of the CHA.

- CHA formatter service

  The *CHA formatter service* handles formatting and unformatting of strings and data items for applications and services. It converts a specific data item into a string representation of the data item and parses a string into a specific data item.

Web container components

These terms and concepts describe the components used by the Web container:

- Invoker

  An **invoker** is the interface to an EJB in the application logic layer. A request handler, which is a part of the multichannel architecture, or the toolkit Struts extension uses a specific invoker to start the business process performed by the EJB associated with the invoker.

- Client/Server Messaging API

  The *Java Client/Server* connectivity component enables the Java client and application presentation layer to communicate through a specific communication channel. The component contains a request handler, a Bean Invoker Factory, and a presentation handler. The request handler passes the request to the Bean Invoker Factory, which then instantiates an invoker to call a Single Action EJB or a business process in the application logic layer. The presentation handler handles the response from the application logic layer to render the result appropriately for the Java client.

- Struts extensions

  The *Struts extensions* component provides a set of features and mechanisms that support an HTML-based GUI that is presented in a Web browser, using an HTTP connection. The toolkit Struts Extensions component is based on the Apache Struts Web application Framework.

EJB container components

These terms and concepts describe the components used by the EJB container:

- Business process

  This component enables applications to perform business processes using the Business Process Choreographer in WebSphere Business Integration Server Foundation. Applications can invoke the business process using the request handler feature of the multichannel architecture and an EJB interface or by using a flow processor through the EJB or Web Services Invocation Framework (WSIF) interface. The request handler and the flow processor both reside in the application presentation layer.

- Single action EJB

  This component enables applications to perform business processes using the stateful session EJBs. The Invoker component in the presentation layer is the interface to the single action EJBs.

- Startup beans

  A *startup bean* is a session EJB that loads and runs before an application starts. The Branch Transformation Toolkit uses startup beans to perform initialization for some application logic layer components such as the CHA, CHA formatter service, and services.

1.5 Concept mappings

Branch Transformation Toolkit 5.1 has many additions and enhancements when compared with Branch Transformation Toolkit 4.3. In this section, we map the concepts from Branch Transformation Toolkit version 4.3 to version 5.1.

1.5.1 Toolkit application architecture

A new feature of Branch Transformation Toolkit 5.1 is that it is based on the J2EE standard architecture. On the client side, the Java client application remains the same as in version 4.3. The existing code can be used in the new toolkit environment without modification. However, the Branch Transformation Toolkit application server tier has been split into two layers to separate the presentation from the business logic. The application presentation layer residing in the Web container of the J2EE environment is now restricted to creating requests for business logic hosted in the application-logic layer. For non-Java clients, the application presentation layer provides view navigation based on the Apache Struts framework. The application-logic layer focuses on performing the business logic requests.

This change enables applications to use many more capabilities of WebSphere Application Server or WebSphere Business Integration Server Foundation while providing backward compatibility with version 4.3. Version 5.1 of the Branch Transformation Toolkit provides of a set of entirely new components that perform and support a business process, and helps access services and data from the back-end enterprise tier.

1.5.2 Application server components

This section describes the difference between the old version and the new version of the application server components resulting from the new architecture. Many old toolkit components that ran in the application server have been modified or changed significantly. These components include the following:

Flow processors

The flow processors do not exist in application servers any longer. They have been replaced with business processes. Using the client/server interfaces, toolkit clients now pass requests to an invoker or to a WebSphere Server Integration Foundation action. These requests call a business process component or single action EJBs in the application logic layer.

Server operations

Server operations do not exist in application servers now. Stepped operations have been replaced with business processes, and non-stepped operations with business processes or single action EJBs. Through the client/server interfaces, toolkit clients pass requests to an invoker or a WebSphere Server Integration Foundation action that calls the business process component or single action EJBs in the application logic layer.

Server side components

This section describes the changes made to the server components, based on the container they are used by.

Shared components across containers

These components can be shared across the Web container and the EJB container, that is, they can be used both by the application presentation layer and the application logic layer.

- Common Hierarchical Area (CHA)

  This component uses J2EE technology to wrap contexts as EJBs so that they can be accessed by any entity running in the EJB container. This enables a business process to get the data it needs. Since CHA contexts are EJBs, non-toolkit applications can use CHA to contain general global session information. This is a new concept that has been introduced in version 5.1.

- CHA contexts

  These contexts, which run in the application server, are different from those running in the Java client side. Compared to contexts running on the Java client side, the CHA contexts have additional wrapping so that they can exist as EJBs. There is no connection between the contexts and CHA contexts. This too is a new concept introduced in version 5.1.

- Events

  The events running in the application server side have a different structure from those running in the Java client side. The application server side events use Java Message Service (JMS) for events propagation.

- Externalizers

  Externalizers exist in both the layers, except that the application logic layer does not contain externalizers for toolkit entities that are related purely to presentation, such as flow processors and views.

Application presentation layer components (Web container)

This section describes the components in the new application presentation layer. These components operate within the Web container of WebSphere Application Server. They call application server components through WSIF messages or through EJB method invocation to carry out transactions.

- Sessions

  The session management component now exists only in the application presentation layer. All session handling is managed through a session handler, and CHA contexts do not provide any session management features.

- Bean invoker factory

  This component creates the invoker that invokes the business logic implementation in the application logic layer. The implementation can be a single action EJB or a business process running in the Business Process Container of WebSphere Business Integration Server Foundation.

- Struts extensions

  This component extends the Apache Struts framework to support an HTML-based GUI that is presented in a Web browser, using an HTTP connection. The Struts extensions component replaces the HTML request handler and flow processors. It receives requests from HTML clients and handles the presentation for those clients. This is another new concept introduced in version 5.1.

- JavaServer Pages(JSP™)

  The JSPs used in Branch Transformation Toolkit 5.1 solutions now make intensive use of the Struts tag library. Branch Transformation Toolkit 5.1 further extends the tag library to provide more tags for toolkit applications.

- Java client/server messaging APIs

  The Java client/server messaging APIs contain the Java client request handler and the Java client presentation handler. This component does the same things for Java clients as the Struts extensions component does for HTML clients.

Application logic layer components (EJB container)

This section describes the components in the new application logic layer. These components operate within the EJB container of WebSphere Application Server.

- Business process

  This component prepares the data for a business process running in the Business Process Container of WebSphere Business Integration Server Foundation and creates the response, once the process flow has finished. WebSphere Studio Application Developer Integration Edition provides the process editor as a visual tool to facilitate the creation of business processes. To reduce migration effort, the business process component provides the following:

   - com.ibm.btt.server.flow.BTTOperActivity
   - com.ibm.btt.server.flow.BTTOperStepActivity

  These classes replace *DSEServerOperation* and *OperationStep* respectively.

  This too is a new concept introduced in version 5.1.

- Single action EJBs

  These are EJBs that perform business processes. They are functionally equivalent to business processes running in the Business Process Container, except that they can run on any edition of WebSphere Application Server and perform better. This is also a new concept introduced in version 5.1.

- Communication services

  These services support communication with back-end enterprise systems, through JCA. The toolkit provides the SNA JCA Lu0 connector and the SNA JCA Lu62 connector as resource adapters that conform to the JCA standard. The SNA JCA Lu62 connector is a new feature of the toolkit.

- Database services

  Database services exist in the application logic layer and follow its architecture. The toolkit provides electronic journal and database table mapping services.

  In the electronic journal, the JDBCJournal is the service requester and maintains the same API as it did in the previous version.

  Note: Because WebSphere Application Server now handles the connection to the database, JDBCJournal no longer needs to get, set, or load a JDBC™ driver. The JDBCJournal communicates with the service object JDBCJournalImpl, using local Java calls, EJB method calls, WebSphere Server Integration Foundation SOAP binding messages, or WebSphere Server Integration Foundation EJB binding messages. The service object accesses the database to perform the request sent to it by the requester JDBCJournal.

  Because the schema generator JDBCJournalSchemaGenerator and the JDBC driver must be in the same Java virtual machine (JVM™), the electronic journal no longer allows an application to call the generator to create the tables. Instead, the database administrator uses the generator to create the tables so that they are available when the application starts.

In the database table mapping service, the JDBCTable is the service requester and maintains the same API as it did in the previous version.

Note: Since WebSphere Application Server now handles the connection to the database, JDBCTable no longer needs to get, set, or load a JDBC driver. The JDBCTable communicates with the service object JDBCTableImpl using a WebSphere Server Integration Foundation interface. The service object accesses the database to perform the request sent to it by the requester JDBCTable.

With the new implementation, the externalizer for the service ignores the following attributes in the service definition:

– databaseURL
– JDBCDriver
– poolName
– sharedConnection
– statementPoolSize

The JDBCServicesConnectionManager has the following new attributes:

– orphanTimeout

  The number of seconds that pass before the JDBCServicesConnectionManager discards an unused or idle connection.

– reapTime

  The number of seconds that pass between runs of the pool maintenance thread.

1.5.3 New tools in Branch Transformation Toolkit V5.1

New tools have been added and existing tools updated in the Branch Transformation Toolkit V5.1. The changes include the following:

- Graphical builder

  The graphical builder is the tool used during toolkit development for creating and maintaining external definitions. It is a plug-in for WebSphere Studio Application Developer and WebSphere Studio Application Developer Integration Edition. The graphical builder provides a development environment used throughout the development cycle of toolkit applications. It also acts as a portal from where you can start other tools the toolkit provides.

- CHA editor

  The CHA editor is the tool used during toolkit development for creating and maintaining the external definition files of CHA contexts and their data elements and types. It provides a visual representation of the structure and relieves you of the need to deal with XML tags. The CHA editor is a WebSphere Studio Application Developer plug-in.

- Format editor

  The format editor is the tool used during toolkit development for creating and maintaining the external definition files of formatters and their CHA contexts and data elements. It provides a visual representation of the structure and relieves you of dealing with XML tags. The format editor is a WebSphere Studio Application Developer plug-in.

- Business process wizard

  The business process wizard is the tool used to customize your business processes to take advantage of toolkit-specific entities such as CHA contexts. It provides a GUI that relieves you from editing the Business Process Execution Language(BPEL) files directly.

- Struts tools extensions

  The Struts tools extensions provides a GUI to help you extend your Struts configuration files to take advantage of toolkit-specific entities such as CHA contexts. It provides a GUI that relieves you from editing the Struts configuration files in XML format directly. The Struts tools extensions is an integral part of toolkit development and is documented separately.

- Migration tools

  The migration tools provides wizards to help you migrate the toolkit configuration files, context definitions, formatter definitions, server operations, flow processors, and screen flows of your version 4.3 application to the corresponding components of your version 5.1 application. The migration tools can also generate graphical builder files that enable you to further modify your migrated application with the graphical builder.

2

Migration strategy

This chapter discusses the best practices in migration strategies and planning. Three migration stages are described and their advantages and disadvantages discussed here.

The following topics are discussed in this chapter:

2.1 Introduction to migration

Change is an inevitable part of enterprise application ownership. As new technologies evolve, applications are adapted to make use of them. The related technologies used by IBM Branch Transformation Toolkit for WebSphere Studio, change on a regular basis. For example, a new and major version of the WebSphere Application Server is often released every 18-24 months. Correspondingly, a new version of the Branch Transformation Toolkit is often released to take advantage of the changes in the WebSphere Application Server. This means that you should consider migrating to new product versions and plan for migration activities.

Migration strategy is an important prerequisite to consider before starting any migration work. There are many aspects and details that should be considered in the migration strategy. Migration is not simply a code upgrade or replacement of a working platform. The mechanical aspects of migration are relatively easy to complete. However, a complete migration plan should include the necessary upgrades to the solution architecture as well as to the application code. You should also consider the effect migration will have on your business processes and on the development and deployment environments for your solution.

The migration should not impact normal business practices negatively. Normal business must continue. When planning for migration to Branch Transformation Toolkit 5.1, be aware that it will have an impact on several aspects of your current business. The migration of toolkit applications involves far more than just the migration of application code. You should plan for migration impacts on your development environments, build processes, and runtime environments. The key areas to think about include:

- Application code
- Development environment
- Testing
- Runtime environments
- Deployment processes

At the same time, we recognize that our clients cannot always upgrade their application software all at once. You can decide how much or how little toolkit functionality to retain. Version 5.1 allows you to upgrade to a full J2EE environment incrementally. Alternatively, with the help of Branch Transformation Toolkit 5.1, you can bypass toolkit functionality entirely, and access the J2EE components directly.

This chapter describes the migration strategies and considerations at every stage when migrating to Branch Transformation Toolkit 5.1. It also outlines the key elements and details to be considered during the migration process.

2.2 Hardware and software requirements

During the migration process, prepare the runtime and development environments to meet the migration requirements. This section lists the hardware and software requirements for the runtime and development environments of Branch Transformation Toolkit 5.1.

The IBM Branch Transformation Toolkit for WebSphere Studio 5.1 supports the following operating systems:

- Microsoft® Windows® 2000
  - Microsoft Windows 2000 Professional with Service Pack 4
  - Microsoft Windows 2000 Server with Service Pack 4
  - Microsoft Windows 2000 Advanced Server with Service Pack 4
- Microsoft Windows XP
  - Microsoft Windows XP Professional with Service Pack 1
- Microsoft Windows Server® 2003
  - Microsoft Windows Server 2003, Standard
  - Microsoft Windows Server 2003, Enterprise
- Sun™ Solaris™
  - Sun Solaris 8 with the Recommended Patch Cluster of November 2003
  - Sun Solaris 9 with the Recommended Patch Cluster of November 2003
- Red Hat Linux® on Intel®
  - RedHat Linux v7.2
  - RedHat Linux v8.0
  - RedHat Linux v9.0
  - Red Hat Enterprise Linux WS 3.0 Update 1 or 3
  - Red Hat Enterprise Linux AS 3.0 Update 1 or 3
  - Red Hat Enterprise Linux ES 3.0 Update 1 or 3
- IBM AIX®
  - AIX Version 5.1 with the 5100-05 maintenance package
  - AIX Version 5.2 with the 5200-01 recommended maintenance package and APAR iY44183, and PTF U484272
  - AIX Version 5.2 with 5200-03 recommended maintenance package
  - AIX 5L^{M} version 5.3 with WebSphere Application Server APAR PK01428
- IBM z/OS®
  - IBM z/OS 1.2 or later

2.2.1 Development and runtime requirements

The hardware requirements for using Branch Transformation Toolkit V5.1 are different between development environments and runtime environments. For example, this section talks about the basic requirements for the Microsoft Windows platform to demonstrate how development requirements compare with runtime requirements.

For a Windows client, the minimum requirements for a Branch Transformation Toolkit runtime environment are listed in Table 2-1.

Table 2-1 Client runtime requirements for Branch Transformation Toolkit on Windows

Microsoft Windows 2000 client side	Requirement description
Processor	PII 266 MHz or higher
Memory	48 MB minimum /64 MB recommended
Hard drive	50 M
Display	800 x 600 minimum / 1024 x 768 recommended
Intranet test LAN base	16 MB token ring / 100 MB BaseTx Ethernet recommended
Operating system	Microsoft Windows 2000 Professional with Service Pack 4
Browser	Any of the following: ▶ Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later ▶ Internet Explorer® 6.0 SP1
Communication protocol	TCP/IP
JDK™	Java 2 SDK supplied by WebSphere Application Server

For a Windows server, the minimum requirements for a Branch Transformation Toolkit runtime environment are listed in Table 2-2:

Table 2-2 Server runtime requirements for Branch Transformation Toolkit on Windows

Windows 2000 server side	Requirement description
Processor	PIII 500 MHz or higher
Memory	512 MB minimum / 768 MB recommended

Windows 2000 server side	**Requirement description**
Hard drive	100 MB minimum
Intranet test LAN base	16 MB token ring / 100 MB BaseTx Ethernet recommended
Operating System	One of the following: ► Microsoft Windows 2000 Advanced Server with Service Pack 4 ► Microsoft Windows 2000 Server with Service Pack 4
Application server	One of the following is required: ► IBM WebSphere Application Server - Express V5.1.1 ► IBM WebSphere Application Server V5.1.1 ► IBM WebSphere Application Server Network Deployment V5.1.1 ► IBM WebSphere Application Server for Developers V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1 for Multiplatforms
Communication protocol	TCP/IP
Communication services (LU0, LU6.2)	IBM Communications Server V6.1.2
Database manager	One of the following: ► DB2® UDB Enterprise Server Edition v8.1 FP5 or FP7 ► DB2 UDB Enterprise Server Edition v8.2 ► Oracle 8i Standard/Enterprise Release 3 V8.1.7.4 ► Oracle 9i Standard/Enterprise Release 2 V9.2.0.5 ► Microsoft SQL Server 2000 Enterprise SP3 ► Microsoft SQL Server 2000 Standard Edition

Windows 2000 server side	Requirement description
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

For Windows 2000 development environments, the minimum requirements are listed in Table 2-3:

Table 2-3 Windows requirements for Branch Transformation Toolkit development

Windows 2000 development	Requirement description
Processor	PIII 500 MHz or higher recommended
Memory	1024 MB recommended
Hard drive	2 GB minimum for WebSphere Studio Application Developer; 4.5 GB minimum for WebSphere Studio Application Developer Integration Edition
Display	TCP/IP
Operating system	100 MB Ethernet recommended
Integrated development environment	One of the following: ► IBM WebSphere Studio Application Developer V5.1.1 ► IBM WebSphere Studio Application Developer Integration Edition V5.1.1
Browser	► Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later ► Internet Explorer 6.0 SP1

2.2.2 Additional requirements

Depending on the framework services you use, you might require other hardware and software to support the financial devices. The additional requirements shown in Table 2-4 apply to the type of workstation (client, server, or development) that accesses the financial device.

Table 2-4 Additional requirements for framework services

Framework component	Additional requirements
J/eXtensions for Financial Services	Any financial printer, magnetic stripe reader/encoder, or check reader with a device service that is compliant with the J/XFS™ specification
eXtensions for Financial Services	Any financial printer, magnetic stripe reader/encoder, or check reader with a device service that is compliant with the J/XFS specification
LANDP® MSR/E Device Service	Any magnetic stripe reader/encoder supported by the LANDP MSRE47## server

2.3 Migration stages

After you make the decision to migrate your toolkit application from Branch Transformation Toolkit 4.3 to version 5.1, you should come up with a detailed plan that includes all the required phases. This section details the different stages of the migration process and identifies all the important steps. The migration process is divided into three stages:

1. The premigration stage, where you determine the scope of the migration effort required
2. The migration stage, where you perform the code and infrastructure migration
3. The post-migration stage, where you tune the migrated solution for scalability and performance

Figure 2-1 shows the three migration stages.

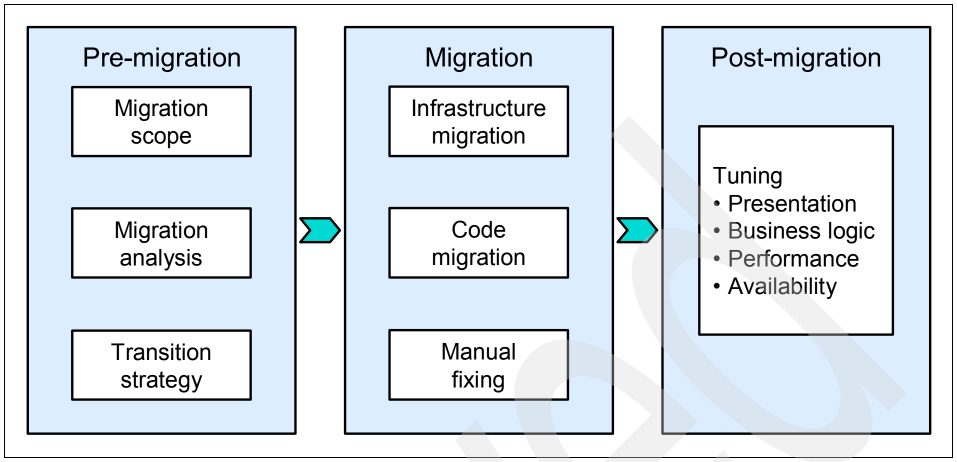


Figure 2-1 Stages of migration

2.3.1 Premigration stages

Before the migration, assess the architecture, infrastructure, process flows, and code functionality of your Branch Transformation Toolkit 4.3 application, besides all ongoing information technology projects in your organization. The goal of the premigration stage is for you to gain a deep understanding of your original application so that you can determine your migration and redevelopment approach.

The premigration details are described in 2.4, "Things to do before migration" on page 29.

2.3.2 The migration stage

In the migration stage, the fundamental migration efforts are performed. The basic work can be done by the migration tool, which can also be customized to meet your specific migration needs. The migration tool is embedded in the Branch Transformation Toolkit. It has a graphical presentation and is easy to use. The primary migration tasks are centered around reengineering the application so that it runs in Branch Transformation Toolkit V5.1. You should also redevelop the application code to comply with the new features and the technology provided by the new version of the Branch Transformation Toolkit. The exact way in which you carry out the migration tasks will depend on the migration and redevelopment approach you defined in the premigration stage.

2.3.3 Post-migration stage

Because of the new technologies that have been adopted in the latest version of the Branch Transformation Toolkit, some components are dramatically changed. The migration tool does not migrate everything in the old version of your application, including the logic inside the application, server operation, actions of flow process and services, and so on. For these application contents, you must perform manual modification and redevelopment. The server side event mechanism and communication services are also changed in V5.1 of the Branch Transformation Toolkit, and the migration tools do not support automatic migration of these features.

After completion of post-migration activities, the migrated application should run in the Branch Transformation Toolkit 5.1 runtime environment. Although the migration has taken place in effect, you must make sure that the application performance meets at least the migration objectives, in addition to making the code run. Some essential tuning at the code and runtime levels is required.

Because each migration scenario is unique to each different site, we do not tell you exactly what your migration strategies should be in this book. Instead, we walk you through the process of identifying what you must do to set the migration scope, and conclude by providing a set of recommendations that are helpful in planning and executing a migration part.

Details of these guidelines are discusses in the remaining chapters of this book.

2.4 Things to do before migration

There are several activities that should be undertaken at the beginning of a migration effort. The assessment of migration complexity is an important step that should *not* be skipped, since it can be used to set the stage for the steps that follow. Premigration activities include:

- Assessment
- Education
- Installation of a new development environment
- Installation of a runtime environment
- Research
- Code preparation
- Planning

2.4.1 Migration assessments

Before attempting migration, assess the efforts. The time required for migrating an application is determined by a number of factors. The most significant ones are the complexity of the application and the runtime configuration.

Generally, assessing the migration complexity for a single business logic processor running on a single runtime configuration, is likely to take a day or two. A more complex application running in a complex runtime configuration involving both presentation and business process requires more time. As a guide for your time assessments, allow for the following timeframes:

- One day to assess each logic processor
- One to two days to assess the runtime environment
- Two to three days to assess documentation discoveries
- One day to assess the build and deploy processes and for general education

2.4.2 Education

Reengineering is not enough for a migration to work. Education and training is needed for your employees to work with the Branch Transformation Toolkit 5.1. The amount of training needed depends on the nature of the competitive product that the new version of Branch Transformation Toolkit is replacing. There are a number of different skill sets to consider. In particular, you should think about the following:

- J2EE architecture and coding
- The new solution architecture of a toolkit application
- New concepts and technologies
- Runtime environment configuration
- Enhanced components in the new version of Branch Transformation Toolkit

A new development environment also comes with the new version of Branch Transformation Toolkit. Developers familiar with the old version of the Branch Transformation Toolkit will benefit from some amount of tools training.

In any case, it is best that specific needs are assessed against actual skills and the gaps filled. Migration provides a good opportunity to assess the existing skills of your team and fill the gaps before moving forward. The amount of training you need depends on your goals. In general, we recommend that you meet the following objectives through classroom education:

- Leverage the core IBM application development platform.
- Make developers more familiar with tool capabilities.
- Stay in synch with current and evolving standards, open source, and platform evolution.

- Provide a path forward for clients with limited J2EE development experience, who must, at the same time, develop transactional business applications quickly.

When migrating to Branch Transformation Toolkit 5.1, assume that classroom education will take at least a week for developers already familiar with version 4.3. Expect that developers will require at least a few weeks to become familiar with the new environment and return to full productivity.

2.4.3 Installation of the new development environment

Before any application code is modified, a suitable development environment must be installed. For Branch Transformation Toolkit 5.1, you can choose either WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition.

Whatever your chosen development environment, you should spend some time researching, installing, and configuring source code management tools.

There are new hardware and software requirements for the development environments of Branch Transformation Toolkit 5.1. You should first read the requirements and ensure that your environments match.

Although the Branch Transformation Toolkit V5.1 has major changes compared with version 4.3, some operation concepts remain similar. If you are already familiar with version 4.3 and have general knowledge of J2EE, you should become comfortable with the changes relatively quickly.

Your development environments comprise more than just an integrated development environment (IDE). Other tools that you use are also important and must be reviewed as part of the migration. In some cases, new versions of these development tools might suffice or complete replacements might be required. Other aspects to consider as part of the migration effort include testing the servers, the development process, and the development practices and methodologies.

Research might be required to handle special considerations such as the use of special tools. A development environment also comprises testing the servers, databases, and other systems. These must also be considered a part of the migration effort. Developers should not immediately be expected to be 100% productive in the new environment. They will need some time to acquaint themselves with the changes.

2.4.4 Installation of a runtime environment

As part of the migration effort it is necessary to install a new Branch Transformation Toolkit runtime environment for application execution purposes. As part of the installation, runtime environment working staff will learn the nuances of the new version of Branch Transformation Toolkit 5.1.

Even the most well-written code will not run without a properly configured application server. Setting up good production and testing environments is a key requirement for success. Configuring the production environment can be quite complex, so starting from scratch and installing and configuring a Branch Transformation Toolkit runtime environment might take significant time.

Even in normal conditions, the time taken to set up a runtime environment is not short. Therefore, the correct migration approach requires that you interfere with the current system as little as possible. The time and complexity involved in setting up the new runtime depends on the following factors:

- How much additional hardware can you purchase?
- How is the existing environment configured?
- Does the existing environment provide failover support?

We suggest that you use a trial migration of a system runtime environment to learn the ropes and make mistakes. Based on the lessons learned through this migration, formulate a plan to effect the entire migration and test that plan against the QA environment. If possible, you should also consider migrating your configuration in vertical slices, that is, measure the time required to migrate a single slice and use that information to estimate the entire migration effort.

2.4.5 Research

As described in 2.4.1, "Migration assessments" on page 30, research is typically a part of every migration assessment effort. For example, migrated applications sometimes use code libraries provided by a third party. You should confirm that these libraries will function on Branch Transformation 5.1 or that there are newer versions of these libraries. In any case, the third-party vendor can be required to provide some kind of certification that these libraries will function in the new environment.

In the same way, some functions in the old toolkit application might have been replaced with other mechanisms or maybe done in a different manner in the new version. You should therefore study the current technology in Branch Transformation Toolkit V5.1 to decide on the right transition path to use in your migration.

2.4.6 Code preparation

The amount of time you must allocate to coding practices is a significant factor to consider when evaluating migration complexity. To migrate application code more easily and more efficiently, we recommend that you refactor the code before migration. If you apply the refactoring discipline properly, you can handle code migration more easily because the code will be simple and clean. Regardless of the development methodology you choose, refactoring improves the overall code quality. Migration is good time to reexamine and refactor your code.

To reduce the effort of migration work, remove the code that is not used. If you have completed the refactoring process before migration, it becomes easier to exclude the code that is not a part of the migration effort. The practice of removing what you do not use can also be extended to the runtime environment. Unused resources such as data sources should be removed as a matter of practice.

Coding practices can have a direct impact on the complexity of migration. Well designed and architected coding is generally easier to understand and modify. Layered architectures are a huge win when change is required.

The new version of Branch Transformation Toolkit also supports more recent versions of the relevant specifications. In anticipation of an upcoming migration, you can update the existing code to conform to the most recent version. Branch Transformation Toolkit 5.1 supports multiple versions of the related standards such as J2EE and BPEL, but in anticipation of a migration, you may consider updating your application to comply with the most recent versions of the specifications that run on Branch Transformation Toolkit V5.1.

2.4.7 The migration plan

The migration plan is what should guide you during the entire migration process. The various tasks at every stage should be captured and documented in the plan and we recommend that all migration activities follow the plan.

However, the migration plans should also be flexible. As part of the migration assessment, you should document the risks and unknowns that can impact your plan. Assume that uncertain issues will occur during the migration process. For example, a common mistake is factoring in some time for testing but not enough time for fixing problems. Be sure to allocate enough time in the plan to fix the problems encountered.

Scheduling is an important part of the planning process. It helps chain migration tasks together. Remember that many tasks can be completed in parallel.

Figure 2-2 shows the skeleton of a high-level migration schedule. Use this as a guide to make your own schedule appropriate to the situation for your site.

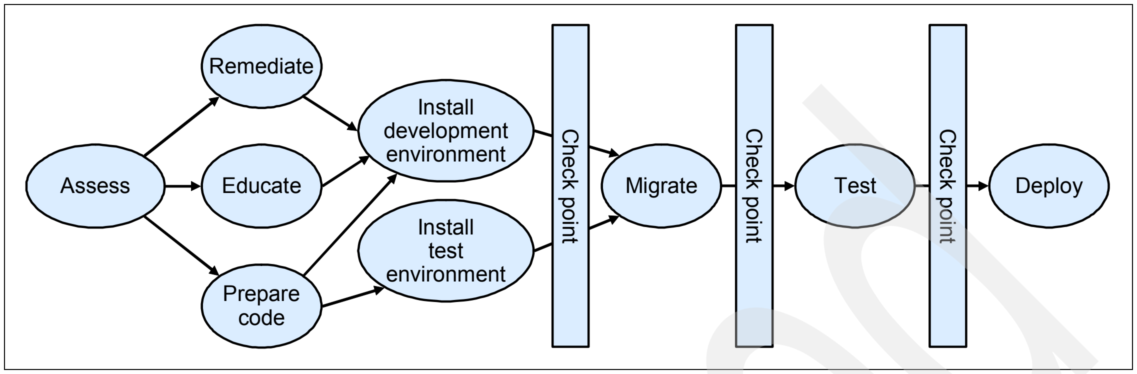


Figure 2-2 Sample migration schedule

Documentation is an important part of the planning process. It helps make things simple by separating the description of tasks from the scheduling of their application. There are a number of popular ways in which to capture this information. One approach is to break the migration problem into reasonable-sized tasks and then compose a task flow, for example, consider education as a single task. Estimate the effort required and identify the prerequisite tasks and post conditions that should be met for each task.

A sample collection of project tasks is shown in Table 2-5. The tasks identified in this sample are for our specific migration effort. Table 2-5 is only a sample task list. Use it as a skeleton or a template for a your specific Branch Transformation Toolkit project plan. Your task list will be different, depending on your project and environment. Set the prerequisites for every task.

Table 2-5 Migration task list

Task ID number	Task	Prerequisite task number	Comments
1	Assessment		
2	Education	1	
3	Install development environment	2	
4	Install deployment environment	2	
5	Code preparation	3	

Task ID number	Task	Prerequisite task number	Comments
6	Migrate infrastructure	5	
7	Migrate code	6	
8	Manual fixing	7	
9	Modify presentation layer	8	Post-migration activities
10	Modify business logic layer	8	Post-migration activities
11	Modify client side	8	Post-migration activities
12	Test	11	
13	Deployment	4, 12	

Table 2-5 on page 34 shows only a skeleton for documenting tasks. After you have identified the migration tasks, the next step is to estimate the required time to complete each task. In addition, estimate the number of resources needed for each task and consider if the documentation needs more execution details.

Use the captured task information to generate a schedule. The prerequisites will help you organize the tasks, set completion dates, and allocate resources for the migration effort. In this phase of a migration project, use multiple project management software tools to assist with planning and scheduling the tasks.

2.5 Elements of migration

Migration to Branch Transformation Toolkit V5.1 will have an impact on several aspects of your business. The toolkit applications are far more than just code. Along with the application code, there are development environments, build processes, runtime environments and other aspects that must be considered as part of a migration. At the end of the migration process, the migrated application should run on a production environment, the application should be built and deployed, and developers have to be productive with the new development environment. It is important to consider all the following elements when doing a migration:

- Application code
- Development environment
- Installation of development environment
- Testing
- Runtime environments
- Deployment processes

This part of the chapter discusses each of these elements and considers their place in the entire migration process. In some cases, such as migration prerequisites, we have provided only abstract guidance. However, for some elements, we have provided detailed instructions.

2.5.1 Application code

For a Branch Transformation Toolkit application, the migration from version 4.3 to version 5.1 will have a significant impact on the application code. The exact workload required for code modification depends on the technology adopted by the application. In some cases, deprecated APIs used in the application have to be replaced with new ones.

During the migration process, your staff can follow standard, iterative development practices. Generally, we assign a special team to carry out the migration work. However, we can also involve the application developers.

2.5.2 Development environment

Development environments change over time. The Branch Transformation Toolkit is based on the J2EE standard. The development environment of toolkit applications is integrated with WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition, both of which are based on the Eclipse Platform.

Development environments comprise more than just the IDE you use. The changes necessary for a successful migration will impact several aspects of your development environment because the development platform and the toolkit plug-ins have changed dramatically between Branch Transformation Toolkit V4.3 and V5.1. You should also think about how migration will affect other areas of your development environment such as test servers, or development practices and methodologies.

Figure 2-3 shows the main tools provided as part of the Branch Transformation Toolkit development environment, including the graphical builder, the CHA editor, and the formatter editor.

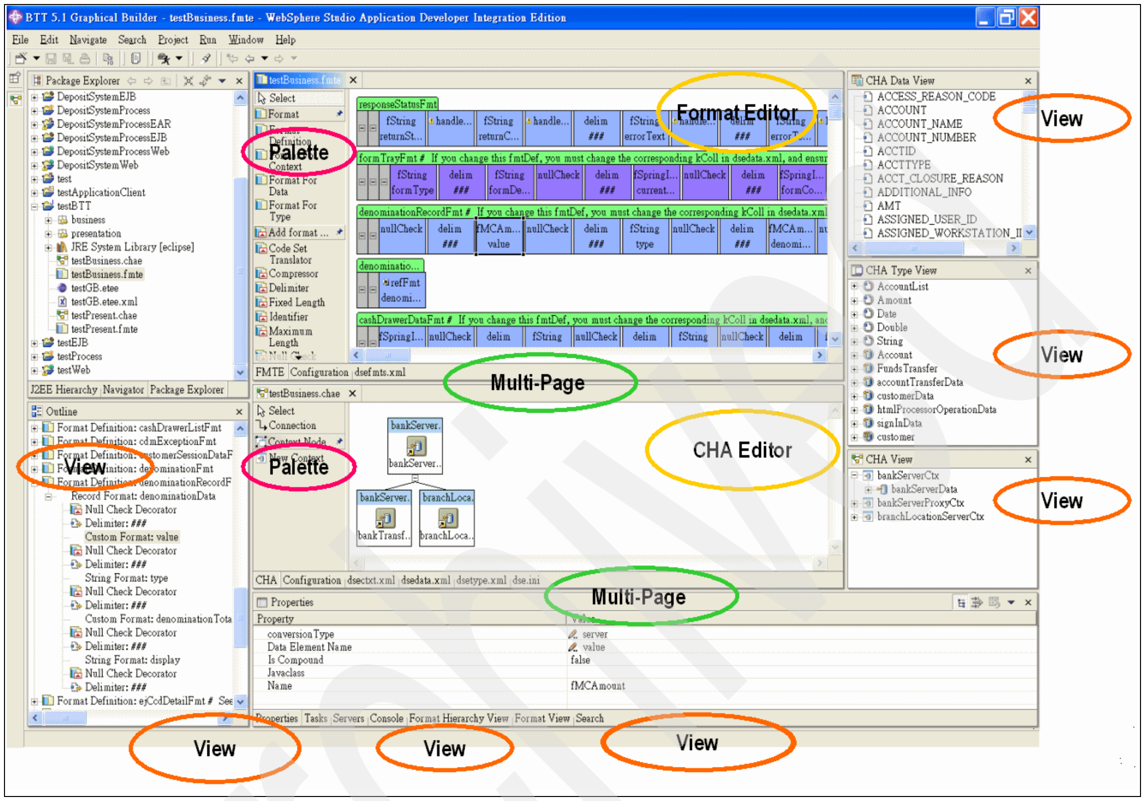

BTT 5.1 Graphical Builder - testBusiness.fmte - WebSphere Studio Application Developer Integration Edition
File Edit Navigate Search Project Run Window Help
Format Editor
View
Palette
Multi-Page
CHA Editor
Palette
View
Multi-Page
View
View
View
View
View
View

Figure 2-3 Branch Transformation Toolkit development tools

Branch Transformation Toolkit 5.1 provides a number of tools that support the development of applications. All the tools are plug-ins of WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition. The key development tools provided by Branch Transformation Toolkit include:

- The Business Process Wizard

  This provides a GUI to help extend your business processes to take advantage of toolkit-specific entities.

> **Note:** This tool is only available when you use WebSphere Studio Application Developer Integration Edition.

- The Graphical Builder

  This provides a set of tools to define the entities required by the applications and to distribute the runtime files. It provides the development environment throughout the development cycle of toolkit applications. It also acts as a portal from which you can start other tools provided by the toolkit.

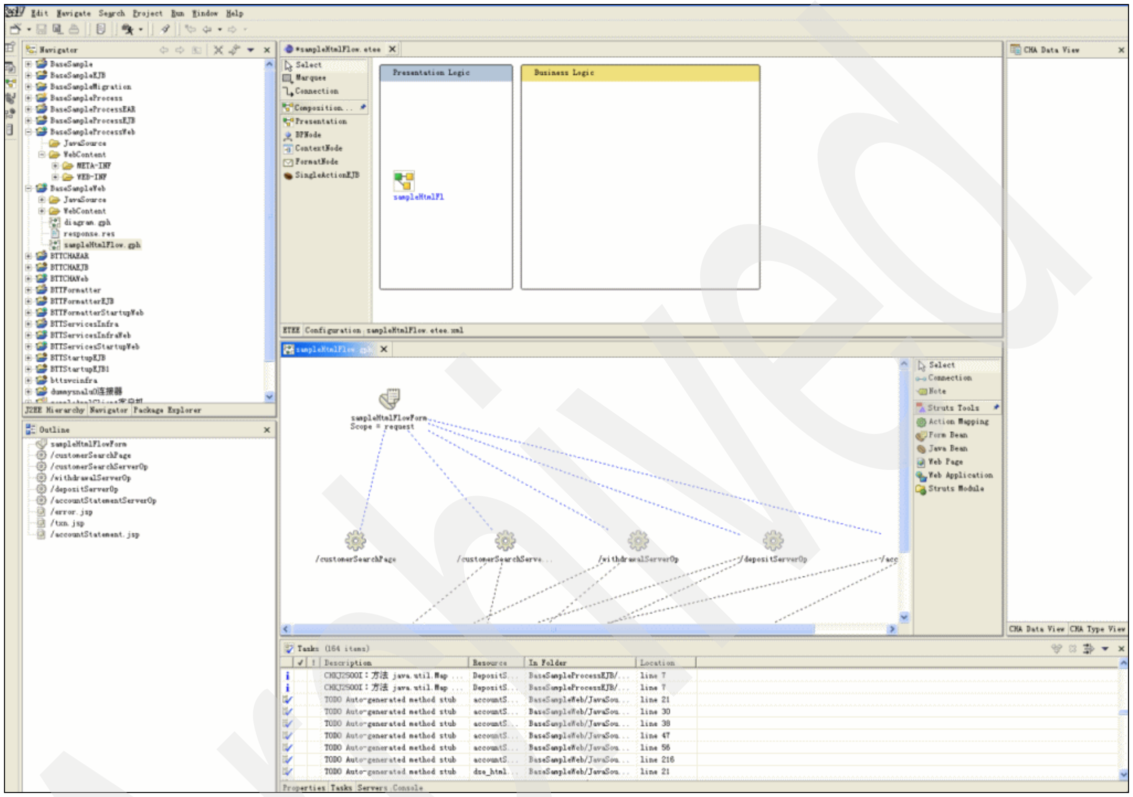
  Note: Some functions of the Graphical Builder are only available when you use the Integration Edition of WebSphere Studio Application Developer.

- The CHA Editor and Formatter Editor

  This provides user-friendly interfaces for creating or maintaining the definitions needed by the applications in the application logic layer.

- The Struts Tools Extension

  This provides a GUI to help you extend your Struts configuration files to take advantage of toolkit-specific entities. Figure 2-4 shows an outline of the Struts extension.



Tasks (164 items)

	✓	!	Description	Resource	In Folder	Location
i			CHKJ2500I：方法 java.util.Map ...	DepositS...	BaseSampleProcessEJB/...	line 7
i			CHKJ2500I：方法 java.util.Map ...	DepositS...	BaseSampleProcessEJB/...	line 7
✓			TODO Auto-generated method stub	accountS...	BaseSampleWeb/JavaSou...	line 21
✓			TODO Auto-generated method stub	accountS...	BaseSampleWeb/JavaSou...	line 30
✓			TODO Auto-generated method stub	accountS...	BaseSampleWeb/JavaSou...	line 38
✓			TODO Auto-generated method stub	accountS...	BaseSampleWeb/JavaSou...	line 47
✓			TODO Auto-generated method stub	accountS...	BaseSampleWeb/JavaSou...	line 56
✓			TODO Auto-generated method stub	accountS...	BaseSampleWeb/JavaSou...	line 216
✓			TODO Auto-generated method stub	dse_html...	BaseSampleWeb/JavaSou...	line 21

Figure 2-4 Struts extension tool

Because Branch Transformation Toolkit 5.1 supports two types of development platforms, there are two sets of plug-ins, including different files:

- Plug-ins forWebSphere Studio Application Developer 5.1.1

  These plugins include components that do not depend on the features of WebSphere Studio Application Developer Integration Edition 5.1.1.

  If you haveWebSphere Studio Application Developer5.1.1 installed on your system, the toolkit installation wizard uses WebSphere Studio Application Developer as your development environment. After the installation, the wizard decides that WebSphere Studio Application Developer is your development environment and automatically copies the plug-ins for WebSphere Studio

Application Developer 5.1.1 to the wstools/eclipse/plug-ins directory of the WebSphere Studio Application Developerinstall location.

- Plug-ins for WebSphere Studio Application Developer Integration Edition 5.1.1

  These plug-ins include components that do not have dependencies on the features provided by WebSphere Studio Application Developer Integration Edition 5.1.1, such as plug-ins for the Business Process Wizard.

  If you have WebSphere Studio Application Developer Integration Edition 5.1.1 installed on your system, the toolkit installation wizard takes WebSphere Studio Application Developer Integration Edition as your development environment, even if you also have WebSphere Studio Application Developer installed. After the installation, the wizard decides that WebSphere Studio Application Developer Integration Edition is your development environment and automatically copies the plug-ins for WebSphere Studio Application Developer Integration Edition 5.1.1 to the wstools/eclipse/plug-ins directory of the WebSphere Studio Application Developer Integration Edition 5.1.1 install location.

If neither WebSphere Studio Application Developer 5.1.1 nor WebSphere Studio Application Developer Integration Edition 5.1.1 is installed on your system, you should copy the plug-ins to the wstools/eclipse/plug-ins directory after you install WebSphere Studio Application Developeror WebSphere Studio Application Developer Integration Edition. Plug-ins for WebSphere Studio Application Developercan be found in the <toolkit_root>/plug-ins/wsad51 directory, and plug-ins for the WebSphere Studio Application Developer Integration Edition can be found in the <toolkit_root>/plug-ins/wsadie51 directory.

Apart from the development tools listed above, the toolkit also provides a migration tool to help you migrate your toolkit applications developed with version 4.3 of the toolkit to the new version 5.1 architecture.

2.5.3 Installation of development environments

To set up the development environments, install the required tools on your preferred development workstation so that you can develop applications based on Branch Transformation Toolkit 5.1.

Notes: The physical machine must comply with the requirements listed in 2.2, “Hardware and software requirements” on page 23.

The following procedure describes how to install the Branch Transformation Toolkit 5.1 on top of WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition. Each functional unit is contained in its own JAR file to provide greater flexibility in both the development and runtime environments. Consider reviewing functional units, packages, and dependencies to decide which functional units you need in order to develop your application.

You can add or remove functional units at any time, provided you account for their corequisite functional units, that is, functional units that must also exist on the system at the same time.

Installing the Branch Transformation Toolkit

To set up your development workstation, perform the following tasks:

1. According to your business need, first install either WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition.
2. Insert the Branch Transformation Toolkit CD into a drive on the workstation and browse the CD. If you want to install Branch Transformation Toolkit 5.1 for Windows, run `C55HDML.exe` from the Windows platform directory on the installation CD. If you want to install Branch Transformation Toolkit 5.1 for Linux, run `C55HEML.bin` from the Linux platform directory. Either command starts the Installation Wizard for Branch Transformation Toolkit.

   During the installation, the Installation Wizard checks for previous versions of the Branch Transformation Toolkit already installed on your system. Version 5.1 can coexist with version 4.3. If you already have version 5.1 installed, the Installation Wizard displays a warning that you already have version 5.1 installed. If you continue with the installation, your previous installation of version 5.1 will be overridden.

   The Installation Wizard also checks for the edition of WebSphere Studio you have installed. If you have installed WebSphere Studio Application Developer, the Installation Wizard installs the toolkit components designed for WebSphere Studio Application Developer and those for WebSphere Application Server. If you have WebSphere Studio Application Developer Integration Edition installed, the Install Wizard installs the toolkit components designed for WebSphere Studio Application Developer Integration Edition, and prompts you on which set of runtime components you want to install, either the components for the WebSphere Application Server or the components for WebSphere Business Integration Server Foundation.

Table 2-6 shows the details of the directories created by the Branch Transformation Toolkit installation program.

Table 2-6 Branch Transformation Toolkit installation directories

Directory name	Description of contents
dbtools	Scripts to manage database tables for the CHA component
desktop	Desktop DTD file
doc	The Branch Transformation Toolkit documentation plug-ins for WebSphere Studio Application Developer Integration Edition
plug-ins	► Visual beans plug-in for WebSphere Studio Application Developer Integration Edition ► Eclipse Modeling Framework (EMF) plug-in ► Graphical Builder plug-in ► Struts Tools Extensions plug-in ► CHA Editor plug-in ► Format Editor plug-in ► Business Process Wizard plug-in ► Migration Tool plug-in
ear	EAR files to provide the infrastructure for the CHA, CHA formatter service, and Service Infrastructure components of the toolkit. You can use the EARs to build applications on the Branch Transformation Toolkit 5.1.
jars	The Branch Transformation Toolkit code separated into various JARs according to the functional unit to which the code belongs. A solution built on the Branch Transformation Toolkit 5.1 can use the JARs for the functional units that it is using. Check the functional units, packages, and dependencies for a listing of the JARs and their contents and co-requisites.
samples	EAR files to run the sample applications provided by the Branch Transformation Toolkit 5.1 in the WebSphere Studio Application Developer workbench. This directory also contains the source code of the samples.
services	Runtime files that are needed by some of the services of the framework.

Directory name	Description of contents
source	Source code of selected Branch Transformation Toolkit 5.1 components to provide a better understanding of the functional units. This helps in tasks such as extending the framework and reduces the development cycle. The source code must not be modified.
Java doc	Java reference documentation

Figure 2-5 shows a view of the Branch Transformation Toolkit installation directories.

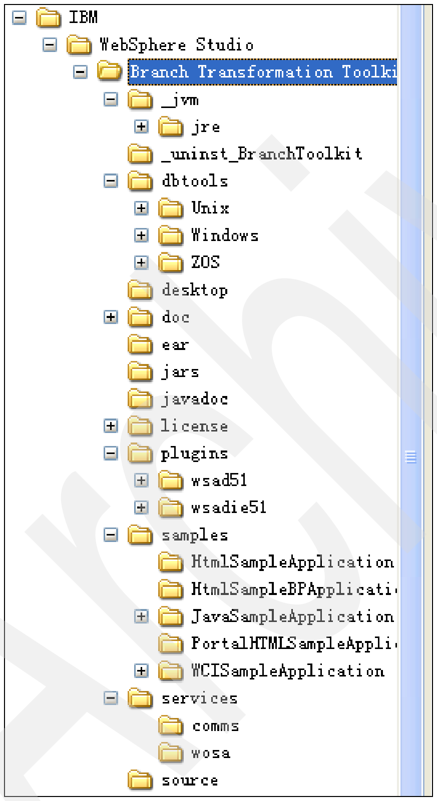
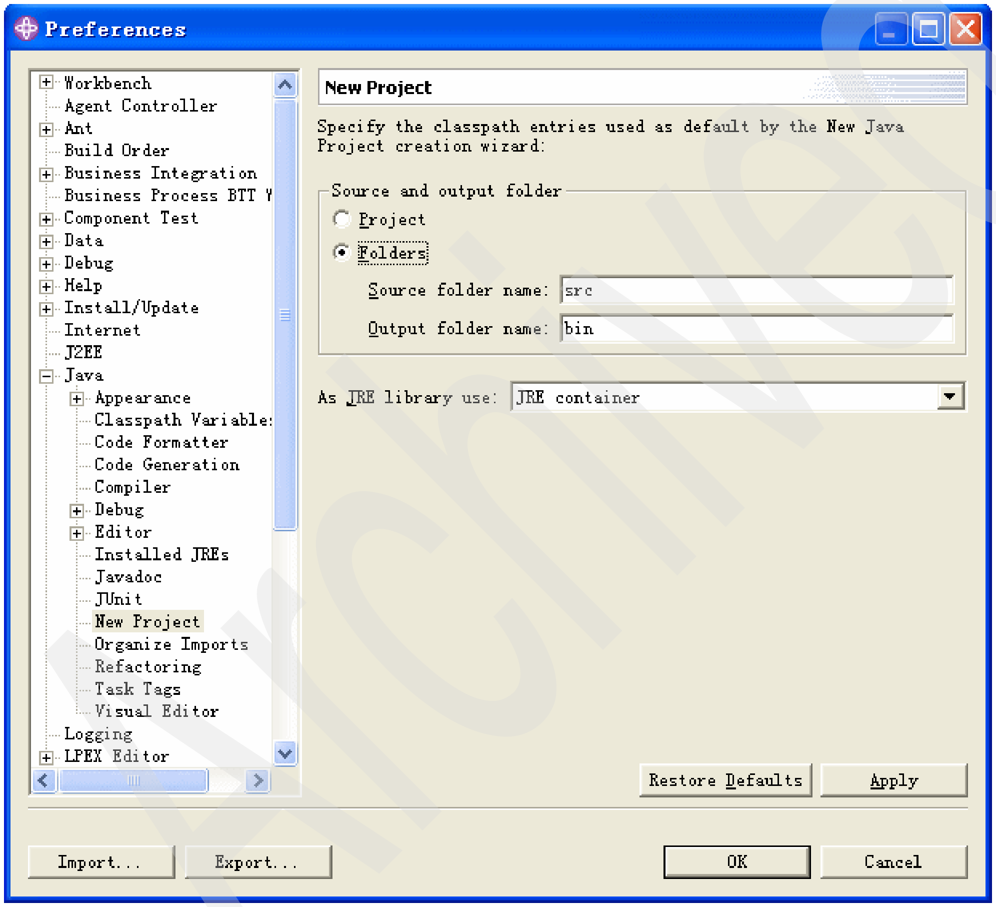


Figure 2-5 A view of the BTT installation directories

3. Start WebSphere Studio Application Developer.

4. Set some preferences before you import the Branch Transformation Toolkit V5.1 Java source:

   – To work with complex projects, use the source folders as source containers instead of creating packages directly inside the project. To do this, create source folders as children of the project and create the packages inside these source folders. To automate this, select **Window** → **Preferences**. Expand the **Java** node and select the **New Project** node. Enable the **Folders** check box, as shown in Figure 2-6.


Preferences

Workbench
Agent Controller
Ant
Build Order
Business Integration
Business Process BTT Y
Component Test
Data
Debug
Help
Install/Update
Internet
J2EE
Java
Appearance
Classpath Variable:
Code Formatter
Code Generation
Compiler
Debug
Editor
Installed JREs
Javadoc
JUnit
New Project
Organize Imports
Refactoring
Task Tags
Visual Editor
Logging
LPEX Editor

New Project

Specify the classpath entries used as default by the New Java Project creation wizard:

Source and output folder
( ) Project
(●) Folders
Source folder name: src
Output folder name: bin

As JRE library use: JRE container

Restore Defaults | Apply

Import... | Export... | OK | Cancel

Figure 2-6 Source and output folder preferences

   – To develop with Branch Transformation ToolkitV5.1, add the framework functional units, the JAR files containing the classes, to the application classpath. You can use different approaches for this. However, you should consider the impact of the class loader policies.

We recommend that you add an overall classpath variable named `BTT` to point to the root directory of the Branch Transformation Toolkit installation. To do this, perform the following steps:

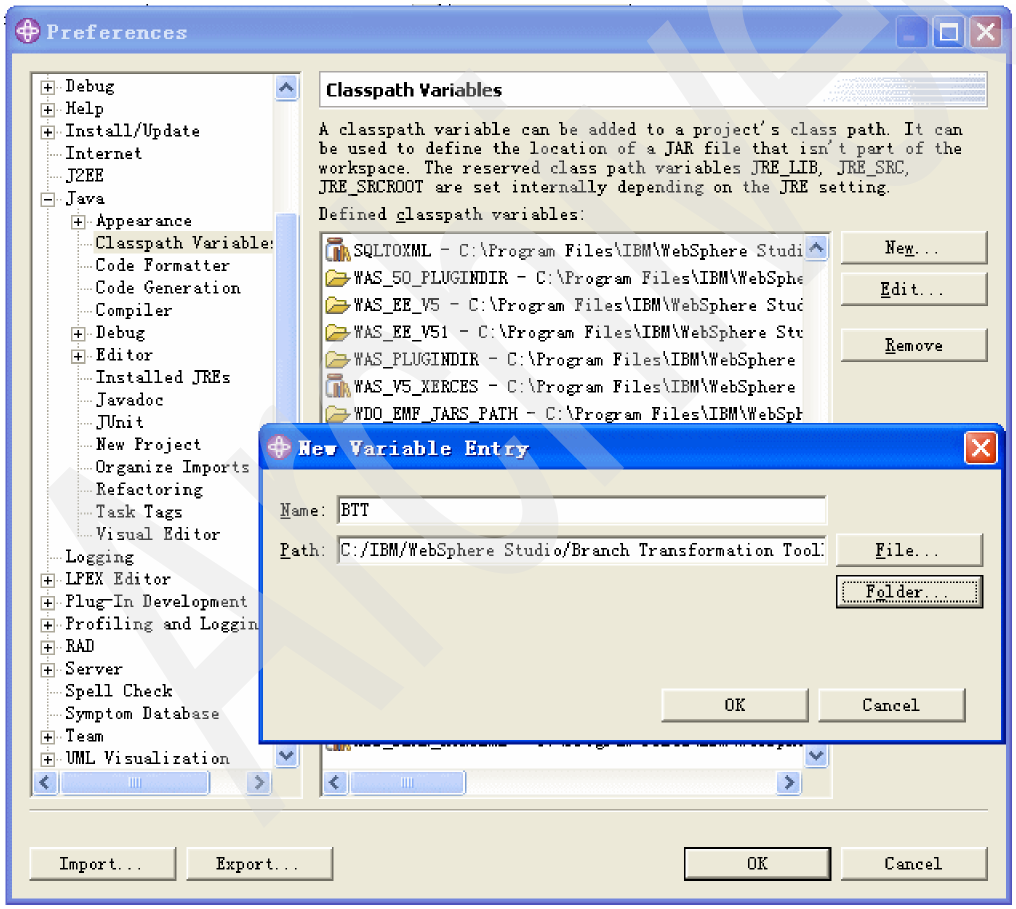
i. Select **Window** → **Preferences**.

ii. Expand the **Java** node and select **Classpath Variables**.

iii. Click **New** and in the Name entry field, enter `BTT`.

iv. In the Path entry field, type the path where the Branch Transformation Toolkit is installed, as shown in Figure 2-7.

v. Click **OK**.



A classpath variable can be added to a project's class path. It can be used to define the location of a JAR file that isn't part of the workspace. The reserved class path variables JRE_LIB, JRE_SRC, JRE_SRCROOT are set internally depending on the JRE setting.

Defined classpath variables:

```
SQLTOXML - C:\Program Files\IBM\WebSphere Studi
WAS_50_PLUGINDIR - C:\Program Files\IBM\WebSphe
WAS_EE_V5 - C:\Program Files\IBM\WebSphere Stud
WAS_EE_V51 - C:\Program Files\IBM\WebSphere Stu
WAS_PLUGINDIR - C:\Program Files\IBM\WebSphere
WAS_V5_XERCES - C:\Program Files\IBM\WebSphere
WDO_EMF_JARS_PATH - C:\Program Files\IBM\WebSph
```

Name: BTT

Path: C:/IBM/WebSphere Studio/Branch Transformation Tool

Figure 2-7 Creating a classpath variable

In the same way, add another classpath variable, for example, `BTT_EXTERNAL` to point to the external dependencies. To make the external classpath variable work, all the required external JARs must be in the directory to which this variable points. Using classpath variables makes it easier for you to select the Branch Transformation Toolkit JAR files for your Java project.

Another approach is to define a classpath variable for each functional unit available in the product. Each classpath variable points to the concrete JAR associated with the functional unit.

Note: The rest of this procedure and the other procedures assume that you are using our recommended approach to setting the classpaths.

5. Create a Java project for an application by performing the following tasks:
   a. Select **File** → **New** → **Project**.
   b. Select **Java** in the left panel and then select **Java Project** in the right panel. Click **Next**.
   c. Type the name of the project, such as `BaseSample`, and click **Finish**.
6. When deploying an application, you must embed Branch Transformation Toolkit5.1 functional units along with the required application resources, inside the J2EE EAR file. The Branch Transformation Toolkit V5.1 uses functional units to provide a coherent structure of JAR files you use to build an application. The JARs in the functional units contain the required set of classes for a given execution environment.

   To make the functional units available to your project, you can either add the compiled JARs to the project's classpath or you can import the JARs into the workspace. You can choose the second option if you want to embed the JARs and resources into your application when, for example, it is self-contained and independently deployed.

   For either option, you add or import the JARs and their corequisites needed for the application, only to optimize the deployment and distribution of the application.

   If you are unsure about the actual dependencies among the JARs, add all the JARs that you believe might be needed for your application. You can always remove the unwanted JARs later. To get an idea of the JARs you should select, browse the list of JARs used by the sample applications shipped in the <Toolkit root>/samples folder. For example, if you open the BTTHtmlSampleWeb.war file that is inside the BBTTHtmlSample.ear file, you will see that the following functional units have been used to build the application:

- BTTBase (bttbase.jar)
- BTTInvoker (bttinvoker.jar)
- BTTJavaClient (bttjavaclient.jar)
- BTTServerBean (bttsvrbean.jar)
- BTTSessionManagement (bttsm.jar)
- BTTStrutsExtension (bttstruts.jar)
- BTTHTMLSampleEJB (BTTHTMLSampleEJB.jar)

To add the required JARs to the project's classpath, perform the following tasks:

a. Right-click the project and select **Properties**.

b. Select **JavaBuildPath** and then select the **Libraries** tab.

c. Select **Add Variable** and, in the New Variable Classpath Entry window, select the classpath variable (`BTT`) and click **Extend**, as shown in Figure 2-8.

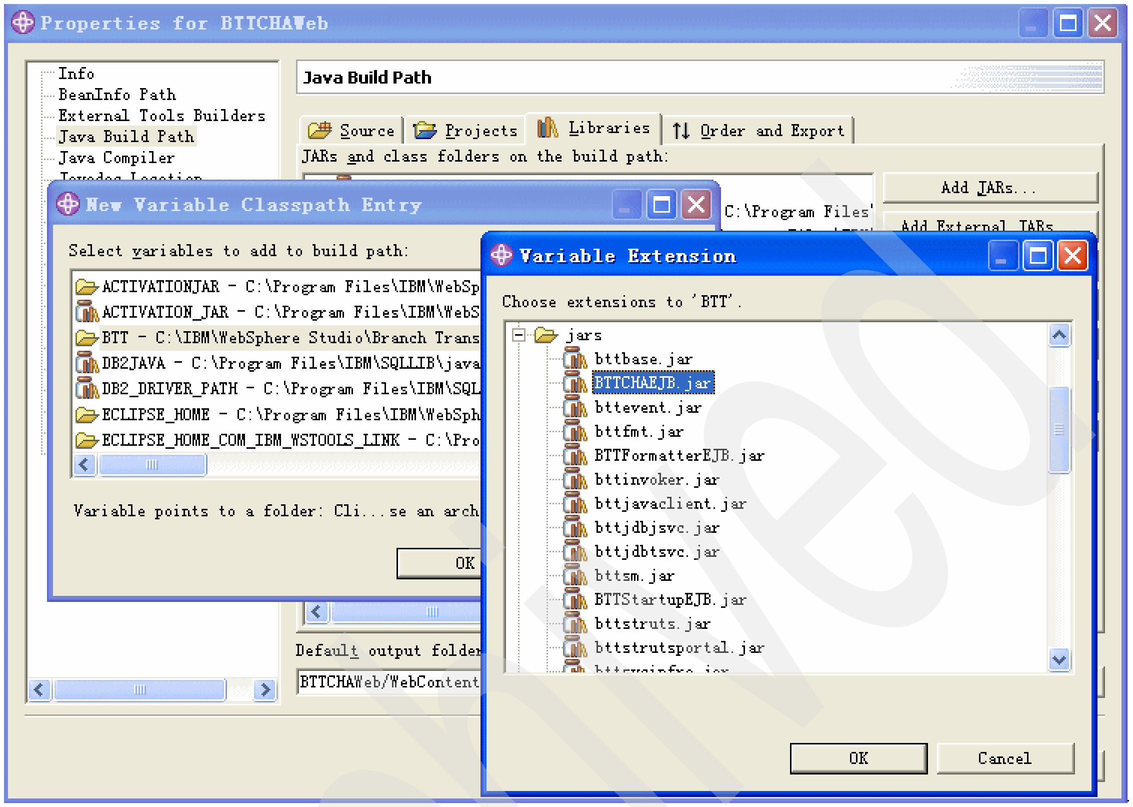

Properties for BTTCHAWeb
Info
BeanInfo Path
External Tools Builders
Java Build Path
Java Compiler
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
C:\Program Files'
Add JARs...
Add External JARs
New Variable Classpath Entry
Select variables to add to build path:
ACTIVATIONJAR - C:\Program Files\IBM\WebSp
ACTIVATION_JAR - C:\Program Files\IBM\WebS
BTT - C:\IBM\WebSphere Studio\Branch Trans
DB2JAVA - C:\Program Files\IBM\SQLLIB\java
DB2_DRIVER_PATH - C:\Program Files\IBM\SQL
ECLIPSE_HOME - C:\Program Files\IBM\WebSph
ECLIPSE_HOME_COM_IBM_WSTOOLS_LINK - C:\Pro
Variable points to a folder: Cli...se an arch
OK
Default output folder
BTTCHAWeb/WebContent
Variable Extension
Choose extensions to 'BTT'.
jars
bttbase.jar
BTTCHAEJB.jar
bttevent.jar
bttfmt.jar
BTTFormatterEJB.jar
bttinvoker.jar
bttjavaclient.jar
bttjdbjsvc.jar
bttjdbtsvc.jar
bttsm.jar
BTTStartupEJB.jar
bttstruts.jar
bttstrutsportal.jar
OK
Cancel

Figure 2-8 Add JARs by extending a variable

d. In the Variable Extension window, expand the jars directory and select the JAR you want to add to your project. Click **OK**.

e. Repeat this procedure for each JAR file you want to add. Keep in mind the dependencies that a JAR or functional unit can have.

After you have installed the Branch Transformation Toolkit V5.1 in WebSphere Studio Application Developer Integration Edition, start developing Branch Transformation Toolkit solutions. For a high-level overview of the development process and details about where to get more information, refer to Chapter 8, “Building an application with Branch Transformation Toolkit V5.1” on page 225. For examples of Branch Transformation Toolkit based applications, including how to install and run them on the various supported platforms, refer to the documentation for Java and HTML sample applications that ship with the Branch Transformation Toolkit.

Setting up the CHA Editor

The CHA Editor for theBranch Transformation Toolkit is a WebSphere Studio plug-in. CHA Editor configuration files or CHA Editor files help you create CHA elements with a GUI.

If you installed the toolkit before installing the WebSphere Studio Application Developer Integration Edition, copy the following plug-in files to the wstools\eclipse\plugins\ folder of your WebSphere Studio installation folder:

- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.chaeditor.model.emf_5.1.0
- com.ibm.btt.tools.chaeditor_5.1.0

To create a CHA Editor file, perform these tasks:

1. Start WebSphere Studio Application Developer Integration Edition.
2. Create a simple project to contain the CHA Editor files.
3. From the File menu, select **File** → **New** → **Other**.
4. In the dialog box, select **IBM Branch Transformation Toolkit** in the left panel.
5. In the right panel, select **CHA Editor file,** starting the CHA Editor Configuration Wizard.
6. Select the project to hold the CHA Editor's files. Usually, this is the project file for the application you are creating.
7. In the file name field, enter the name of the editor's configuration file. The file must have the .chae extension.
8. Click **Finish**.
9. From the menu bar, select **Window** → **Show view** → **Other**.

10. In the window that opens, expand **IBM Branch Transformation Toolkit**, and select the **CHA Editor views** you want to show.

    The wizard then creates the configuration file and the dsedata.xml, dsetype.xml, and dsectxt.xml files in the project folder. It then launches the CHA Editor, as shown in Figure 2-9.

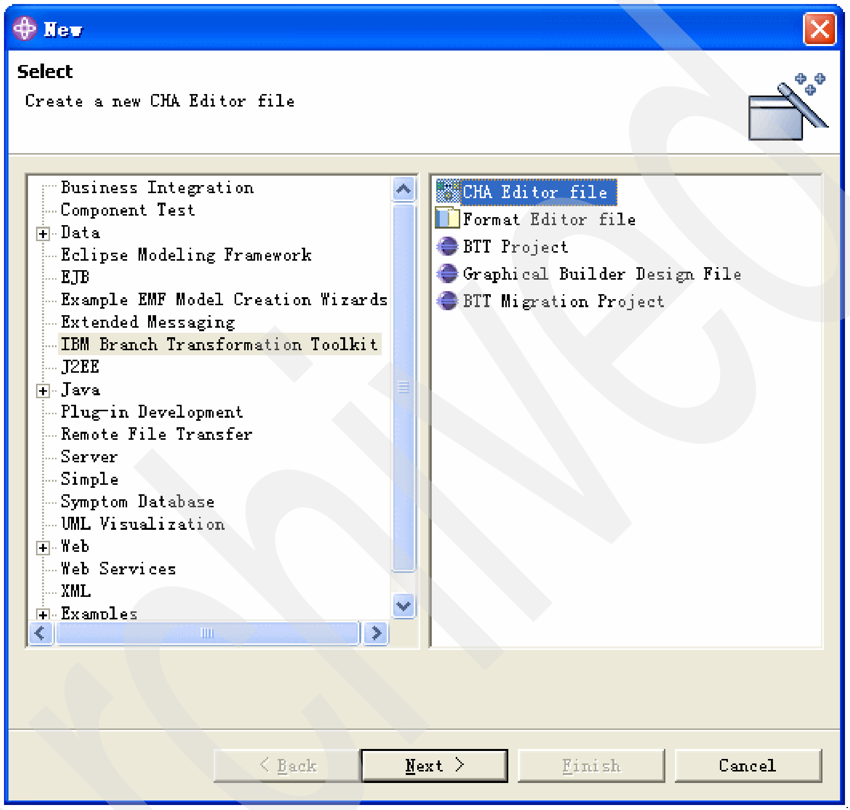


Figure 2-9 Set up CHA editor file

Setting up the Format Editor

The Format Editor for the Branch Transformation Toolkit is a WebSphere Studio plug-in. Format Editor configuration files or Format Editor files help you create formatters with a GUI.

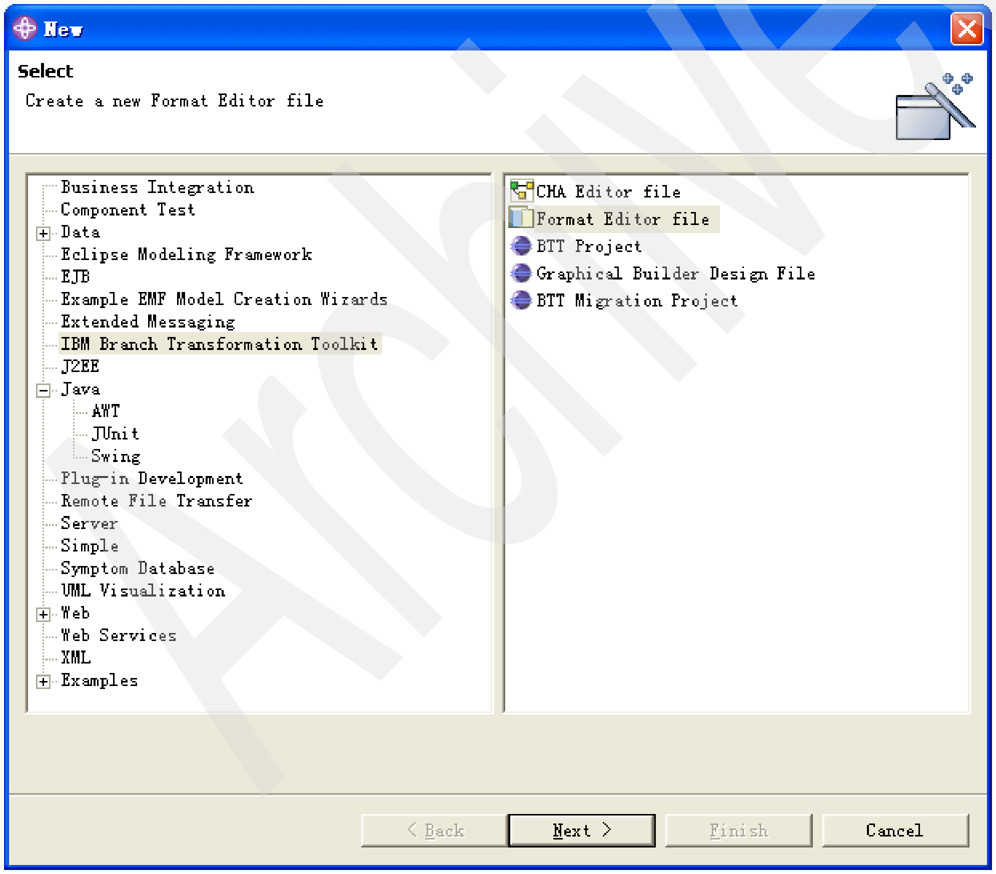
If you installed the toolkit before installing WebSphere Studio Application Developer Integration Edition, copy the following plug-in files to the wstools\eclipse\plugins\ folder of your WebSphere Studio installation folder:

- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.fmteditor_5.1.0

- com.ibm.btt.tools.chaeditor_5.1.0

To create a Format Editor file, perform these tasks:

1. Start WebSphere Studio Application Developer Integration Edition.
2. Create a simple project to contain the Format Editor files.
3. From the File menu, select **File** → **New** → **Other**.
4. In the dialog box , select **IBM Branch Transformation Toolkit** in the left panel.
5. In the right panel, select **Format Editor file**. This starts the Format Editor Configuration Wizard, as shown in Figure 2-10.
6. Select the project to contain the Format Editor's files. Usually, this is the project file for the application you are creating.


New

Select

Create a new Format Editor file

- Business Integration
- Component Test
- Data
- Eclipse Modeling Framework
- EJB
- Example EMF Model Creation Wizards
- Extended Messaging
- IBM Branch Transformation Toolkit
- J2EE
- Java
  - AWT
  - JUnit
  - Swing
- Plug-in Development
- Remote File Transfer
- Server
- Simple
- Symptom Database
- UML Visualization
- Web
- Web Services
- XML
- Examples

- CHA Editor file
- Format Editor file
- BTT Project
- Graphical Builder Design File
- BTT Migration Project

Figure 2-10 Format editor wizard

7. In the file name field, enter the name of the editor's configuration file. The file must have the .fmte extension.

Note: Each Format Editor file works with a CHA Editor file to provide the formatters for the CHA elements described in the CHA Editor file. Define the CHA Editor file name in the configuration of the Format Editor file to ensure that they can work together.

8. Click **Finish**.
9. From the menu bar, select **Window** → **Show view** → **Other**.
10. In the window that pops up, expand **IBM Branch Transformation Toolkit**, and select the **Format Editor** views you want to show, as shown in Figure 2-11.

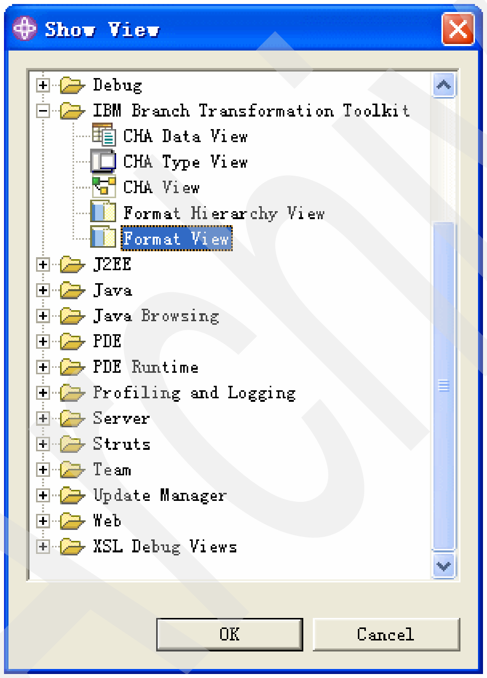


Figure 2-11 Format editor views

The wizard then creates the configuration file and the dsefmt.xml file in the project folder. It then launches the Format Editor.

Setting up the Business Process Wizard

The Business Process Wizard for the Branch Transformation Toolkit is a WebSphere Studio plug-in. The Business Process BTT Wizard provides a GUI to help you extend your business processes to take advantage of the BTT Abstract Layer.

If you installed the toolkit before installing WebSphere Studio Application Developer Integration Edition, copy the following plug-in files to the eclipse\wstools\plugins\ folder of your WebSphere Studio installation folder:

- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.bp_5.1.0

Setting up the Struts Tools Extensions

The Struts Tools Extensions for the Branch Transformation Toolkit is a WebSphere Studio plug-in. The Struts Tools Extensions provides a graphical and easier way to work with toolkit-extended Struts configuration files.

If you installed the toolkit before installing the WebSphere Studio Application Developer Integration Edition, copy the following plug-in files to the wstools\eclipse\plugins\ folder of your WebSphere Studio installation folder:

- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.struts_5.1.0
- com.ibm.btt.tools.webdiagrameditor_5.1.0

Setting up the Graphical Builder

The Graphical Builder for the Branch Transformation Toolkit is a WebSphere Studio plug-in. The Graphical Builder integrates all the key development tools and provides application visualization, integrated development, and seamless deployment techniques that can be applied to the full life cycle of your application development.

If you installed the toolkit before installing the WebSphere Studio Application Developer Integration Edition, copy the following plug-in files to the wstools\eclipse\plugins\ folder of your WebSphere Studio installation folder:

- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.chaeditor.model.emf_5.1.0
- com.ibm.btt.tools.chaeditor_5.1.0
- com.ibm.btt.tools.fmteditor_5.1.0
- com.ibm.btt.tools.fmteditor.model.emf_5.1.0
- com.ibm.btt.tools.struts_5.1.0
- com.ibm.btt.tools.webdiagrameditor_5.1.0
- com.ibm.btt.tools.bp_5.1.0
- com.ibm.btt.tools.gw_5.1.0

- com.ibm.btt.tools.gw.model.emf_5.1.0
- com.ibm.btt.tools.migration_5.1.0

Setting up the migration tool

The migration tool is a WebSphere Studio Application Developer plug-in that helps you migrate your version 4.3 toolkit applications to version 5.1 applications.

If you installed the toolkit before installing the WebSphere Studio Application Developer Integration Edition, copy the following plug-in files to the wstools\eclipse\plugins\ folder of your WebSphere Studio installation folder:

- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.chaeditor.model.emf_5.1.0
- com.ibm.btt.tools.chaeditor_5.1.0
- com.ibm.btt.tools.fmteditor_5.1.0
- com.ibm.btt.tools.fmteditor.model.emf_5.1.0
- com.ibm.btt.tools.struts_5.1.0
- com.ibm.btt.tools.webdiagrameditor_5.1.0
- com.ibm.btt.tools.bp_5.1.0
- com.ibm.btt.tools.gw_5.1.0
- com.ibm.btt.tools.gw.model.emf_5.1.0
- com.ibm.btt.tools.migration_5.1.0

2.5.4 Testing

Testing is an essential part of the migration process and cannot be overlooked. Subtle changes can be introduced in your application's behavior when you use new software versions. Testing is the only way to discover any possible problems. Normally, the migrated project already has an existing test scheme in place, and in most cases, this can be used with little or no modification to test the migrated application. Testing a migration effort can be executed generally in much the same way as for a new release of your application, even when the migration has resulted in significant changes to the application infrastructure or core behavior.

New software specifications often introduce clarifications of details covered in older versions and this can translate into subtle changes in application behavior. As part of a migration, a full regression test of your application code might be necessary to ensure that subtle changes have not introduced errors into your application. The specifications are often the best source of information about changes that have occurred. You should carefully study the specification changes when planning and executing migration testing.

2.5.5 Runtime environment

The runtime environment of toolkit applications is based on WebSphere Application Server or WebSphere Business Integration Server Foundation.

Runtime environment package structure

Depending on the development environment you have on your workstation, the Branch Transformation Toolkit 5.1 installation wizard decides the runtime environment to be installed. For the runtime environment, there are two sets of runtime data and different files that are used. These are:

- Runtime data for WebSphere Application Server 5.1.1

  This runtime data does not include components that have dependencies on the features provided by WebSphere Business Integration Server Foundation.

  Whether you have WebSphere Studio Application Developer Integration Edition installed on your workstation or you have neither edition of the application developer installed, the Branch Transformation Toolkit installation wizard prompts you about whether to install the runtime environment for WebSphere Business Integration Server Foundation or the runtime environment for WebSphere Application Server.

- Runtime data for WebSphere Business Integration Server Foundation 5.1.1

  This runtime includes components that have dependencies on features provided by WebSphere Business Integration Server Foundation, such as business processes, startup beans, work areas, and activity sessions.

  If you have WebSphere Studio Application Developer installed on your workstation, the Branch Transformation Toolkit installation wizard automatically takes WebSphere Application Server as your runtime environment and installs the matching runtime data for you.

Installing toolkit applications on a runtime platform

Installing Branch Transformation Toolkit applications on a runtime platform consists of deploying EAR files on to WebSphere Application Server. Develop a solution in WebSphere Studio and package the code and resources for the solution in EAR files. Do the packaging by using WebSphere Studio or the Application Assembly Tool in WebSphere Application Server.

The Branch Transformation Toolkit 5.1 provides four sample applications that you can deploy immediately on WebSphere Application Server after you make a few customizations to adapt the sample applications to your particular requirements.

2.5.6 Deployment processes

Branch Transformation Toolkit 5.1 supports both WebSphere Application Server and WebSphere Business Integration Server Foundation. After testing an application, you can deploy the application on any of the server platforms you have available. For example, to deploy on WebSphere Application Server, the following procedure describes the basic steps involved in installing your application. This procedure holds good regardless of whether WebSphere Application Server is running on a Windows, Linux, or UNIX® platform.

1. Copy external files.

   Extract the .war file from your .ear file for the application, and then create a /dse directory in the right location. Copy all the needed files to the /dse directory.

2. Create the database and tables.

   Create a database and the tables within it by using DB2 commands.

3. Launch the WebSphere Administrative Console.

   Use this URL for the Web-based WebSphere Administrative Console:

   ```
   http://serverName:9090/admin
   ```

4. Import the .ear file for the application into WebSphere Application Server.
5. Install the new application and then set the database configuration.
6. Set up JDBC Providers.
7. Create new data sources with JDBC Providers.
8. Run the applications.

Try to run the application of runtime environment configuration as we did during testing.

3

Planning a Branch Transformation Toolkit migration

This chapter describes the methodology used for Branch Transformation Toolkit and discusses migration activities. Details about Branch Transformation Toolkit programming model and topology are also discussed, besides custom extensions and limitations encountered during a migration process.

This chapter discusses the following topics:

3.1 Version 5.1 End-to-end programming model

From the migration point of view, Branch Transformation Toolkit 5.1 is a set of advanced components designed to provide a technology bridge between Branch Transformation Toolkit 4.3 applications and pure WebSphere J2EE applications. The majority of these components allow Branch Transformation Toolkit 4.3 source and designs to be reused in this advanced environment with minimal recoding.

The Graphical Builder provides a set of tools to define the entities required by the applications and to distribute the runtime files. It provides a development environment that can be used through the entire cycle of developing toolkit applications. It also acts as a portal from where you can start other tools the toolkit provides.

The Graphical Builder offers an end-to-end programming model that provides the following benefits:

- Less reliance on high-level programming skills

  Using the Graphical Builder decreases the complexity threshold involved in developing toolkit-based applications. It provides components that are easy to use in areas ranging from backend connector development to user interface building blocks. This increases the size of the developer pool and reduces training costs.

- Enhanced development using a graphical user interface (GUI)

  To increase productivity, Branch Transformation Toolkit 5.1 provides WebSphere Studio Application Developer Integration Edition with an enhanced GUI. With the help of the Process Editor in WebSphere Studio Application Developer Integration Edition, developers can visually choreograph business processes for various applications. They do not have to spend time working with different interfaces and low-level APIs. Drag-and-drop tools allow them to define the sequence and flow of information between different business logic activities. Individual business logic activities and even entire workflows become building blocks that can be reused in developing other applications. Further gains in productivity are possible because runtime support for these new J2EE workflow capabilities is fully integrated in the application server to deliver a single administration and deployment environment.

3.1.1 Branch Transformation Toolkit components

This section describes the components that make up the end-to-end architecture of Branch Transformation Toolkit 5.1.

Branch Transformation Toolkit client

The Branch Transformation Toolkit 5.1 supports two client types. While one approach uses Java clients, including Java application or Java applets hosted in Web pages, the other approach uses HTML clients.

- Java clients

  The Branch Transformation Toolkit 5.1 does *not* provide new Java client components. It supports the Branch Transformation Toolkit 4.3 Java client, using modified servlets and request handlers in the server side. The server side modules translate requests from the Java client into a form that is manageable within the Branch Transformation Toolkit 5.1 server environment.

  The main focus of this support is to ensure that Java clients have a workalike server environment that can be accessed by existing Java clients without code changes on the client side. The Branch Transformation Toolkit server supports the following features:

  – A Branch Transformation Toolkit 4.3 compatible Session subsystem

    For more details, refer to “Session management” on page 65.

  – A Branch Transformation Toolkit 4.3-compatible Event subsystem

    For more details, refer to “Events” on page 70.

  – Servlets for session establishment, operation execution, event registration, and event processing

  For the Java client, no client code changes are necessary. However, configuration changes might be needed to change server operation names to process invoker names. Changes are made to the client/server channels, request and response handlers, as well as to any classes they interact with, such as the Branch Transformation Toolkit Context.

- HTML client

  The Branch Transformation Toolkit HTML client has been redesigned as a Struts/JSP framework. The Branch Transformation Toolkit 4.3 processor (Automaton) is replaced with Struts framework that provides similar functionality. Branch Transformation Toolkit HTML clients based on Automation must be migrated to the new Struts environment.

Presentation layer components

The presentation architecture comprises several components, which are shown in Figure 3-1 and Figure 3-2 on page 61. This section describes each component and how it fits in the overall architecture.

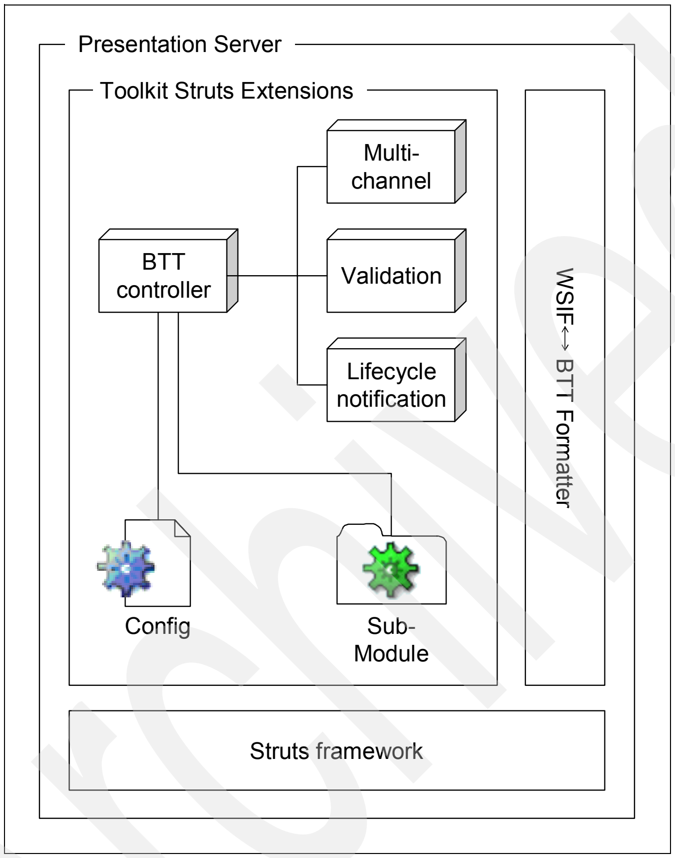


Figure 3-1 Presentation server

- BTT Controller

  This is the main driver of the architecture because it is the entry point to the presentation framework. The BTT Controller provides all the functionalities needed to handle incoming request data and outgoing response data.

- Base Action

  Every action that requires access to the Common Hierarchical Area (CHA), or uses the application flow processor, or any toolkit-provided service, must extend a Branch Transformation Toolkit base action. This action provides a hook into the Struts Action life cycle that allows developers to supply the business logic and hide the framework's complexity.

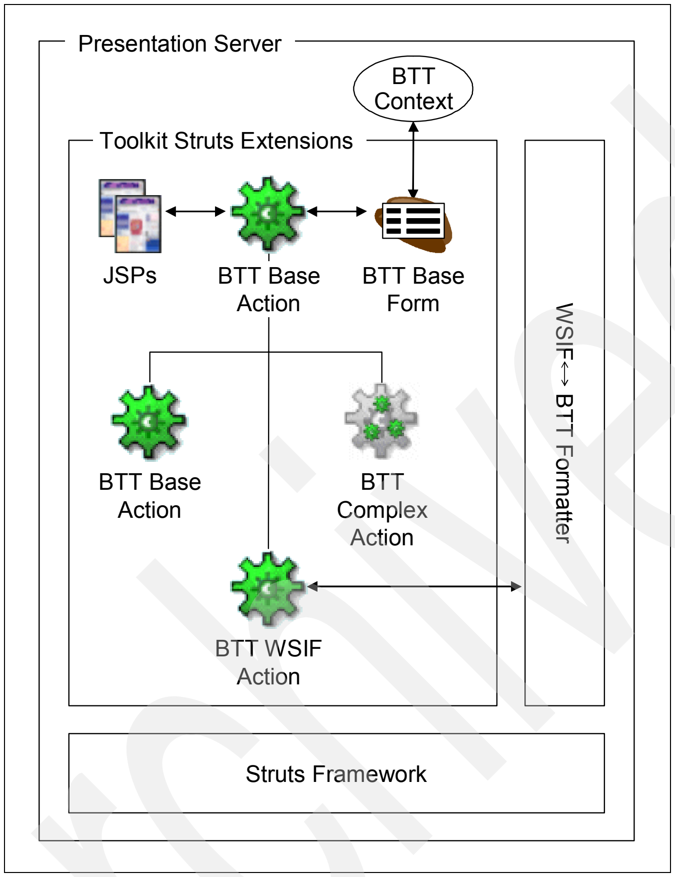


Figure 3-2 Presentation server continued

- Complex Action

  In Branch Transformation Toolkit, an action can comprise many closely related tasks. In Struts, an action can embody a number of tasks. These tasks are implemented as methods within a concrete action, for example, *BTTComplexAction*, that extends *BTTBaseDispatchAction*. The method signatures are similar to the `execute()` method in regular actions.

- Base Form Bean

  One of the actors providing the Branch Transformation Toolkit base action with access to the CHA, is the Branch Transformation Toolkit base form bean. This form bean provides a facade that communicates with the underlying CHA implementation, regardless of the technology used.

Alternatively, framework users can provide access to their own data models, leveraging on their existing designs and implementation, by using technologies such as WebSphere Data Objects (WDO).

- WSIF Access Action

  The Web Services Invocation Framework (WSIF) access action provides the interface necessary for the Struts-based presentation layer to talk to the backend business process service architecture.

- WSIF-to-Context Formatter

  One important component in achieving seamless invocation of Web services through WSIF is the ability to map data from the context to a WSIF message and back.

  As part of initiating a WSIF call to a Web service, the WSIF Access Action, as part of its processing logic, delegates mapping data to and from the context to the WSIF-to-Context formatter. It does so twice, first when it tries to map data from the context to the WSIF outgoing message just before the call to operation on the service, and secondly, upon the return from the call to the service, in which case the response WSIF message is mapped back to the context.

- Configuration

  The Struts framework provides a way to define application artifacts externally, namely, in a file referred to as struts-config.xml. This file is loaded when the framework is initialized and its in-memory representation is available for read-only access.

  In order to externally configure support for extensions to the Struts framework, the struts-config.xml file's grammar is extended to include CHA configurations, WSIF service access parameters, formatter definitions, condition flow definition, and so on.

  At runtime, the BTTActionServlet reads the extended configuration file and makes it available in-memory for the different components of the application to consume.

- JSP Context Access & Tag Libraries

  Not explicitly shown in Figure 3-1 on page 60 and in Figure 3-2 on page 61 is the need to provide some basic constructs that help developers to author toolkit-specific JSPs. This part of the architecture comes from the inherently heavy use of the presentation context within JSPs. In Branch Transformation Toolkit 4.3, the JspContextService gives a user full access to the context through a proxy. This pattern of use necessitates the preservation of the Branch Transformation Toolkit 4.3 JspContextService to access the context from within JSPs.

Additional constructs in the form of tag libraries are also needed to ease JSP development efforts. These tag libraries will, by extending Struts tag libraries, share the functionalities of the Struts-provided tag libraries and add the awareness of the context to those constructs. Tag libraries are also channel-aware and render the output for the appropriate channel.

- Multi-channel capabilities

  The Apache Struts framework does not provide any multi-channel capabilities. However, Branch Transformation Toolkit provides the ability to serve up heterogeneous clients through the same controller code. This multi-channel capability is HTTP-centric and is achieved by hooking in to the BTT Controller code to add the necessary functionality. The BTT Controller keeps a list of registered channel handlers and forwards requests to their appropriate handlers, which in turn render the contents to the clients in their supported markup.

  The main entry into Struts-based application (ActionServlet) keeps track of one RequestProcessor. This means that although several implementations of the request processor can be provided, only one can be associated with an ActionServlet. This leaves you with two options:

  – Provide an ActionServlet per channel supported

    The disadvantage of this approach is that it provides limited functions.

  – Provide a multi-channel aware request processor with a generic channel registry

    This approach is more flexible.

  As the WebSphere multi-channel strategy changes over time, the Branch Transformation Toolkit aligns with the strategies provided by the platform, whether it is WebSphere Application Server, WebSphere Portal Server, or a Rich Client Platform.

- Application flow

  The flow of the application, including pages to serve and actions to execute, should be defined externally to the application. The Branch Transformation Toolkit extends struts-config.xml to specify conditions and branches according to condition evaluation.

  It is important to remember that the condition processor is stateless, and thus any state information required should be kept in the CHA, and accessed through the context.

- Sub-modules/Applications

  An application can be further subdivided into granular submodules. The Struts framework provides the ability to define submodules. Each module has its own configuration, struts-config.xml.

The different struts-config files are specified as parameters to the Struts ActionServlet or its subclass in the web.xml file. Moving from one subapplication to another is a matter of prefixing the request action Uniform Resource Identifier (URI) with the subapplication context.

Subapplications share the same ActionServlet instance. When the framework encounters a context-relative request action URI, it searches for the subapplication that matches the request prefix. When found, the application configuration representing the subapplication struts-config is loaded into the request.

To preserve session management, subapplications are only handled within the context of one EAR file. The session manager is required to manage data sharing and access.

- Validation

  The validation component provides syntactic (on the JSP) as well semantic validation, including field, form, and cross-validation to application data. The validation can either be carried on the client side or the server side.

  Client side validation will be in the form of JavaScript and will be application-specific. On the server side, the framework makes use of the Apache Common Validation framework to define the validator and externally configure validation rules and parameters.

- Life cycle notification

  The Struts plug-in interface offers a standard way of getting notification from the framework on server start/stop events. Struts plug-in APIs provide a notification service for events that map to the servlet container's init() and destroy() calls on the corresponding ActionServlet instance or its subclasses.

  The Struts plug-in mechanism is used to notify Branch Transformation Toolkit-specific services of the life cycle of the BTTActionServlet instance in an effort to eliminate the need to provide different startup or shutdown servlets for the application.

- Abnormal application navigation

  Using a thin client, browser-based architecture introduces some risks of uncontrollable user behavior such as double and multiple-clicks, and problems using backward button or forward button navigation. Abnormal application navigation should be handled in a way that is consistent with the application behavior. Double-click should not result in re-execution of transaction.

  The BTT base action manages double-clicks and back navigation or forward navigation through the use of a token stored in the session and sent back with every request (as hidden field).

- Exception handling

  The Apache Struts framework comes with an easy-to-use declarative exception handling component, which allows for externally defined error messages as well as exception handlers.

Session management

Sessions in Branch Transformation Toolkit V5.1 are handled in much the same way as in Branch Transformation Toolkit V4.3. The major differences are the necessary changes (Can this be changed to “changes made”?) to remove session management physically from the BTT Context, and tie in the WebSphere ActivitySession.

The Java API for applications does not change. However, the base classes used for session management components such as SessionTable and SessionEntry are simplified to use standard Java collections. This reduces runtime memory overheads and enhances performance.

Business logic layer components

In Branch Transformation Toolkit 4.3, there are three ways of designing business logic:

- Using self-contained ServerOperation components that contain all process logic internally.
- Using composite ServerOperation components that will execute a procedural series of externalized Flow Steps.
- Using the Branch Transformation Toolkit Automation state machine to execute a series of actions based on the result of the last action performed.

This section discusses the migration considerations of the following business logic designs:

- Self-contained and composite server operation processes

  Starting from Branch Transformation Toolkit V5.0, server operation architecture has been upgraded using the latest WebSphere Application Server technology. There are two alternatives available in Branch Transformation Toolkit 5.0 for Server Operation, that is, Flow Definition Markup Language (FDML) process and Single Action Enterprise JavaBeans™ (EJB). Branch Transformation Toolkit 5.1 goes further in the same direction. It is based on the latest WebSphere Business Integration Server Foundation. Instead of leveraging on FDML, Branch Transformation Toolkit 5.1 uses Business Process Execution Language (BPEL), which is an industry standard generally accepted by the market.

Branch Transformation Toolkit 5.1 business process architecture depends on a combination of BPEL, tooling, code generation, and framework abstraction. After a user completes a process layout with BPEL, a plug-in tool generates code and adds modifications to the BPEL process file. The generated code and the modifications access user-defined logic through the abstracted framework logic. The major areas of interest in Branch Transformation Toolkit 5.1 business process architecture include:

– Custom Property

  In Branch Transformation Toolkit 5.0, Custom Attribute is an important feature in FDML that processes used. In Branch Transformation Toolkit 5.1, BPEL is used, and the functions previously covered by Custom Attribute are now handled by Custom Property.

  Beside CHA type for the process, Custom Property also keeps system data for the process, for example, indicators for process initialization and termination, and properties to store snippet execution results.

  Tooling is provided in Branch Transformation Toolkit to add Custom Property support to a BPEL file.

– State Observer

  Branch Transformation Toolkit 5.1 does not use State Observer. Initialization and termination logic is implemented in helper classes and added to Java snippet code as part of the code generation step.

– Helper Class and Access Class

  A lot of system logic is now implemented in helper classes, for example, launching a snippet, initializing, and terminating a process. An *access class* is a class generated for a process. It is responsible for providing synchronized access to the process initialization and termination logic. Corresponding to an access class are Custom Properties that indicate if the process has already been initialized or terminated.

  Java code that accesses the helper classes and the access class is added to the BPEL file through code generation. By doing so, the Branch Transformation Toolkit minimizes the lines of code that are required to add to the BPEL file.

– Process Initialization and Termination

  As mentioned above, process initialization and termination logic are implemented in helper classes. Java code that accesses the helper classes will be added to the BPEL file as part of code generation.

  Java code is added to the first navigation link for process initialization and the last snippet for process termination. As a result, it is necessary to locate the first activity and the last activity in a BPEL process. If a BPEL process is ended with a Web service, rather than with a snippet, it is necessary to add an extra snippet for process termination at the end of the process.

In some special situations, a BPEL process can start and end with more than one link or snippet. This is the reason why an access class is required. An access class ensures synchronized access to the process initialization and termination logic. If a BPEL process is defined so that it ends with different snippets depending on the process navigation, it is necessary to put the same navigation condition on the termination helper class access logic. In this case, it prevents premature termination of the process. The addition of the navigation condition is handled by the Branch Transformation Toolkit.

– Common Hierarchical Area (CHA)

  BPEL supports the concept of message variable. A variable holds a message that contains various parts.

  In Branch Transformation Toolkit 5.1, a CHA variable is defined with each process. All the processes share the same message definition and the message contains only one part, that is, CHA instance. Access to the CHA instance is through a Branch Transformation Toolkit abstract layer, where the logic related to the message variable is hidden from general developers. Addition of the CHA variable is handled by tooling.

  CHA instances stored in a message variable are accessible by the process and by local Java snippets defined along with the process. If a remote Web service is defined as part of a process, the work area will store the CHA instance as well. When WebSphere Business Integration Server Foundation is not available in the remote end, transfer of CHA instance ID to the remote service through the service interface, is required.

– Java snippets

  Branch Transformation Toolkit 5.1 provides helper classes to launch an operation step from a Java snippet defined in a BPEL process. The implementation of a snippet, which includes the helper class access code, is added to a BPEL process by the Branch Transformation Toolkit.

  Operation steps that can be launched by the helper class extend the Branch Transformation Toolkit abstract layer. The Branch Transformation Toolkit abstract layer provides APIs for CHA and service and formatter access, besides providing backward compatibility to the existing customer code.

  Java snippets use the abstract layer to access the CHA instance that is stored in the message variable.

– Web service

  As an alternative to using local Java snippets, developers can also use a remote Web service as an activity in their BPEL process.

  When a remote Web service is used, this remote service will not have access to the CHA instance stored in the local message variable. If the Web service still requires access to the CHA instance, it is required to either store the CHA instance in a work area and extend the Web service from the Branch Transformation Toolkit abstract layer, or if WebSphere Business Integration Server Foundation is not available at the remote end, to send the CHA instance ID to the Web service through its service interface.

  Detection of remote Web service definition in a process is handled by the code generation tooling. When a remote Web service is detected within a process definition, the tooling should generate helper class access code that reflects the issue.

– Navigation condition

  In Branch Transformation Toolkit 4.3, operation steps may return an integer that determines the navigation path of a process. This feature is preserved in Branch Transformation Toolkit 5.1.

  To implement this, the Branch Transformation Toolkit code generation should detect all possible paths from a given activity and then add navigation conditions to all the links.

  The result of each operation step is stored in Custom Property. These results will be evaluated as navigation conditions and used to determine the flow of the process. It is important that each operation step has its own corresponding custom property. Otherwise, the results might overwrite each other if a parallel flow is defined within a process.

– Tooling

  Tooling plays an important role in the Branch Transformation Toolkit 5.1 business process architecture. It is implemented as a WebSphere Studio Application Developer Integration Edition plug-in and available to the developers after they build the skeleton of a process. This tool is responsible for adding modifications to a BPEL process and generating an access class for the process.

  Some of the modifications that are added to a BPEL process include:

  - Import class statements
  - CHAmessage variable
  - Snippet implementation
  - Process initialization and process termination logic
  - Extra Java snippet if a process ends with a Web service

        - Navigation logic
        - Detection of Web service usage within a process
        - Custom properties
    - Modifying a process after code generation

      If a process is modified after the completion of code generation, it is necessary to run the code generation tool again.

- Automation-based business processes

  Automation-based business processes are replaced by *Invokers* calling BPEL business processes.

Runtime data management

The Branch Transformation Toolkit 4.3 used a shared data collection scheme called *Context,* for handling runtime data. However, this scheme was not scalable outside of a single Java Virtual Machine (JVM). To comply with the scalability requirements of the J2EE architecture, it is necessary to provide a distributed context. In Branch Transformation Toolkit 5.1, you can call CHA to implement the facility.

For a Branch Transformation Toolkit solution, there are three logical context types. These are:

- Local dynamic

  These contexts do not have names and are transient in nature. Contexts of this type are created locally and do not interact with a back-end CHA service. They operate strictly within the local JVM and should not be passed between processes for serialization performance reasons. Their usage includes provision of temporary runtime data storage within a business process.

- Local predefined

  These are local contexts that are named and predefined. They are created by the backend CHA service on behalf of the caller and returned as a fully self-contained context, complete with predefined internal structure. Once received, the context instance is truly a local context with no further interaction with the backend CHA. All data management and storage is within the local context. These can be used for interaction with services or formatters within the same business process.

- Fully distributed

  Distributed contexts are made of two components: a client facade and a backend CHA service. For this type of context, the context class is nothing more than a thin shell over J2EE-compatible interfaces. Contexts of this type delegate all method invocations to a remote CHA entity EJB for processing.

From the application developer's point of view, there is very little difference between these types of contexts once they are created. In fact, the local context types provide 95% of the functionality found in the Branch Transformation Toolkit 4.3 context class.

The distributed context, however, has some restrictions. Removed from this type of context are session management logic and event notifier storage. The APIs remain, but the logic is replaced by delegation to external classes. Internally, the context deals solely with data storage tasks.

Message formatting services

Another important component of the Branch Transformation Toolkit architecture is the facility used to transfer data from message buffers to and from the Branch Transformation Toolkit context. The version 5.1 toolkit extends the version 4.3 formatters and becomes an external, shared service. Because formatting services do not change during an application run, it is possible to move this entire facility out of the client and share these definitions and runtime processing across all client applications within the enterprise. This reduces duplicated definitions and logic across the enterprise network and reduces the client memory and processing requirements.

Services

Branch Transformation Toolkit services are shared executable components stored in the context and retrieved by the application as needed.

Due to the highly distributable nature of Branch Transformation Toolkit 5.1, it is not possible to share executable components in the CHA. Instead, configuration information for a service is stored in the context and is applied to an external Web Services Invocation Framework (WSIF) component that encapsulates the service functionality.

Events

The Branch Transformation Toolkit V5.1 supports a client/server event framework for client/server event processing for compatibility with Java clients. Changes are made to the Event Manager and to the context to support client/server events. All the servlets associated with event processing are preserved for client compatibility.

An event bridge is included to translate and route Branch Transformation Toolkit events to and from JMS events.

3.1.2 Tools

The development environment of Branch Transformation Toolkit applications is based on the development workbench plugin for WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition. The development workbench allows nontechnical users to define and store new business process flows and their associated data structures in a central multi-user repository called the Development Workbench Repository. Business process flows and framework entity definitions are managed with wizards designed to simplify this task.

In the production environment, the Branch Transformation Toolkit 5.1 framework uses these definitions as its configuration parameters to build the application dynamically. This approach to application development is a key benefit of the framework, minimizing the need for raw code development and promoting code reuse.

The Branch Transformation Toolkit Development Workbench, which is a plugin for WebSphere Studio, is a tool used to facilitate the creation of the definitions needed for the product runtime. This tool helps the corresponding wizards to allow and guide the entry of definitions and a repository to persist them.

The Branch Transformation Toolkit 5.1 Development Workbench includes the following tools that help developers build an end-to-end solution application:

- CHA Editor and Format Editor

  The CHA Editor provides a graphical and easier way to work with CHA contexts and their data elements and types. As the number of definitions in the files increases and the CHA structure increases in size and complexity, maintaining a mental picture of the entire CHA structure becomes increasingly difficult. The CHA Editor provides a visual representation of the structure and lessens the need to deal directly with XML tags. This allows developers to concentrate on business requirements and issues.

  The Format Editor provides a graphical and easy way to work with the definition file for formatters. It also provides a visual representation of the structure and lessens the need for developers to deal directly with XML tags. It provides common editing features such as cut, copy, paste, undo and redo, load and save, delete, drag-and-drop, reorder format elements, and sorting.

- Struts tool extensions

  The Struts tools extensions provide a graphical and easy way to work with extended Struts configuration files. Since the toolkit's Struts extensions component provides customization to the Apache Struts framework, some settings in the extended Struts configuration files are hidden from the standard Struts configuration file editor provided by WebSphere Studio Application Developer.

The Struts tools extension has a friendly user interface that saves you from directly editing the XML source of Struts configuration files.

- Business process wizard

  The Business process wizard provides a GUI to help developers extend business processes that take advantage of the Branch Transformation Toolkit abstract layer. It helps to customize the business process code of the BPEL files in the following ways:

  – Specifying the CHA context associated with the business process.
  – Specifying the process type, that is, general process, log on process, or log off process.
  – Specifying the mapping relationship between CHA contexts and process results.
  – Specifying external snippet classes for the business process.
  – Enabling conditional navigation based on snippet results.
  – Adding the variables in the process to the BPEL file and adding the associated message definition of the variables to the WSDL file.

- Business operations migration tool

  Branch Transformation Toolkit 5.1 provides a set of tools to migrate applications developed with version 4.3 to the new Branch Transformation Toolkit version. The migration tools provides GUI to help developers migrate the server operations, flow processes, and screen flows of version 4.3 applications to the corresponding components of the version 5.1 applications.

  In Branch Transformation Toolkit version 5.1, server operations are replaced by single action EJBs or by business processes that are called by invokers. You can migrate nonstep server operations to single action EJBs or business processes and generate the associated invokers by using the migration tools.

  You can migrate only stepped operations to business processes and generate the associated invokers. The operation steps are migrated to Java snippets for business processes. In general, links between the Java snippets are automatically based on the server operation definition files.

- JSP screen flow migration tool

  In Branch Transformation Toolkit version 5.1, screen flow processors are based on the Apache Struts Framework. You can use the migration tool to migrate version 4.3 screen flows to the corresponding constructs in version 5.1. After the migration, you can edit the screen flow either by editing the

Struts configuration file in an XML editor or by editing the .gph file in a graphical screen flow editor.

- BTT graphical workbench

  You can use all the tools described here to facilitate the development of applications based on the Branch Transformation Toolkit. The graphical builder integrates all these tools. It provides visualization tools, integrated development, and seamless deployment techniques that apply to the full life cycle of your application development.

  Figure 3-3 shows the tools provided by Branch Transformation Toolkit 5.1 and the relationship amongst them.

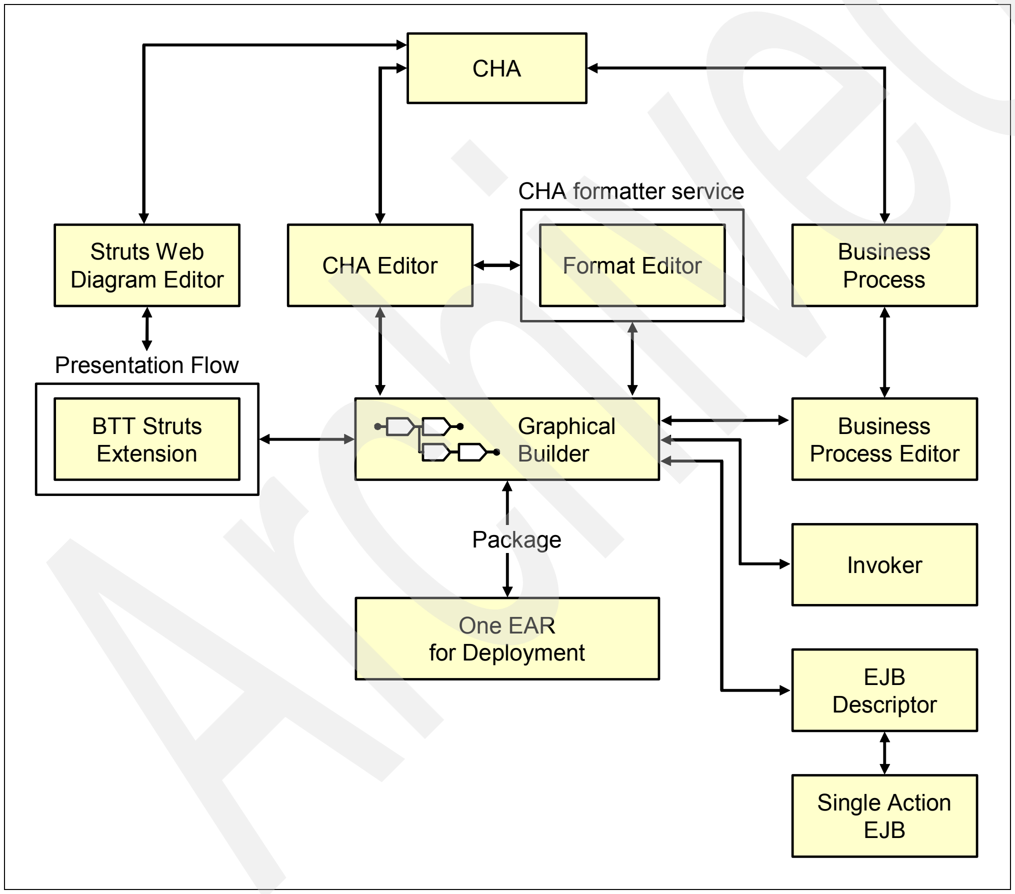


Figure 3-3 Branch Transformation Toolkit tools

3.1.3 Other features

The Branch Transformation Toolkit 5.1 adds the following features to an end-to-end solution:

- LUO and LU6.2 Java Connector Architecture (JCA) connectors. These connectors are:
  - SNA JCA LU0

    The SNA JCA LU0 Connector enables a Java application to send requests to and get responses from an existing Enterprise Information System (EIS). It enables developers to deploy these applications into a managed environment in which a J2EE-capable application server such as WebSphere Application Server handles connection pooling, transactions, and security. To support this, the SNA JCA LU0 Connector implements the JCA.
  - SNA JCA LU62

    SNA JCA LU62 Connector is a resource adapter that enables an application to send requests to and get responses from an existing EIS. It enables developers to deploy these applications into a managed environment in which a J2EE-capable application server such as WebSphere Application Server handles connection pooling, transactions, and security.

    To deliver the messages, the SNA JCA LU62 Connector supports the WSIF interface through the JCA plug-in and JCA through the Common Client Interface (CCI). CCI supports only local connections, which means that the application must be on the same machine as the SNA JCA LU62 Connector. The WSIF interface supports local and remote connections. In a remote connection, the application and SNA JCA LU62 Connector can be on different machines. In this case, you can use the SNA JCA LU62 Connector to generate a SOAP proxy class and a set of WSDL files. The application uses these files to access the remote SNA JCA LU62 Connector.
- Java light weight client sample application

  Branch Transformation Toolkit 5.1 provides a sample application that implements a Java client. This Java client sample is designed and implemented to show the most important steps in a Branch Transformation application development project.

  The Java client sample application follows a four-tier architecture:
  - The client tier, consisting of Java clients

  - The application presentation tier, providing a bridge between the client and the application logic tier, and managing the views of the Java client sample application
  - The application logic tier, conducting business transactions.
  - The back-end system tier, storing the business data.

- HTML sample application

  This sample demonstrates the important steps in developing an online home banking application using the features of the Branch Transformation Toolkit that support HTML clients. This sample can be used as a guideline for solution providers who wish to gain a better understanding of the toolkit facilities.

  The target environment for the HTML sample is a four-tier architecture: client, application presentation layer, application logic layer, and backend system layer. In this case, the client is the Web browser used to access the home banking application. This sample does not show the details of communication between the server and the host because this is not its main purpose.

  The application comprises the following:

  - Client user interface components (JSP files)
  - Business navigation processes (toolkit Struts extension externalized XML files)
  - Invoker definition (resource bundle file)
  - Business logic components (Java code in Single Action EJB and business process component)

3.2 Application packaging and topology

As part of building a Branch Transformation Toolkit solution, you should consider application packaging and managing runtime topologies.

3.2.1 Application packaging

An important deployment issue is to determine a policy for packaging the application code and resources. This policy determines how to distribute the code and resources on different servers. A solution based on Branch Transformation Toolkit may use Java Archive (JAR) files that provide physical packaging for a set of files. The JAR files have the following advantages:

- Reduced interactions with the server during the download process.
- Improved transmission performance because of object compression.
- Optimized memory usage in the browser cache.

When using JAR files for application packaging, consider the sizing and number of packages to achieve optimal network performance.

The migration tool assists with application packaging. The migration tool can generate an application deployment file, an EAR file. The EAR file is the outcome of the migrated components. The migration tool user can select the components needed to be put into the EAR file, and then generate the EAR. Currently, only one EAR file can be generated at a time.

Applications are packaged into EAR files, allowing applications to be deployed on a server as coherent, self-contained units. A server can have multiple EAR files deployed. These EAR files will not interoperate with each other except through established communication protocols such as JMS, HTTP, and IIOP. While it is possible for the code within EAR files to interact with the file system using absolute file addressing, this is considered to be an unsafe programming practice that should be avoided and vigorously discouraged.

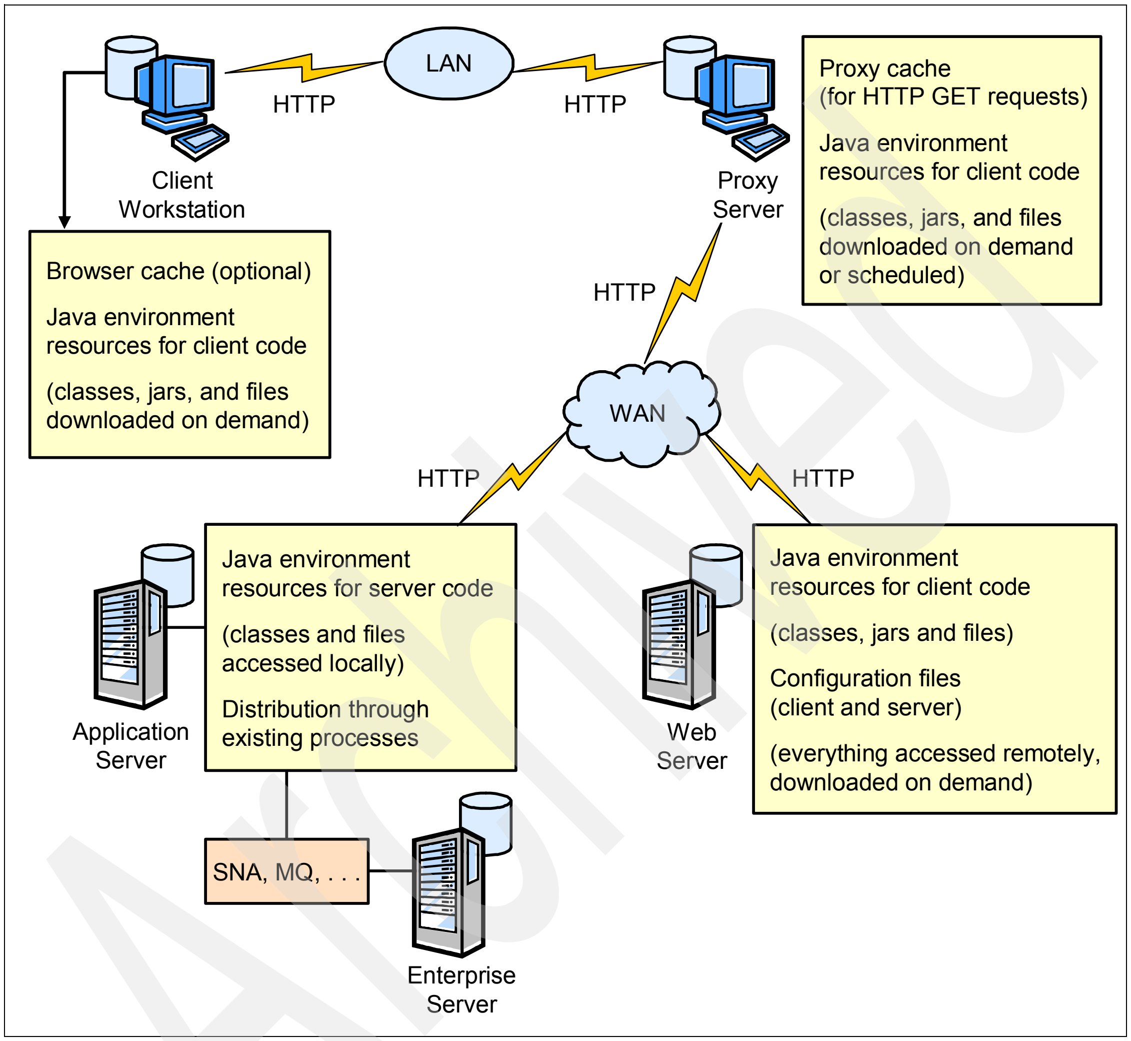
An EAR file can contain multiple Web Application (WAR) and EJB components. Within an EAR file, interaction between these components can be controlled using WebSphere Application Server security and access control mechanisms. WebSphere Application Server provides implementations of the Java Authentication and Access Service (JAAS), as well as extensions to the J2EE Servlet API. WebSphere Application Server also contains role-based resource access extensions to provide high levels of security to sensitive information. These security features are configured as a part of the deployment configuration when an application is deployed on a server. These features can also be enabled globally or disabled as a part of the server configuration.

3.2.2 Topology

The physical location of the Branch Transformation Toolkit components depends on the project environment and requirements. All the resources required for these components, such as definition files, configuration files, and icons, can be located either on a local client machine or on a remote server. Branch Transformation Toolkit 5.1 resources comprise the following:

- Java classes that are executed as Java applications or applets.
- Configuration files and definition files that specify the user settings of the Branch Transformation Toolkit 5.1 environment.

This way of managing resources allows for several different deployment configurations, for example, the application code can be executed in a Web browser or as an application. Figure 3-4 shows an example of a topology where Web browser-based clients are used.



LAN
HTTP
HTTP
Client Workstation
Proxy Server
Proxy cache (for HTTP GET requests)
Java environment resources for client code
(classes, jars, and files downloaded on demand or scheduled)
Browser cache (optional)
Java environment resources for client code
(classes, jars, and files downloaded on demand)
HTTP
WAN
HTTP
HTTP
Java environment resources for server code
(classes and files accessed locally)
Distribution through existing processes
Application Server
Web Server
Java environment resources for client code
(classes, jars and files)
Configuration files (client and server)
(everything accessed remotely, downloaded on demand)
SNA, MQ, . . .
Enterprise Server

Figure 3-4 Application topology with Web browser clients

The Web browser-based topology has four kinds of servers:

- The *enterprise* server contains the older applications.
- The *application* server contains the Java code to communicate with enterprise servers and clients
- The *proxy* server contains the cache.

- The *Web* server contains the application resources needed by the browser.

In a Java client topology, the proxy server is not required because Java clients do not have the same features as Web browsers.

3.3 Workload management decisions

While not strictly related to migration activities, certain implications of Workload Management (WLM) can affect migration planning. These are mostly related to test and deployment activities.

Workload management is the process of spreading multiple requests for work over the resources that can do the work. It optimizes the distribution of processing tasks in a WebSphere Application Server environment. It also improves performance, scalability, and availability of systems and applications, besides providing failover when servers are not available.

3.3.1 Benefits

Workload management is most effective when the deployment topology comprises application servers on multiple machines, since such a topology provides both failover and improved scalability. It can also be used to improve scalability in topologies where a system comprises multiple servers on a single, high-capacity machine. In either case, it enables the system to effectively use the available computing resources.

WLM provides the following benefits when constructing applications:

- It balances client requests, allowing incoming work requests to be distributed according to a configured WLM selection policy.
- It provides failover capability by redirecting client requests to a running server when one or more servers are unavailable. This improves the availability of applications and administrative services.
- It enables systems to be scaled up to serve a higher client load than that provided by basic configuration. With clusters and cluster members, additional instances of servers can easily be added to the configuration.
- It enables servers to be transparently maintained and upgraded while the applications remain available for users.
- It centralizes the administration of application servers and other objects.

Two types of requests can be workload-managed in WebSphere Application Server or WebSphere Business Integration Server Foundation:

- HTTP requests distributed across multiple Web containers.
- EJB requests distributed across multiple EJB containers.

3.3.2 Web server workload management

This is the mechanism for sharing the load by distributing HTTP requests among a group of Web servers. These servers make up a cluster, grouping together independent nodes that are interconnected and working together as a single system. The cluster appears as a single Web server to the Web client or browser.

This type of workload balancing is also called IP spraying. As shown in Figure 3-5, it is used to intercept HTTP requests that are then redirected to the appropriate machine in the cluster, providing scalability, load balancing, and failover.

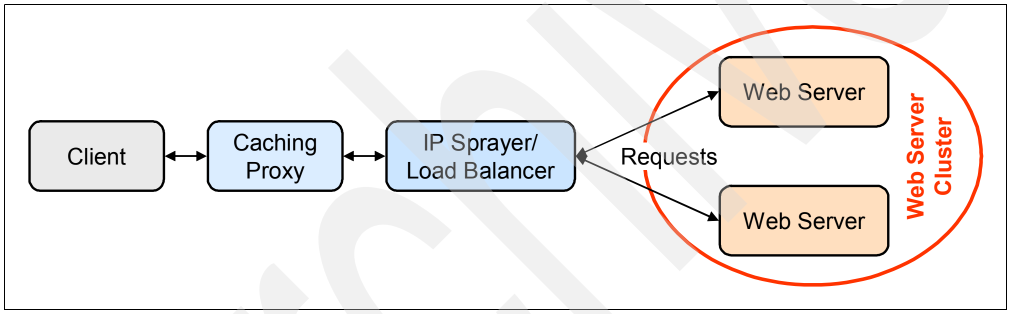


Figure 3-5 Web server workload management

3.4 Server cluster workload management

Clusters are sets of application servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine. A cell can have one cluster, multiple clusters, or no clusters at all.

A cluster is a logical grouping of the application servers, as shown in Figure 3-6. It is not necessarily associated with any node, and does not correspond to any real server process running on any node. A cluster contains only application servers and the weighted workload capacity associated with those servers.

To prevent workload imbalances in which one server is overburdened and the other servers have low or zero activity, the weighted definition allows nodes to have different hardware resources and still participate in a cluster. The higher the weight, the faster the request is served and vice versa.

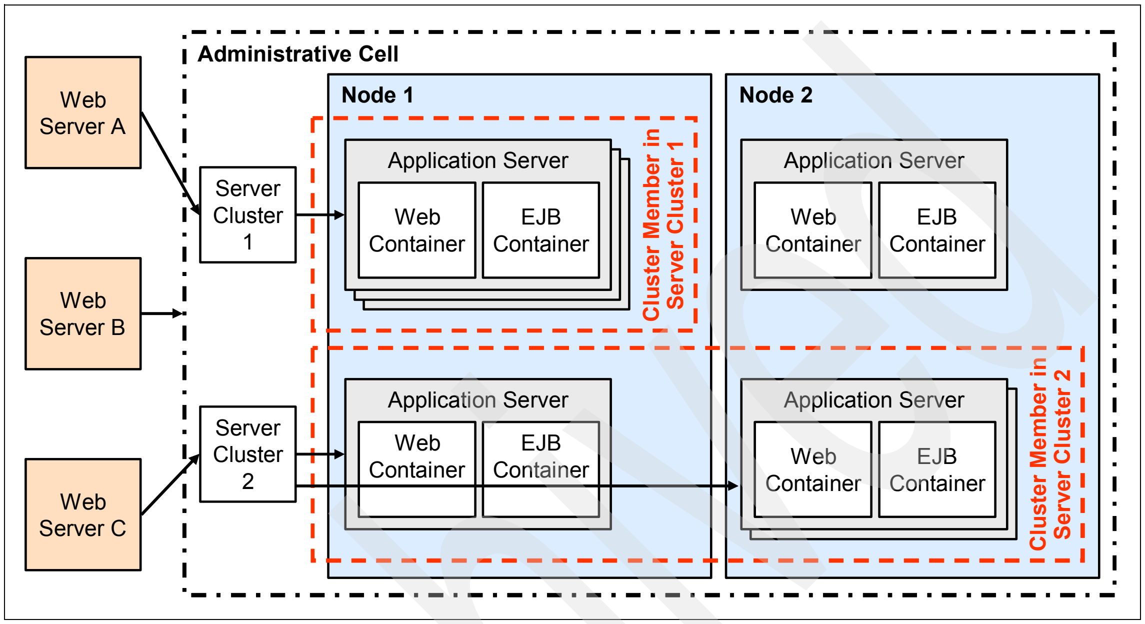


Figure 3-6 Application server cluster

3.4.1 Workload management considerations

A WebSphere Application Server *server cluster* is a grouping of application servers that can be managed together to participate in workload management. Application servers participating in a cluster can be on the same node or on different nodes. A deployment cell can either contain no clusters or have many clusters, depending on the need of the cell administration.

Servers that belong to a cluster are members of that cluster set and must all have identical application components deployed on them. Other than the applications configured to run on them, cluster members do not have to share any other configuration data. Following are some key guidelines to enable an application to support clustering.

Application

WebSphere Application Server can respond to increased use of an enterprise application by automatically replicating the application to additional cluster members as needed. This lets developers deploy an application on a cluster instead of on a single node, without considering workload. This means that the session and application must not be physically tied to any particular server's JVM.

- Web applications should as stateless as possible.

  If a state must be stored in the Web application, use the HTTP Session persistence feature to store state data between request invocations.
- Session EJB instances should be stateless.

  If a state is required, the stateful session bean should implement persistence logic at its transaction boundary, usually at method invocation.
- Application state data should be implemented using Entity EJB classes.

  Performance tuning is required to ensure that only critical data is persisted frequently.
- J2EE prohibits multiple threads in EJBs.

  Multiple threads should be avoided in the Web application as well, unless these background threads are totally stateless and self-maintaining. It is very difficult to coordinate background threads with session-level persistence.

Session management

In a clustered environment, the session management facility requires an affinity mechanism so that all requests for a particular session are directed to the same application server instance in the cluster. This requirement conforms to the Java Servlets 2.3 specification in that multiple requests for a session cannot coexist in multiple application servers. One such solution provided by IBM WebSphere Application Server is session affinity in a cluster, available as part of the WebSphere Application Server plug-ins for Web servers. It also provides for better performance because the sessions are cached in memory. In clustered environments other than WebSphere Application Server clusters, use an affinity mechanism, for example, WebSphere Edge Server affinity.

Load balancing

A clustered environment supports load balancing, where the workload is distributed among the application servers that comprise the cluster. In a cluster environment, the same Web application must exist on each of the servers that can access the session. Developers can accomplish this setup by installing an application on to a cluster definition. Each of the servers in the group can then access the Web application.

Fault tolerance

If one of the servers in the cluster fails, it is possible for the request to reroute to another server in the cluster. If distributed sessions support, that is, session persistence, is enabled, the new server can access session data from the database or another WebSphere Application Server instance. You can retrieve the session data only if a new server has access to an external location from which it can retrieve the session.

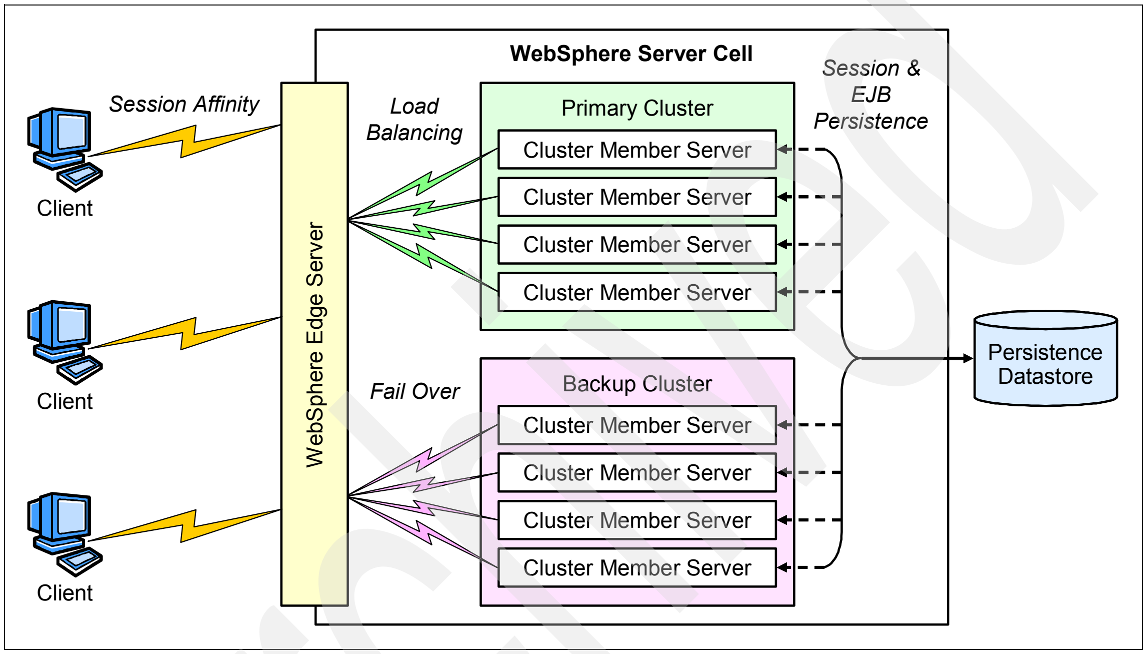


Figure 3-7 WebSphere Application Server cell

As shown in Figure 3-7, each cluster member server might be any of the WebSphere Application Server instance views. However, although cluster member servers are identical to each other, data is not visible between them unless it is specifically shared using database or entity EJB instances. In other words, if one cluster member creates static data, the data created will not automatically be available to the other cluster members.

3.5 Migration considerations for custom extensions

Custom extension is a capability that allows users to plug a custom process into the migration module. In general, this is used to extend the coverage of the migration tool to handle any extended functions in the existing Branch Transformation Toolkit application system.

Using the migration modules provided by Branch Transformation Toolkit as a base, you can make extension customizations to meet application requirements, and then plug them into the migration tool. The migration tool carries out the custom extension as the extra process right after the base migration process. It is also possible to replace completely the default process of the migration tool.

The user extension employs the migration component base. Therefore, a developer can replace and extend base migration functions flexibly. This is a key feature of the migration tool and provides the flexibility for customers to plug in their own extensions. In many cases, Branch Transformation Toolkit projects extend or replace Branch Transformation Toolkit functions and features in order to meet business requirements and extend the capability. Therefore, the migration tool must provide the capability to migrate users' custom extensions to Branch Transformation Toolkit 5.1.

3.5.1 Extensions in Branch Transformation Toolkit 4.3

Branch Transformation Toolkit customization is often needed to enrich and enhance the base functionality to meet application system requirements. The Branch Transformation Toolkit provides a high extensible framework, allowing the customer to extend its base functions.

Customers usually make extensions for Branch Transformation Toolkit 4.3 application systems by modifying the following components:

- Data element
- Type data element
- Format
- Service
- Operation
- Flow processor
- Screen flow processor

These customer extensions affect the following two areas:

- Branch Transformation Toolkit definition files
  - dse.ini

    The definition file has additional information for the new tags.
  - Component-related definition files

    These use new tags with the new attributes.

- Classes
  - Sub-class of the base class

    The customer extensions need to inherit from the base classes of Branch Transformation Toolkit components to add new features or change the existing functions.
  - Application

    The application has to cast the object type to the customer extension to access the new features and APIs. It can also use a sub-class of the customer extension to inherit key behavior. We usually recommend using this method for operation extensions.

3.5.2 Extension points in migration tools

The migration tool should provide extension points in the following areas:

- Definition files

  The included components are:
  - dse.ini file
  - Data model, including Context, Data Element, and Type Data Element
  - Formatter
  - Operation
  - Flow processor
  - Screen flow processor

  For Service components, customer extensions are not included in the migration tool, but your options are described in the migration documentation.
- Applications

  The migration tool provides extension point in code generation for the following components:
  - Operation

    All child classes of the custom operation class will be migrated to single action EJB (SAE) or activities in BPEL. In general, non-step operations will be migrated to a SAE and step operations will be migrated to BPEL. The migration tool provides the extension point as part of code generation.
  - Flow processor

    All flow processors will be migrated to the BPEL activities. The migration tool provides an extension point in the code generation step.
  - Screen flow processor

    Screen flow processor logic is located on both the client and server side of Branch Transformation Toolkit V4.3 solutions. Therefore, screen flow processors will be migrated to a combination of Struts extensions, SAEs and BPEL processes. The migration tool provides an extension point in the code generation step.

For both definition files and applications, the migration tool provides a configuration point to specify any extended classes, so that they can be plugged into the migration process. The extended classes are not for the Branch Transformation Toolkit runtime class, but for the custom migration process to generate the proper definitions or program code for Branch Transformation Toolkit runtime custom extensions.

When migration is needed for subclasses of the base Branch Transformation Toolkit components such as Context, Data Element, and Formatter, and for superclasses of custom operations such as Flow processor and Screen Flow processor, the migration tool does *not* provide support. It is the client's responsibility to modify this code manually as part of the migration project. Details of manual migration of tasks are discussed in the following chapters.

3.6 Limitations

Not every project built with Branch Transformation Toolkit 4.3 can be migrated smoothly to version 5.1. Due to the changes in the architecture of Branch Transformation Toolkit V5.1 and because of the introduction of new technology, several limitations that should be considered when carrying out a migration project still exist. Some of the migration limitations are due to the Branch Transformation Toolkit itself, while others are caused by the new features in the new version. This section provides an overview of some of the migration issues you might encounter.

3.6.1 Limitations of Branch Transformation Toolkit 5.1 function

The Branch Transformation Toolkit provides a number of tools that support the development of applications. All the tools are plug-ins to WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition.

Note: some functions of version 5.1 are only available when WebSphere Studio Application Developer Integration Edition is used.

This section lists the limitations and known issues for IBM Branch Transformation Toolkit for WebSphere Studio V5.1. It also provides information on any fixes or workarounds that exist for these limitations and issues.

Several limitations and issues are identified with the following components:

- Business Process Wizard

  This wizard provides a GUI to help developers extend business processes to take advantage of toolkit-specific entities. This tool is only available when WebSphere Studio Application Developer Integration Edition is used.

- Business Process Component

  This component enables applications to perform business processes using the Business Process Container in WebSphere Application Server Enterprise Edition. Applications can invoke the business process using a request handler feature of the multichannel architecture and an EJB interface, or by using a flow processor through the EJB or WSIF interface.

- CHA Editor and Format Editor

  – These components support only motif mode for WebSphere Studio Application Developer or WebSphere Studio Application Developer Integration Edition on Linux.

  – They only support their tool-specific XML syntax tools. If you load an XML file that contains self-defined tags or attributes not supported by the tools, the system will clean those tags and attributes from the XML file when a save is done.

  – Type view does not support synchronization with other views.

  – For typed data, the descriptor does not support adding parameters such as a subtag with IDs not defined in optional attributes.

  – For typed data, the KCollDescriptor and ICollDescriptor does not support adding a validator into the descriptor.

  – For typed data, the refType descriptor cannot add attributes to override attributes of referenced data.

  – When using WebSphere Studio Application Developer Integration Edition version 5.1.1 on Linux in motif mode, using the mouse scroll button to drag context nodes and format definitions is not supported.

  – When using the Format Editor with WebSphere Studio Application Developer, always close the Format Editor before closing the WebSphere Studio Application Developer. Otherwise, the next time WebSphere Studio Application Developer is started, the Format Editor and CHA Editor that are started automatically can have problems with view synchronization. To solve the synchronization problem, close the automatically started Format Editor and CHA Editor and start them again.

3.6.2 Limitations for migration

Because Branch Transformation Toolkit V5.1 uses many new technologies and is redesigned to take advantage of new software architectures, one of the challenges for a successful migration is to bridge the gap between two applications that use different toolkit versions. There are many ways of solving the problems that result from migration activities. Although the migration tool provided by Branch Transformation Toolkit 5.1 does most of the work, some modification and reconfiguration is needed to successfully migrate a Branch Transformation Toolkit application. For complicated migration changes, rewrite some of the application logic to comply with the new standards or technology.

In some cases, Branch Transformation Toolkit V5.1 can no longer accept some code or features from the existing applications and this can make it difficult to modify the application. To minimize these problems, application designers should design a total architecture that avoid such conflicts in advance.

Some issues that could occur during the migration process are listed here:

- Invokers

  When using JDK 1.4, create the EJB home object of the invokers explicitly. The invoker super class does not provide a common EJBHome creation method.

- Services

  If the service invoker is called in a non-J2EE environment, a SOAP invocation with a nested Hashtable message type works properly. However, in a J2EE environment, for example, application client, servlet, EJB, and so on, the nested Hashtable message type fails. This affects the services architecture, JDBC table services, and the electronic journal.

- Business Process Wizard

  Since the Business Process Wizard makes extensions from the EMF model of the Process Editor and the graphical display of the Process Editor is controlled by the internal mechanism of the EMF/GEF, the Business Process Wizard may have problems displaying the graphical layout of generated BPEL snippets and the content of BPEL variables. In such a situation, manually adjust the graphical layout to get a clearer picture and re-open the editor to refresh the content of BPEL variables.

- Migration tools

  – The migration tool does not support the new attributes of the customer extension.

  – To import definition files when using a URL instead of a local disk, all self-defined files have to be specified in the dse.ini file. It is impossible to query how many files are under a given URL.

– If you carry out screen flow migration multiple times, the wizard will show duplicate Struts configuration files in the list.

4

Preparing for migration

The overall migration process consists of several phases. Before the real migration, you should perform some amount of analysis and preparation. Depending on your migration objectives, you might have to customize the migration tools to meet the requirements of the application you want to migrate.

In this chapter, the following topics are discussed:

4.1 Analysis and preparation

During this phase, the project team should perform a complete analysis of the differences between Branch Transformation Toolkit version 4.3 and version 5.1, paying particular attention to the difference in the application logic layer. The project team can use the documentation of the existing application system to identify the parts that can be migrated automatically and the parts that cannot be migrated automatically. This will help make preparations for the migration.

When preparing for migration work, the project team should analyze the situation before starting the real migration. We recommend that you undertake the following steps:

- Prepare the Branch Transformation Toolkit 4.3 project that will be migrated. In this Redbook, we use the standard Branch Transformation Toolkit 4.3 application sample to illustrate the migration tasks.
- Gather all the definition files of the existing version 4.3 application system. The migration tool can migrate the definition files on the server side from version 4.3 to version 5.1, besides processing the tags based on the specifications of version 4.3. If there are any special tag definitions, the migration tool may not be able to migrate them successfully. To overcome this, either customize the migration tool to extend the process so that it contains special definitions for the application or perform a manual migration later.
- The migration tool cannot migrate the business logic of an application directly to version 5.1. The tool creates skeletons with names corresponding to the server operations or the actions in a flow. However, you can make extensions to the migration tool so that it automatically moves the business logic to the generated code. The project team should carry out an investigation to check whether or not it is worthwhile to extend the migration tool to add more functions.
- To maintain the integrity of a migration, do *not* attempt to customize the migration process that handles screen flow generation, business flow generation, and the version 5.1 tooling artifact generation.
- The project team must package the application into a JAR file, copy the JAR file into the plug-in directory of the migration tool, and then modify the plug-in.xml to include the library. This is because the migration tool has to refer to the applications when it does the migration.
- The migration tool can migrate non-step server operations to either a Single Action EJB or a Business Process. However, the project team must separate the definition files and the application JARs into different sets if the application is to be migrated with both sets. The migration tool allows the creation of multiple projects to migrate different sets of applications.

- Although the migration tool supports services on the server side, the JDBC Table Service and Electronic Journal Service are left unchanged, while the SNA LU0/6.2 communication services are changed to JCA connectors. This means that you must find a different solution for these unsupported services, for example, Lotus® Notes® (R) support, LDAP support, MQ connector, or the relevant JCA connector can be used. Alternately, you can implement a new service based on the new service architecture. The migration tool can migrate server side service definition by modifying the class information for the services supported in Branch Transformation Toolkit 5.1. The migration tool also modifies the DummyDB2Journal.
- The event mechanism is changed to fit the Branch Transformation Toolkit version 5.1 framework in order to be distributable and to span different servers. The project team must check and modify the application accordingly if the event mechanism is adopted on the server side of the current application system.
- For the client applications, there are no changes. The project team does not have to migrate it to version 5.1. The only thing to be done is to verify the migration invoker and the related *.properties files to check whether they match the server operation names.
- If the migrated application runs only on WebSphere Application Server, use WebSphere Studio Application Developer to migrate the application. If the migrated application runs on WebSphere Business Integration Server Foundation, use WebSphere Studio Application Developer Integration Edition for the migration. This provides more feature support in the final application. The migration tool itself provides fewer features on WebSphere Studio Application Developer than it does on WebSphere Studio Application Developer Integration Edition. The main difference is that there is no support for generating business processes when you use WebSphere Studio Application Developer. In WebSphere Studio Application Developer, step operations and flow processors cannot be migrated. You should either switch to using non-step operations or skip migration of these operations and create the Single Action EJB manually. Non-step operation can only be migrated to a Single Action EJB.

4.2 Setting up the migration tools

Before migrating a Branch Transformation Toolkit 4.3 application, ensure that the correct plug-in JARs are in the right directories. If not, copy the following plug-ins from the plug-ins/wsadie51 directory of the Branch Transformation Toolkit 5.1 install package into the wstools/eclipse/plug-ins/ directory of the WebSphere Studio Application Developer Integration Edition.

- com.ibm.btt.tools.bp_5.1.0

- com.ibm.btt.tools.chaeditor.model.emf_5.1.0
- com.ibm.btt.tools.chaeditor_5.1.0
- com.ibm.btt.tools.common_5.1.0
- com.ibm.btt.tools.fmteditor.model.emf_5.1.0
- com.ibm.btt.tools.fmteditor_5.1.0
- com.ibm.btt.tools.gw.model.emf_5.1.0
- com.ibm.btt.tools.gw_5.1.0
- com.ibm.btt.tools.migration_5.1.0
- com.ibm.btt.tools.struts_5.1.0
- com.ibm.btt.tools.webdiagrameditor_5.1.0

Because the migration tools refer to the Branch Transformation Toolkit 4.3 application during the migration phase, before migrating the application, package the application into a JAR file and ensure that this JAR is included by the migration tool. These steps are necessary for migration because the migration tool must use the JAR files to initialize the Branch Transformation Toolkit environment and to read the related information about the application. Include some server-side files also in the runtime library.

Perform the following tasks:

1. Package the server side application into JAR files.
2. Copy the JAR files into the migration tool plug-in directory under the runtime folder.
3. Modify plug-in.xml to include the JARS in the runtime library. Example 4-1 shows a plugin.xml after modification for migration.

Example 4-1 Plugin.xml modified for migration

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
id="com.ibm.btt.tools.migration"
name="BTT Migration Tool Plug-in"
version="1.0.0"
provider-name="IBM"
class="com.ibm.btt.tools.migration.BTTMigrationPlugin">
<runtime>
    <library name="bttmigration.jar">
        <export name="*"/>
    </library>
    <library name="runtime/bttstrutstool.jar"/>
    <library name="runtime/dseb.jar"/>
    <library name="runtime/dseflp.jar"/>
                           :
                           :
    <library name="runtime/yourapp1.jar"/>
    <library name="runtime/yourapp2.jar"/>
</runtime>
                           :
                           :
</plugin>
```

For the migration sample in this book, perform the following steps:

1. In the Java perspective of WebSphere Studio, change to the **Package Explorer** view and expand the **DSE_SampleApplicationWeb** project.

2. In the JavaSource folder, select the packages **com.ibm.dse.samples.appl**, and **com.ibm.dse.samples.comms**, and the files **commssample.properties** and **sampleapplserver.properties**, as shown in Figure 4-1.

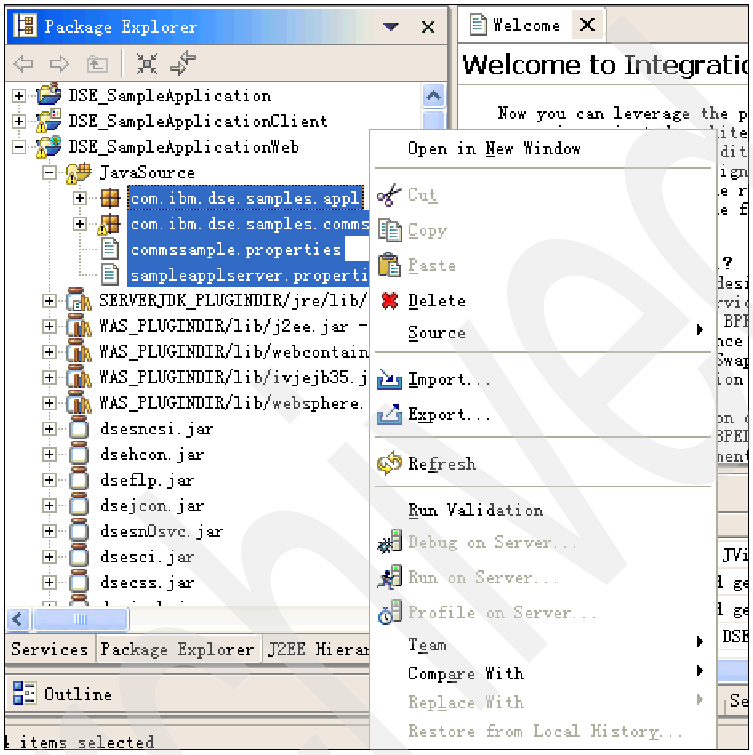
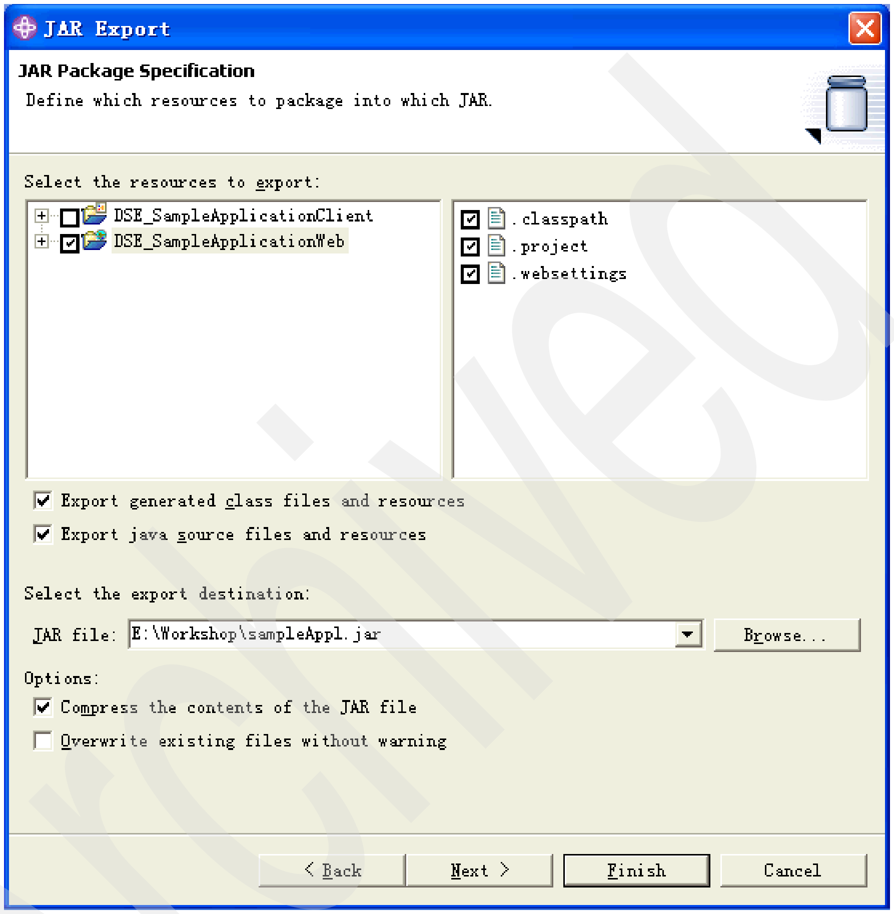


Figure 4-1 Export JAR file of sample application

3. Right-click and select **Export** → **JAR file**. Click **Next**.

4. In the next window, check **Export java source files and resources**, enter `sampleAppl.jar` as the JAR file name, select an export destination, and click **Finish**, as shown in Figure 4-2.


JAR Export

JAR Package Specification

Define which resources to package into which JAR.

Select the resources to export:

- [ ] DSE_SampleApplicationClient
- [✓] DSE_SampleApplicationWeb

- [✓] .classpath
- [✓] .project
- [✓] .websettings

[✓] Export generated class files and resources

[✓] Export java source files and resources

Select the export destination:

JAR file: E:\Workshop\sampleAppl.jar

Options:

[✓] Compress the contents of the JAR file

[ ] Overwrite existing files without warning

Figure 4-2 JAR file export details

5. Add the JAR file to the plugin runtime directory of the Branch Transformation Toolkit migration tool.
   a. Copy sampleAppl.jar to the \wstools\eclipse\plugins\com.ibm.btt.tools.migration_5.1.0\runtime\ directory of your WebSphere Studio Application Developer Integration Edition installation.
   b. Modify plugin.xml found in the WebSphere Studio Application Developer Integration Edition installation directory \wstools\eclipse\plugins\com.ibm.btt.tools.migration_5.1.0\to include the

JAR file. Between the <runtime> and </runtime> tags in the plugin file, add the following line:

```
<library name="runtime/sampleAppl.jar"/>
```

Save and close the plugin file.

6. In the WebContent folder of the DSE_SampleApplicationWeb project, copy all the JSP files, and paste them into a temporary directory, for example, C:\temp\jsp, for later use.

4.3 Customizing the migration tools

During the migration phase, you can customize the migration tool to enrich its functionality. To do this, perform the following tasks:

- Customize the code generation process. The migration tool generates only a code skeleton when it does the migration. You should then enhance its functionality, for example, modify it so that it copies the business logic from the server operation, or the action of the flow to the code generation logic. To do this, extend or change the classes of the migration tool.
- Customize the definition file migration process by extending or changing the migration tool base classes.
- Customize the user interface to the custom migration process. The user interfaces of the migration tool are based on the Eclipse programming model.

5

Migrating an application

This chapter describes how to migrate an application by using the migration tool. This chapter discusses the following topics:

5.1 Creating a new migration project

The overall steps to performing a migration are:

1. Create a migration project by using the Branch Transformation Toolkit migration tool and import the definition files from a specific EAR file, or specify the location of a dse.ini file on your disk. After the process is completed, the migration tool creates the related projects, including an EAR project, a Web project, and an EJB project. A Services project is created if you use WebSphere Studio Application Developer Integration Edition for your migration.
2. Migrate the application automatically by using the tool. Using this process, most of the construction work required to complete a migration can be performed by the migration tool. The tool can migrate the DSE ini file, DSE data file, and the business logic of the application. It can also generate Branch Transformation Toolkit 5.1 tooling artifacts for the Graphical Builder, based on the generation of the screen flow.
3. Perform any manual tasks that are required for the migration.
4. Use the migration tool to diagnose and analyze any migration problems.

Note: The phases are iterative and contain tasks that might have to be refined during the project life cycle.

As described in the previous chapters, before beginning the real migration, perform a migration analysis and customize the migration tool for any special needs. When these tasks have been completed, you can begin a migration by creating a migration project.

In our sample, we used the migration wizard to create a new project with the type *BTT Migration Project* in the Branch Transformation Toolkit collection. To do this, perform the following steps:

1. Start WebSphere Studio Application Developer Integration Edition and create a new workspace. From the main menu, select **File** → **New** → **Other...**

2. In the open dialog box, select **IBM Branch Transformation Toolkit** from the left navigation panel, and select **BTT Migration Project** from the right panel. Click **Next**, as shown in Figure 5-1.

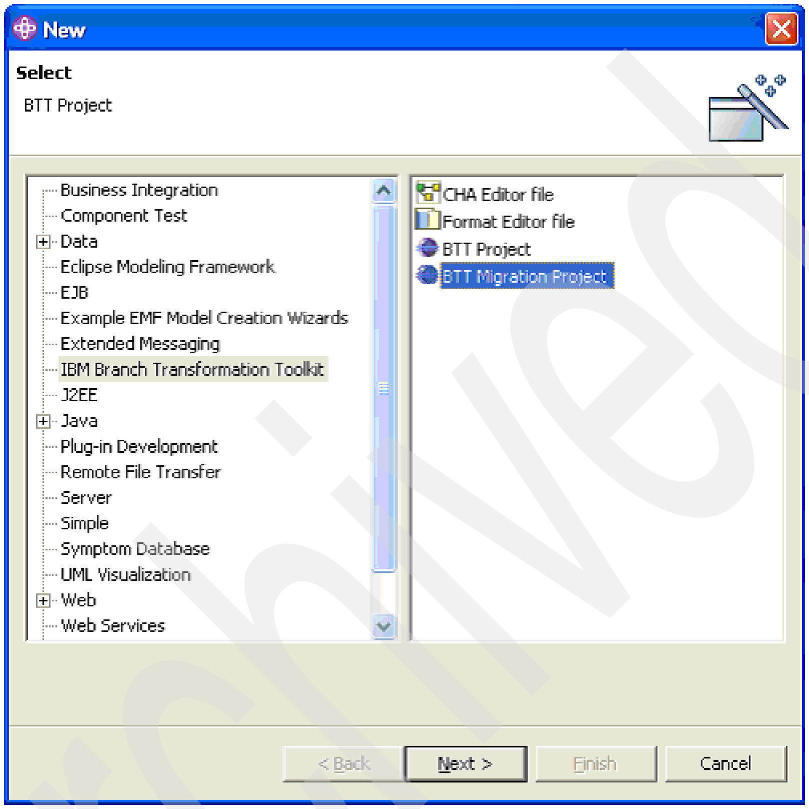
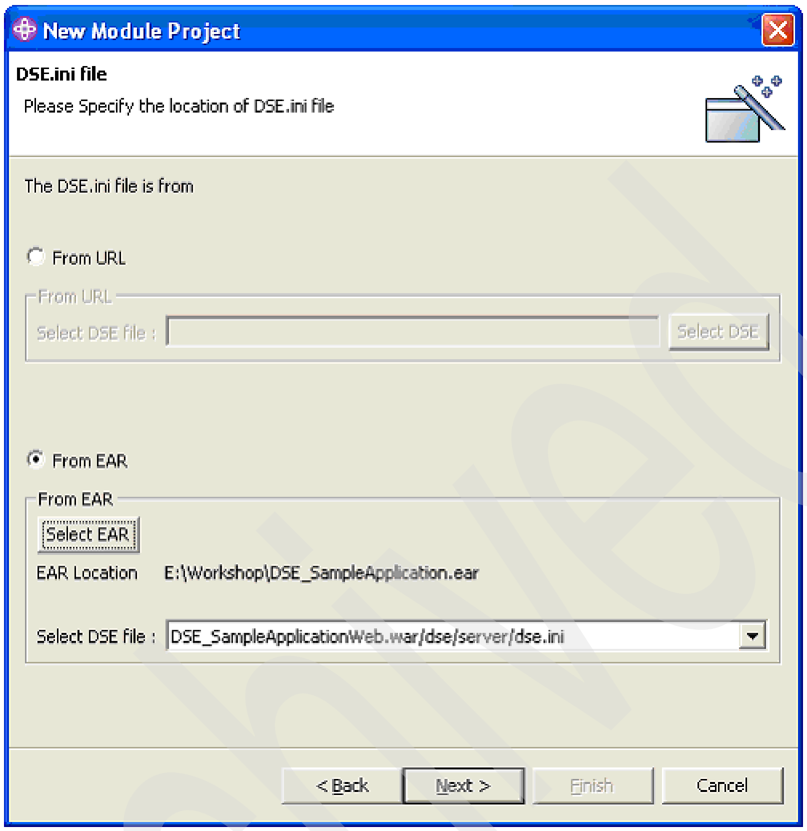


Figure 5-1 New migration project

3. For the J2EE Specification version, select **J2EE 1.3 Enterprise Application Project**. Click **Next**.
4. Enter `BaseSample` as the project name. Click **Next**.
5. In the next window, select **From EAR**. Click **Select EAR** to navigate to the DSE_SampleApplication.ear file. Click **Next**.

   From the Dse.ini panel as shown in Figure 5-2 on page 100, select the dse.ini file of the project you are migrating.



EAR Location E:\Workshop\DSE_SampleApplication.ear

Select DSE file : DSE_SampleApplicationWeb.war/dse/server/dse.ini

Figure 5-2 Selecting the DSE.ini file

You can choose the dse.ini file in the following ways:

– Select the dse.ini from your local directory.

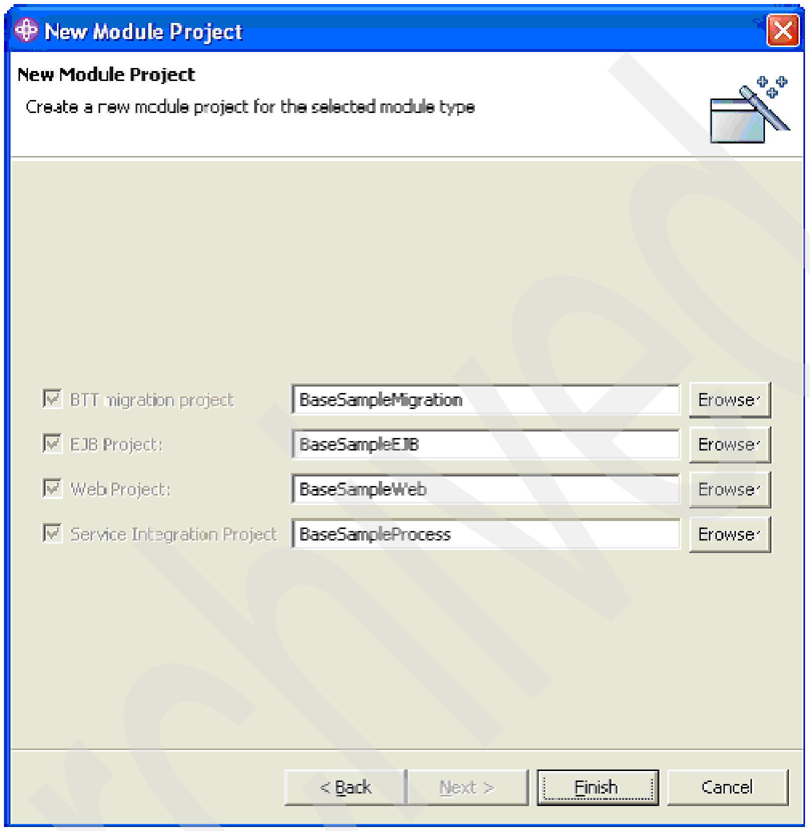
  To do this, select **From URL** and click **Select DSE** to browse to your dse.ini file and select it.

– Select the EAR file containing the dse.ini file.

  To do this, select **From EAR** and click **Select EAR** to browse to your EAR file and select it. You can then select the dse.ini file from the Select DSE File list.

Note: Because the migration tool only migrates server-side application components, make sure the dse.ini file you selected is the server-side one.

6. You can now see the default names of the Branch Transformation Toolkit 5.1 projects that will be created, as shown in Figure 5-3. Click **Finish**. The selected project is created and the Graphical Builder is started automatically.



- [✓] BTT migration project: BaseSampleMigration — Browse...
- [✓] EJB Project: BaseSampleEJB — Browse...
- [✓] Web Project: BaseSampleWeb — Browse...
- [✓] Service Integration Project: BaseSampleProcess — Browse...

Figure 5-3 Migration project names

7. Before continuing, you have to make some changes to one of the definition XML files.
   a. Expand the **BaseSampleMigration** project.
   b. Navigate to the 4.3Definitions/selfDefine folder.
   c. Open **sampleHtmlFlow.xml** with the XML Editor.
   d. Find the following definition:

```
<htmlState id="userErrorInfoPage" type="operation"
typeIdInfo="error.jsp">
```

   Change this to:

```
<htmlState id="userErrorInfoPage" type="page" typeIdInfo="error.jsp">
```

e. Save the file.

f. Close and reopen WebSphere Studio Application Developer Integration Edition.

> **Note:** You must restart WebSphere Studio Application Developer Integration Edition if you change any content in the Branch Transformation Toolkit external files.

After you create the migration project, you will find all the related definition files copied into the category of 4.3 definition. You will also see that EAR, EJB, Web, and Services projects are created.

5.2 Using the migration tools

Branch Transformation Toolkit provides a set of tools to help you migrate toolkit applications developed with version 4.3 of the toolkit, to the version 5.1 architecture.

The migration tools perform the following tasks during a migration:

- Migrate the version 4.3 definition dse.ini file to version 5.1.
- Migrate the version 4.3 dse data definition files to version 5.1 requirements, including dsectx.xml context definitions, dsedata.xml data definitions, dsefmt.xml formatter definitions, dsetype.xml type definitions, dsesrvce.xml service definitions, and so on.
- Generate the corresponding runtime code, including business processes, Java snippets, Single Action EJBs, and so on. These taskscode includes:
  - Migrating the server operations to Single Action EJBs or business processes if WebSphere Studio Application Developer Integration Edition is used. After you complete the process, the migration tool will generate invokers (Java code), and Single Action EJBs or business processes.
  - Migrating the flow processes to business process if WebSphere Studio Application Developer Integration Edition is used. After you complete the process, the migration tool will generate the BPEL file and invoker (Java code).
  - Migrating the screen flow to Struts extensions. After you complete the process, the migration tool will generate the Struts config file, Web diagram (.gph file), and migrated JSPs.
  - Migrating self-definition files, if any, as well as the Struts config file, Web diagram (.gph file), and migrated JSPs?). Also, the definitions of the

context, the data element, the type data and the format in the self-definition files will be added into the generic definition files.

- Generate version 5.1 tooling artifacts such as Graphical Builder definition files.

The migration tools provide GUIs to help you migrate the toolkit configuration files, data model, formatter definitions, server operations, flow processes, and screen flows of the version 4.3 application to the corresponding components of the version 5.1 application. Furthermore, the migration tools are designed to be a scalable toolset for you to build your own batch migration applications using a set of APIs. You can also customize the migration tools for your special needs.

Note: After migration, do *not* change the names of toolkit constructs such as contexts, formatters, and so on. The names of the constructs are assigned automatically by the migration tool. If you change the name of a construct manually after the migration, the migrated application will not be able to find that construct anymore.

5.2.1 Migrating the dse.ini file

In this section, we provide an example to describe how to migrate the definition files from the version 4.3 toolkit sample application to version 5.1. Perform the following tasks to do the same:

1. Expand the **BaseSampleMigration** project in the 4.3 Definitions folder, right-click the **dse.ini** file and choose **BTT Migration** → **DSE ini File Migration**, as shown in Figure 5-4.

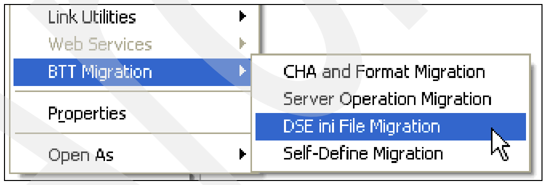
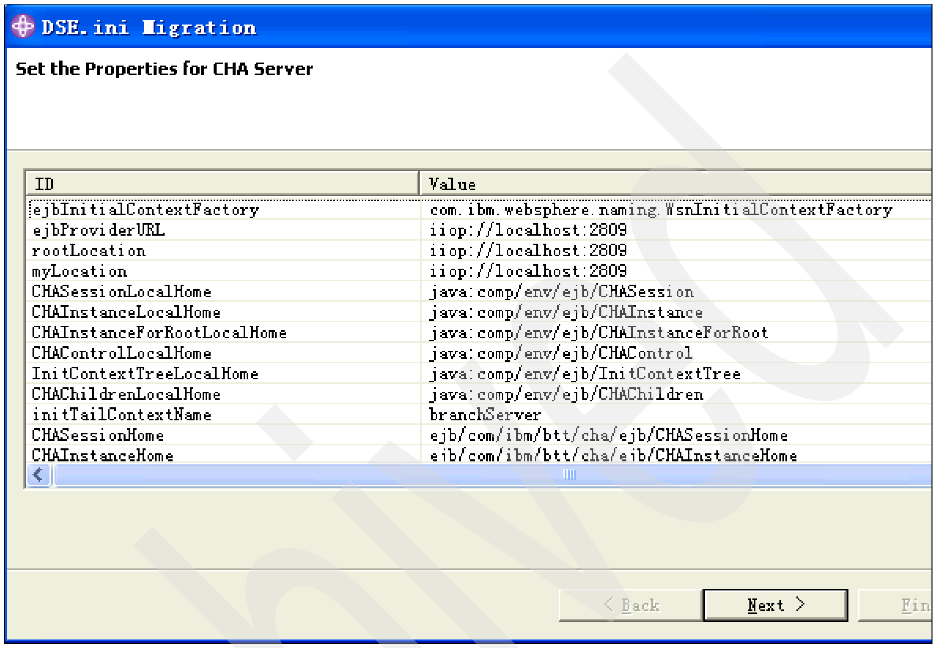


Figure 5-4 Starting DSE ini file migration

2. The properties of the CHA Server are displayed, as shown in Figure 5-5. To change any properties, double-click the property value. To continue with the migration wizard, click **Next**.



Set the Properties for CHA Server

ID	Value
ejbInitialContextFactory	com.ibm.websphere.naming.WsnInitialContextFactory
ejbProviderURL	iiop://localhost:2809
rootLocation	iiop://localhost:2809
myLocation	iiop://localhost:2809
CHASessionLocalHome	java:comp/env/ejb/CHASession
CHAInstanceLocalHome	java:comp/env/ejb/CHAInstance
CHAInstanceForRootLocalHome	java:comp/env/ejb/CHAInstanceForRoot
CHAControlLocalHome	java:comp/env/ejb/CHAControl
InitContextTreeLocalHome	java:comp/env/ejb/InitContextTree
CHAChildrenLocalHome	java:comp/env/ejb/CHAChildren
initTailContextName	branchServer
CHASessionHome	ejb/com/ibm/btt/cha/ejb/CHASessionHome
CHAInstanceHome	ejb/com/ibm/btt/cha/ejb/CHAInstanceHome

Figure 5-5 CHA server properties

3. The remaining pages of the wizard show the properties of the formatter service, the service server, and the work area respectively. Keep all the properties values as default and click **Next** to proceed through all the wizard pages. Click **Finish** to complete the wizard.

   The wizard migrates the version 4.3 dse.ini file to a version 5.1 file and copies the 5.1 dse.ini file into the 5.1 definitions folder of the application migration project.

5.2.2 Migrating data and format definitions

This section describes how to migrate data and format definitions. Perform the following tasks:

1. Make sure that all the IDs of the data fields start with a lowercase character. If the first character of a data field ID is an uppercase character, change that first character into a lowercase one. For example, change the code that is similar to the one shown in Example 5-1 on page 105 to code that looks like the one shown in Example 5-2 on page 105.

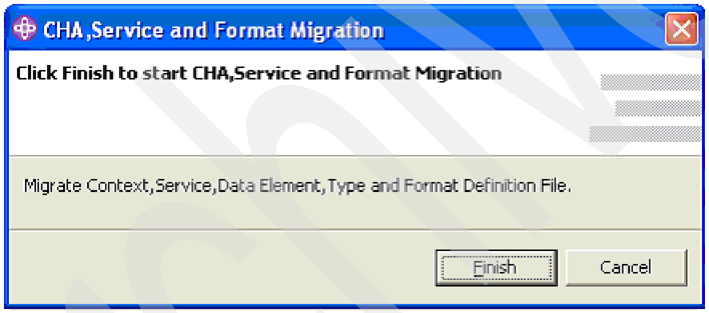
Example 5-1 Data field ID using uppercase

```
kColl id="individualSessionData">
  <field id="LogonPortal" value="1"/>
</kColl>
```

Example 5-2 Data field ID using lowercase

```
<kColl id="individualSessionData">
  <field id="logonPortal" value="1"/>
</kColl>
```

2. In the 4.3 Definitions folder, right-click the **dse.ini** file and select **BTT Migration** → **CHA and Format Migration**. Click **Finish** as shown in Figure 5-6.



Click Finish to start CHA,Service and Format Migration

Migrate Context,Service,Data Element,Type and Format Definition File.

Figure 5-6 CHA, service, and format migration

The migration tool migrates the following files from version 4.3 to the version 5.1 definitions folder of the application migration project:

- dsedata.xml
- dsetype.xml
- dsectxt.xml
- dsefmts.xml
- dsesrvce.xml

Note: The migration tool only copies dsesrvce.xml from 4.3 Definitions to 5.1 Definitions. You should modify dsesrvce.xml manually.

5.2.3 Migrating server operations

In Branch Transformation Toolkit version 5.1, server operations are replaced by either Single Action EJBs or business processes called by invokers.

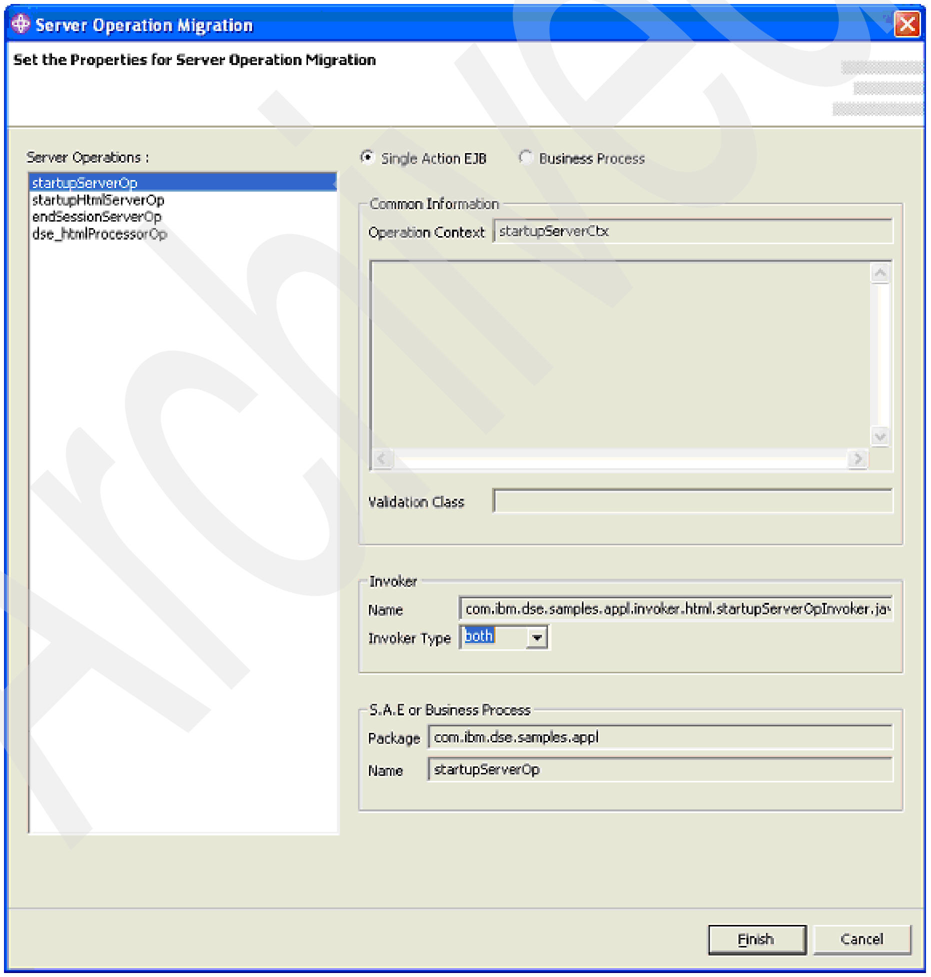
To migrate the nonstep server operations to Single Action EJBs or business processes, and generate the associated invokers, perform the following tasks:

1. In the 4.3 Definitions folder, right-click **dseoper.xml** and select **BTT Migration** → **Server Operation Migration**. This starts the server operation file migration wizard.
2. From the Server Operation Migration dialog box, select the server operations, and decide if you want the nonstep server operations to be migrated to Java snippets for business processes or Single Action EJBs. An invoker will also be generated for calling the Single Action EJB or business process. From the Server Operation Migration dialog box, you can see the name of the invoker,

the name of the Single Action EJB or business process, and the package where the Single Action EJB and business process locates.

For this sample, migrate all the non-step operations to the default **Single Action EJB**, in the pop-up window in Figure 5-7. For the Invoker Type, select **both**.

Note: The selection of Java snippets for business processes or Single Action EJBs affects all the nonstep server operations that are listed in the server operation file migration wizard. To have one set of server operations migrated to Single Action EJB and another set that is not, create two migration projects for each set.


Server Operation Migration
Set the Properties for Server Operation Migration
Server Operations :
Common Information
Invoker
S.A.E or Business Process
Finish
Cancel

Server Operations: startupServerOp, startupHtmlServerOp, endSessionServerOp, dse_htmlProcessorOp

[✓ Single Action EJB] Business Process

Operation Context: startupServerCtx

Validation Class:

Name: com.ibm.dse.samples.appl.invoker.html.startupServerOpInvoker.ja

Invoker Type: both

Package: com.ibm.dse.samples.appl

Name: startupServerOp

Figure 5-7 Server Operation Migration

3. Click **Finish**. You can see the generated invoker and its properties files in the Web project of the application, the Single Action EJBs in the EJB project of the application, and the business processes in the Process project of the application. The entry properties file is also upgraded in the application projects.

   Note: The entry properties file is a kind of index or invoker registry to map an invoker ID to an invoker properties file.

Stepped operations in toolkit version 4.3 can only be migrated to business processes. To migrate the stepped server operations to business processes and generate the associated invokers, perform the following tasks:

1. From the Server Operation Migration dialog box, select the server operations. You can see the name of the invoker, the name of the Single Action EJB or business process, and the package where the Single Action EJB and business process are located.
2. Click **Finish**. You can see the generated invoker and its properties files in the Web project of the application, and the business processes in the process project of the application. The entry properties file is also upgraded in the application projects.

The operation steps are migrated to Java snippets for business processes. Generally, links between the Java snippets are automatically based on the server operation definition files.

If the operation contains suboperations, the stepped suboperations will be migrated to business processes with associated invokers generated, and the nonstep suboperations migrated to Java snippets of the BaseOperSnippet type.

Note: The migration tool automatically migrates Server Operation, which has steps to BPEL process. Tools allow users to migrate Nonstep Server Operation to SAE or BP. In default, it will be migrated to SAE.

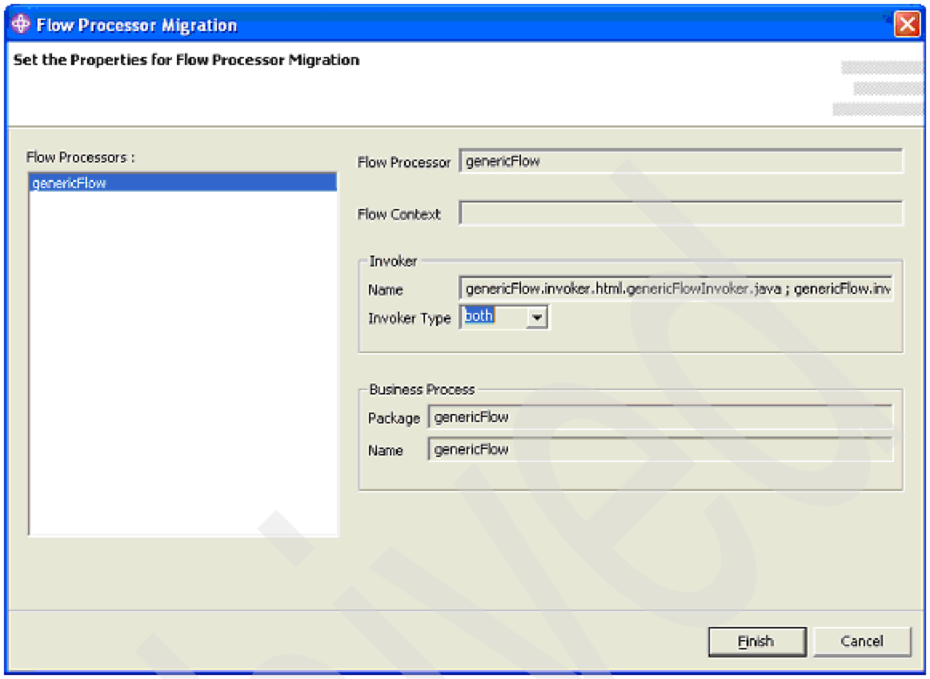
To customize the nonstep operation migration option, select the operation in the Server Operation list in the left side of the panel, and then select **Single Action EJB** or **Business Process** in the right side of the panel. In the same way, you can select another operation in the list and choose the migration target option in the left.

5.2.4 Migrating flow processors

In Branch Transformation Toolkit version 5.1, flow processors are replaced by business processes or Single Action EJBs on the server side. To migrate the server-side flow processors to the corresponding constructs in version 5.1, perform the following tasks:

1. From the Package Explorer view, in the 4.3Definitions folder, right-click dseproc.xml file and select **BTT Migration** → **Flow Processor Migration**. This starts the flow processor file migration wizard.
2. In the pop-up Flow Processor Migration dialog box, select **genericFlow**. For the Invoker Type, select **both,** as shown in Figure 5-8 on page 110.
   - The Actions defined in the flow processor definition file maps to Java^{TM} snippets.
   - Guard conditions map to extra navigation links as a leaf node in the business process. Conditions are added in links to control the navigation.
   - Transition conditions map to normal business process nodes.

   From the Flow Processor Migration dialog box, you can see the name of the invoker, the name of the Single Action EJB or business process, and the package where the Single Action EJB and business process are located.
3. Click **Finish**. Java snippets and business processes are created in the Process project of the application. Validation classes are also migrated.



Flow Processors :
genericFlow

Flow Processor: genericFlow
Flow Context:

Invoker
Name: genericFlow.invoker.html.genericFlowInvoker.java ; genericFlow.inv
Invoker Type: both

Business Process
Package: genericFlow
Name: genericFlow

Figure 5-8

Make sure the following requirements are met when migrating flow processors:

- Flow processor do not contain any page state.
- The subprocessors or descendent processors of the flow processor that you are migrating do not contain any page state.
- The flow processor definition file follows all the rules for defining a processor.

5.2.5 Migrating screen flow processors

In Branch Transformation Toolkit version 5.1, screen flow processors are re-based on the Apache Struts Framework. To migrate version 4.3 screen flows to the corresponding constructs in version 5.1, perform the following tasks:

1. From the Package Explorer view of WebSphere Studio Application Developer, right-click the **dseproc.xml** file that contains the version 4.3 screen flow processors and select **BTT Migration** → **Screen Flow Processor Migration**. This starts the screen flow processor file migration wizard.
2. From the Screen Flow Processor Migration dialog box, select the screen flow processors. From the Screen Flow Processor Migration dialog box, you can

see related information such as reference context, validation class, and Struts file name.

3. Click **Finish**.

Because there is no screen flow definition in the dseproc.xml file of the BTT4311 Base Sample in our sample, you can skip it.

Migrating screen flow processors accomplishes these tasks:

- Convert the <htmlProcessor> tag to a Struts process definition file.
- Convert Actions to Struts Actions.
- Convert operation actions to business processes defined in a BPEL file.
- Translate the navigation links into the Struts definition file. Transitions are transformed into the forward definitions of Struts.
- Convert version 4.3 validation classes to version 5.1 validation classes.
- Convert condition definitions to the condition definitions of Struts Extensions.
- Convert processor contexts to the processor CHA contexts of Struts Extensions.
- Convert transition actions to two Struts actions.
- Convert transition contexts to transition CHA contexts and Form Beans of Struts Extensions.
- For JSP files, change the next event to a URI.

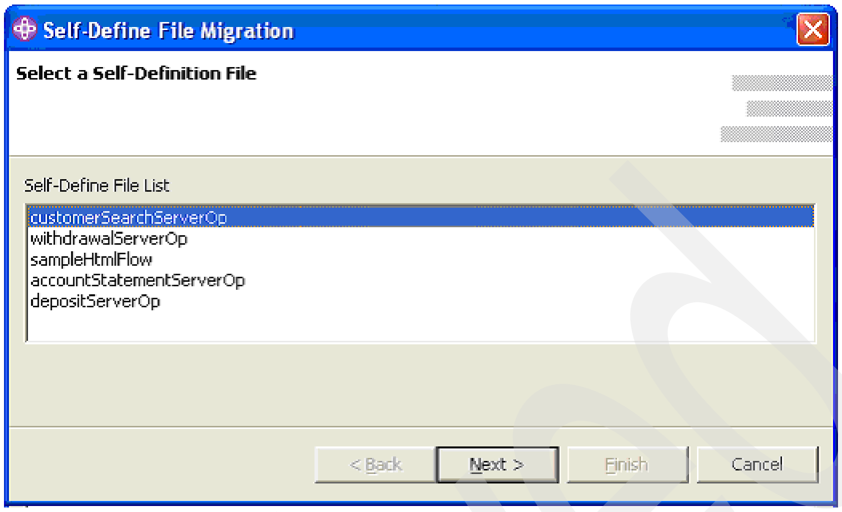
After the screen flow migration process is completed, the result is placed in the Web project. You can browse and edit the screen flow by the Struts configuration file in an XML editor or the .gph file in a graphical screen flow editor.

5.2.6 Migrating self-defined files

Migrating the self-definition files generates struts configure file, Web diagram (.gph file), and migrated JSPs. In addition to the struts extension-related files, it also adds definition of context, data element, type data, and format into the generic definition files.

For our sample, perform the following tasks:

1. In the 4.3 Definitions folder, right-click the **dse.ini** file and select **BTT Migration → Self-Define Migration**.
2. In the pop-up window, select a self-definition file from the list, and click **Next**, as shown in Figure 5-9 on page 112.

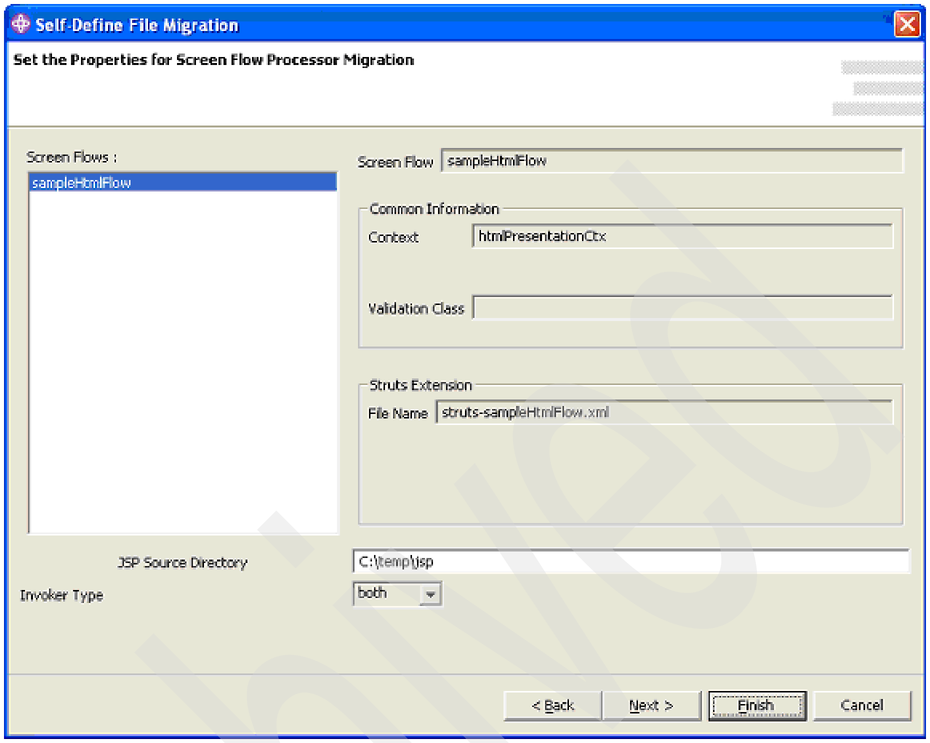


customerSearchServerOp
withdrawalServerOp
sampleHtmlFlow
accountStatementServerOp
depositServerOp

Figure 5-9 Selecting a self-definition file

3. The windows that follow show the context list, data element list, format list, type data list, server operations, and flow processors, in the same order. All you have to do is click **Next** to advance through them.
4. The next pop-up window shows the properties of the defined screen flows, if any. In this sample, you have to configure it only when you select **sampleHtmlFlow** from the Self-Define File List at the beginning. Input the JSP Source Directory, for example C:\temp\jsp, and select Invoker Type as **Both**, as shown in Figure 5-10 on page 113. Click **Next** or **Finish**. (After setting the properties for one definition file for migration, if you want to customize another file, click **Next**. This will take you to the beginning file list panel. If you have configured all the files, click **Finish**.)

   To carry out migration again, you should restart WebSphere Studio Application Developer Integration Edition.



Screen Flow: sampleHtmlFlow
Context: htmlPresentationCtx
Validation Class:
File Name: struts-sampleHtmlFlow.xml
JSP Source Directory: C:\temp\jsp
Invoker Type: both

Figure 5-10 Self-Define File Migration

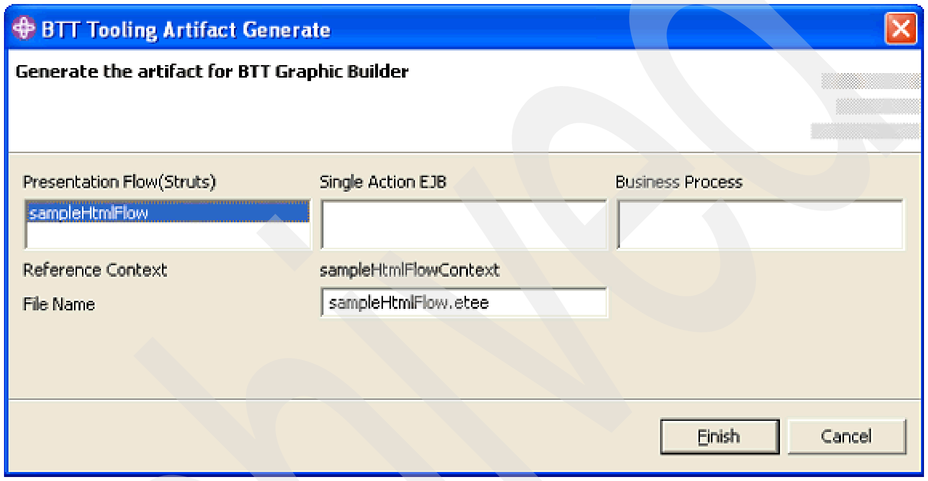
Note: Business processes do not automatically chain process contexts (equivalent to operation contexts and flow processor contexts in version 4.3) to session contexts. You should chain the process contexts to session contexts manually.

5.2.7 Migrating tooling artifacts

Apart from migrating the version 4.3 definitions and runtime components to version 5.1 constructs, the migration tool can also generate artifacts that can be used with toolkit version 5.1 to further develop or maintain your application. This function can generate a Graphical Builder file. When you open the file with Graphical Builder, you will see the components and definitions you migrated in the Graphical Builder views.

To generate tooling artifacts with the migration tool, perform the following tasks:

1. From the Package Explorer view, right-click your migration project, **BaseSampleMigration,** and select **BTT Migration** → **BTT Tooling Artifact Generate**.
2. From the Branch Transformation Toolkit Tooling Artifact Generate dialog box, select the screen flows, business processes, and Single Action EJBs from which you want to generate the tooling artifacts. For our sample, select **sampleHtmlFlow**.


BTT Tooling Artifact Generate

Generate the artifact for BTT Graphic Builder

Presentation Flow(Struts): sampleHtmlFlow

Single Action EJB

Business Process

Reference Context: sampleHtmlFlowContext

File Name: sampleHtmlFlow.etee

Figure 5-11 BTT Tooling Artifact Generate

3. Click **Finish**.

You will now find the Graphical Builder file (.eete) file in your migration project. Double-click the **Graphical Builder** file to start the Graphical Builder. You can see that the components you selected for artifacts generation are present, represented by Graphical Builder nodes.

5.2.8 Diagnosis for the migration tool

The migration tool places the messages into a log file named BTT_Migration.log. The file is placed in C:\. When something appears to be wrong in the migration process, check the log file to see the reason and carry out the necessary actions to make the migration tool work. This list represents the more common errors:

- The entity path is set as fromJAR. The migration tool does not support the path information set as fromJAR.

- Class not found. It is usually caused because the jar file of the application is not found in the plug-in directory, the plugin.xml file is not set correctly, or the class is not packaged in the jar files.
- Branch Transformation Toolkit environment initialization error. It is usually caused because the definition files are not defined correctly.
- The JSP path is not correct.

5.3 Manual modification for migration

In this section we explain some manual modifications that we made to the migration files. Our purpose is to provide examples of how to complete a successful migration because the migration tools cannot automatically complete everything that may be required to migrate an application.

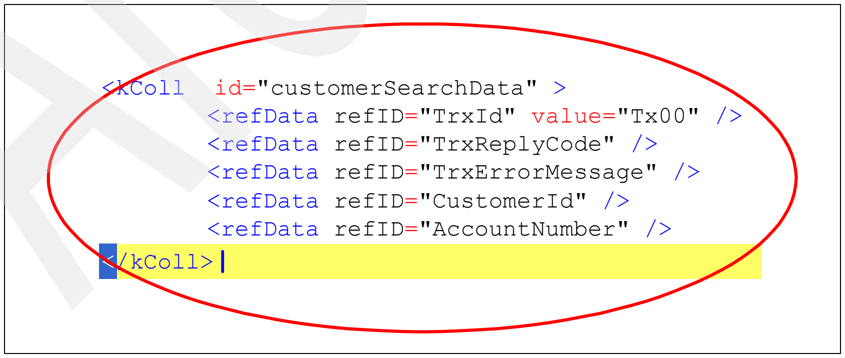
5.3.1 Modifying the generated definition files

Because automated migration tools cannot do everything, some changes have to be applied manually to the generated files. In our sample, they are located in the 5.1 Definitions folder of the BaseSampleMigration project. Before modifying the generated files, we recommend that you back up all the definition files. The modifications that we made include:

1. Modify the dse* definition files.

   a. For the dsedata.xml, find the customerSearchData kColl definition, add items `<refData refId="CustomerId" />` and `<refData refId="AccountNumber" />` to this kColl definition.

   The result should look as shown in Figure 5-12.



```
<kColl  id="customerSearchData" >
        <refData refID="TrxId" value="Tx00" />
        <refData refID="TrxReplyCode" />
        <refData refID="TrxErrorMessage" />
        <refData refID="CustomerId" />
        <refData refID="AccountNumber" />
</kColl>
```

Figure 5-12 Modifying dsedata.xml

You can also use Graphical Builder to do this work in CHA Data View.

   b. For dsesrvce.xml, change the content to that shown in Example 5-3 on page 116.

Example 5-3 Modifying desesrvce.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dsesrvce.xml>
   <DummyDB2Journal autoCommit="false" id="Journal" schema="SCHEMA01">
     <column dataName="UserId" id="USERID"/>
     <column dataName="TID" id="TERMINALID"/>
     <column dataName="HostBuff" id="DATABUFFER"/>
   </DummyDB2Journal>
   <DummyDB2Journal autoCommit="false" id="Journal2" schema="SCHEMA01">
     <column dataName="UserId" id="USERID"/>
     <column dataName="TID" id="TERMINALID"/>
     <column dataName="HostBuff" id="DATABUFFER"/>
   </DummyDB2Journal>
   <GenericPoolService id="GenericPool" serviceName="Journal2" initialSize="2"
maxPoolSize="10" timeBetweenRetries="2000"/>
</dsesrvce.xml>
```

2. Copy the dse* files to the server.

   Copy all the dse* files, including dse.ini, dsectxt.xml, dsedata.xml, dsesvrce.xml, and dsetype.xml from **BaseSampleMigration** → **5.1 Definitions** to the c:\dse directory. This is the default directory accepted by the application. To change to another location, make the corresponding changes in the definition files.

5.3.2 Creating a CHA database

The CHA provides context functionality in a distributed server environment. This enables Branch Transformation Toolkit applications to have distributed runtime repositories so that they can share information with other applications. Non-toolkit applications can use the CHA as a way of holding general global session information. Figure 5-13 on page 117 shows an overview of how Branch Transformation Toolkit applications use the CHA.

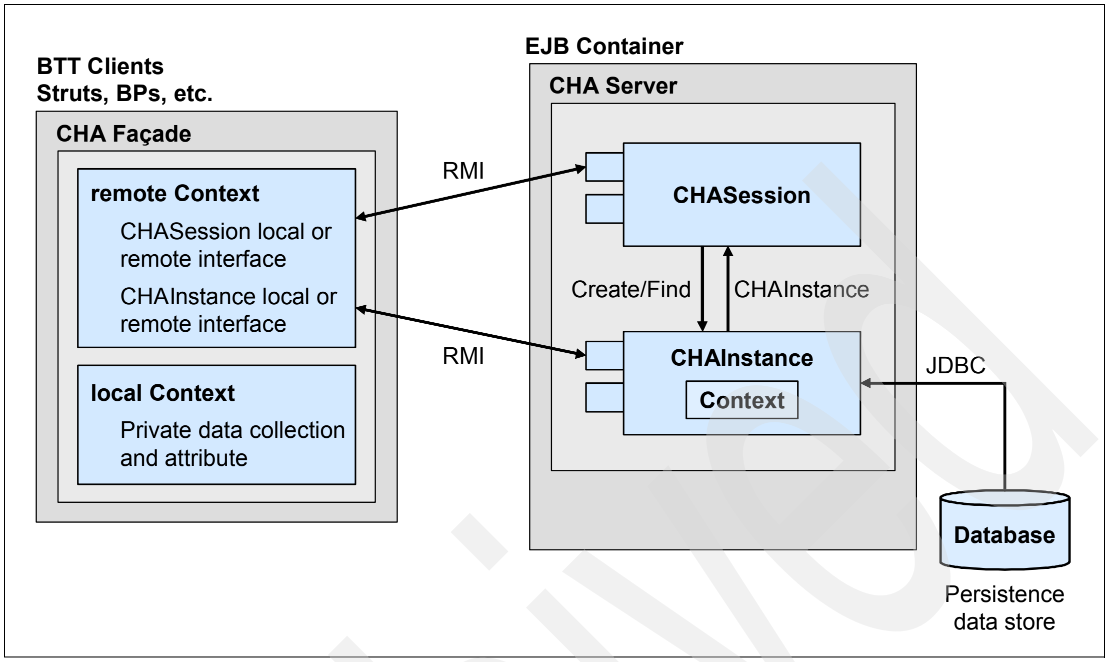


Figure 5-13 CHA overview

To fulfill the functionality of the CHA in Branch Transformation Toolkit 5.1 applications, a database is required. Branch Transformation Toolkit 5.1 supports most database systems. This versabilty makes it easy for you to select a database according to your application situation.

We do not provide specific database setup instructions in our redbook because these depend on the database product you choose to use. Consult the product documentation provided by your database vendor for detailed instructions. For the information about DB2 Universal Database™ 8.2 installation and configuration, refer to the DB2 manual.

Although most database systems are acceptable, in our sample, we presume that you are using IBM DB2 Universal Database 8.2.

To configure CHA with DB2 the steps to follow are:

1. Create a database named SAMPLE.

   Select **Start** → **All programs** → **IBM DB2** → **Command Window**, open the DB2 Command Window, enter:

   ```
   DB2 CREATE DATABASE SAMPLE
   ```

2. Locate the table creating script.

   In the DB2 Command Window, change your current directory to the same folder as the createCHATables.ddl file, which is provided by the Branch Transformation Toolkit 5.1 Toolkit, for example, `C:\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1.0.1\dbtools\Windows\DB2\tableDefinition\cha\`

3. Connect to the SAMPLE database.

   In the DB2 Command Window, type the following:

   ```
   DB2 CONNECT TO SAMPLE USER username USING password
   ```

   Replace the *username* and *password* according to your DB2 account information. In this sample, we presume the username and password are both *db2admin*. Ensure that you have enough privilege in the database. You will see some basic database information after getting connected.

4. Create tables using the DDL script file.

   In the DB2 Command Window, enter the following:

   ```
   DB2 -tvf createCHATables.ddl
   ```

   Look for messages indicating that tables CHAInstance, CHAChildren, and CHAControl have been created successfully.

   After completing this task, you can check the DB2 system using the DB2 control center. You can see the newly created database SAMPLE containing three tables: CHAChildren, CHAControl, and CHAinstance.

5.3.3 Fixing errors

After migrating the application from version 4.3 to version 5.1, you are likely to encounter errors in related projects. You might have to perform certain modifications to fix errors such as code problems, package names, access to the context, the data elements, the format, the JDBC services, and so on.

In addition to this, you need add the business logic by moving the code from the source code of the current application, unless the project team has already enhanced the migration tool to have the capability to copy the business logic into the generated code. You should also build the services that version 5.1 does not support, and change the access to the communication service to use JCA as is now required. Details and instructions pertaining to these activities are detailed in the remaining chapters of this book.

6

Post-migration activities

This chapter discusses several post-migration activities that have to be handled to finish the migration of the sample project

We describe the following topics:

6.1 Sample project requirements

In our sample, although we fixed the errors caused by migration, as described at the end of Chapter 3, “Planning a Branch Transformation Toolkit migration” on page 57, many unsolved errors continued to remain in the new project. Before continuing with the post-migration work, the following actions should be performed to meet the project's requirements.

The migration tool does not automatically migrate everything from version 4.3 to version 5.1. Under certain conditions, you have to migrate some things manually:

- The base functionality of the migration tool will not migrate the logic inside the application code of the server operations and the actions of the flow process.

  If you have not customized the migration tool to include the process for moving the logic to the generated code, you must move it manually.

- Services migration is not included in the migration tool.

  If there are any services in version 4.3 application that are out of the scope of the version 5.1 system, you should migrate them manually.

- The server side event mechanism of version 5.1 is changed and is not migrated by the migration tool.

  You must migrate it manually.

- The communication service access is changed to use a JCA connector.

  You must manually modify it in the Single Action EJB or the activity of business process.

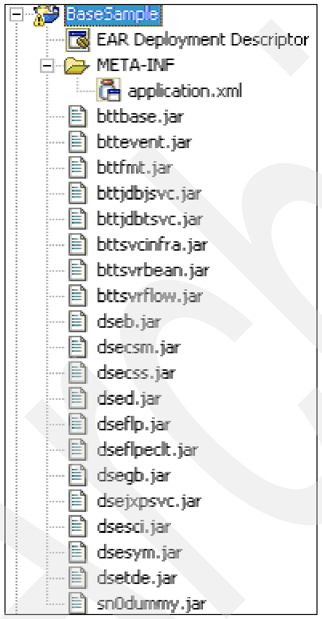
6.1.1 Copying the required JAR files to the BaseSample project

Copy the following JAR files to the BaseSample project from the BTTv51_dir\jars directory. Here, BTTv51_dir is the Branch Transformation Toolkit 5.1 installing packaging path, for example, C:\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1.

- bttbase.jar
- bttevent.jar
- bttfmt.jar
- bttjdbjsvc.jar
- bttjdbtsvc.jar
- bttsvcinfra.jar
- bttsvrbean.jar
- bttsvrflow.jar
- dseb.jar
- dsecsm.jar

- dsecss.jar
- dsed.jar
- dseflp.jar
- dseflpeclt.jar
- dsegb.jar
- dsejxpsvc.jar
- dsesci.jar
- dsesym.jar
- dsetde.jar
- sn0dummy.jar

In our sample, sn0dummy.jar could be extracted from dummysnalu0.rar within the same folder. Extract **C:\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1\jars\dummysnalu0.rar**, and then copy **sn0dummy.jar** to the BaseSample project. See Figure 6-1.



```
BaseSample
    EAR Deployment Descriptor
    META-INF
        application.xml
    bttbase.jar
    bttevent.jar
    bttfmt.jar
    bttjdbjsvc.jar
    bttjdbtsvc.jar
    bttsvcinfra.jar
    bttsvrbean.jar
    bttsvrflow.jar
    dseb.jar
    dsecsm.jar
    dsecss.jar
    dsed.jar
    dseflp.jar
    dseflpeclt.jar
    dsegb.jar
    dsejxpsvc.jar
    dsesci.jar
    dsesym.jar
    dsetde.jar
    sn0dummy.jar
```

Figure 6-1 Sample project JAR files

6.1.2 Copying the response.res file to the BaseSampleWeb project

Copy **response.res** from the BTT5.1 installpackaging path, for example, C:\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1\ samples\JavaSampleApplication\StandAlone\ BTTJavaSample.ear, to the WebContent directory of the BaseSampleWeb project.

In our sample, the file response.res is a host reply file, and will be used by the DummyLu0SnaSession service when the server has to simulate host reply. See Figure 6-2.

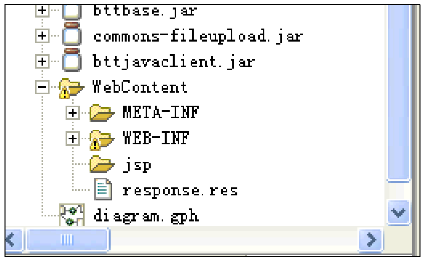

bttbase.jar
commons-fileupload.jar
bttjavaclient.jar
WebContent
META-INF
WEB-INF
jsp
response.res
diagram.gph

Figure 6-2 Sample project Web content

6.1.3 Creating a Java project to include user-defined classes

To create and modify a Java project, follow these steps:

1. From the main menu, select **File** → **New** → **Other...**, and use the wizard to create a Java project named `UserDefineJar`. Click **Finish**. The user-defined services will be contained in the UserDefineJar. See Figure 6-3 on page 123.

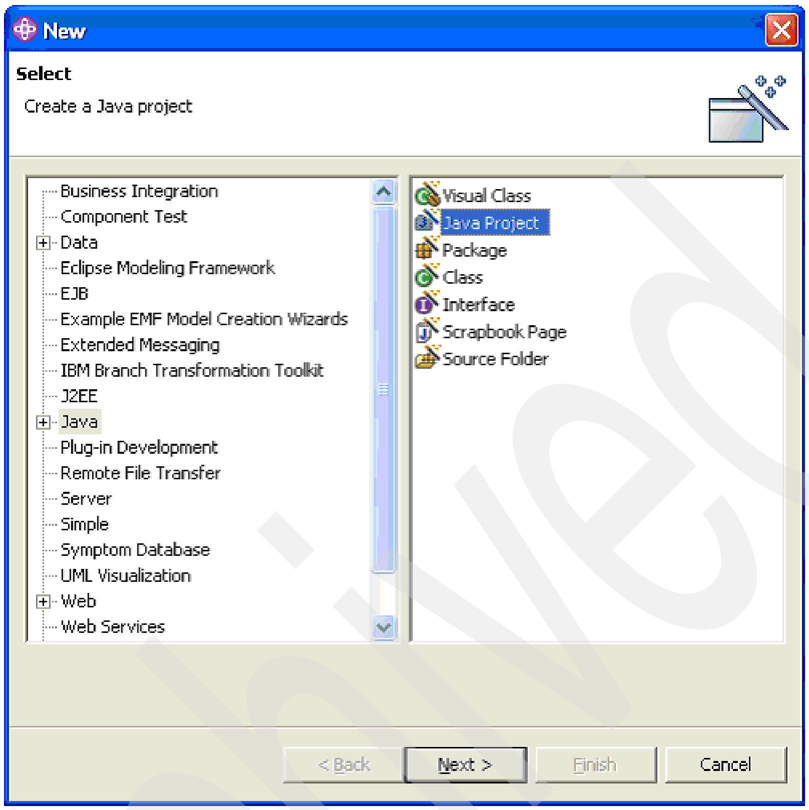


Figure 6-3 Create Java project

2. Under this project, create a package named `com.ibm.dse.samples.appl`, and copy three Java files, **HostDecorator.java**, **HostField.java**, **HostStringFormat.java** from the com.ibm.dse.samples.appl package of the DSE_SampleApplicationWeb project in the Branch Transformation Toolkit 4.3 Base Sample Application's workspace, to this package.

6.1.4 Modifying invoker's properties files

In our sample, since SAE is used as the server business operation, you require an invoker to execute the SAE. During the migration process, modify the properties files to make the bean invoker work.

1. Expand **BaseSampleWeb project** in J2EE perspective's Project Navigator view, and browse to the folder

JavaResources\accountStatementServerOp.invoker.java. Open the file **accountStatementServerOpOP.properties** to edit.

a. Change the last line from `"isLocal=true" to "isLocal=false"`.
b. Append a new line: `"csReplyFormat=accountStatementRepFmt"`.

Similarly, four more properties files should be modified.

2. In BaseSampleWeb project, browse to the folder **JavaResources\com.ibm.dse.samples.appl.invoker.java**, and open the file **startupServerOpOP.properties** to edit.
   a. Change the last line from `"isLocal=true" to "isLocal=false"`.
   b. Append a new line: `"csReplyFormat= startupReqFmt"`.
3. In BaseSampleWeb project, browse to the folder **JavaResources\customerSearchServerOp.invoker.java**, and open the file **customerSearchServerOpOP.properties** to edit.
   a. Change the last line from `"isLocal=true" to "isLocal=false"`.
   b. Append a new line: `"csReplyFormat= customerSearchRepFmt"`.
4. In BaseSampleWeb project, browse to folder **JavaResources\depositServerOp.invoker.java**, and open the file **depositServerOpOP.properties** to edit.
   a. Change the last line from `"isLocal=true" to "isLocal=false"`.
   b. Append a new line: `"csReplyFormat= depositRepFmt"`.
5. In BaseSampleWeb project, browse to the folder **JavaResources\withdrawalServerOp.invoker.java**, and open the file **withdrawalServerOpOP.properties** to edit.
   a. Change the last line from `"isLocal=true" to "isLocal=false"`.
   b. Append a new line: `"csReplyFormat= withdrawalRepFmt"`.

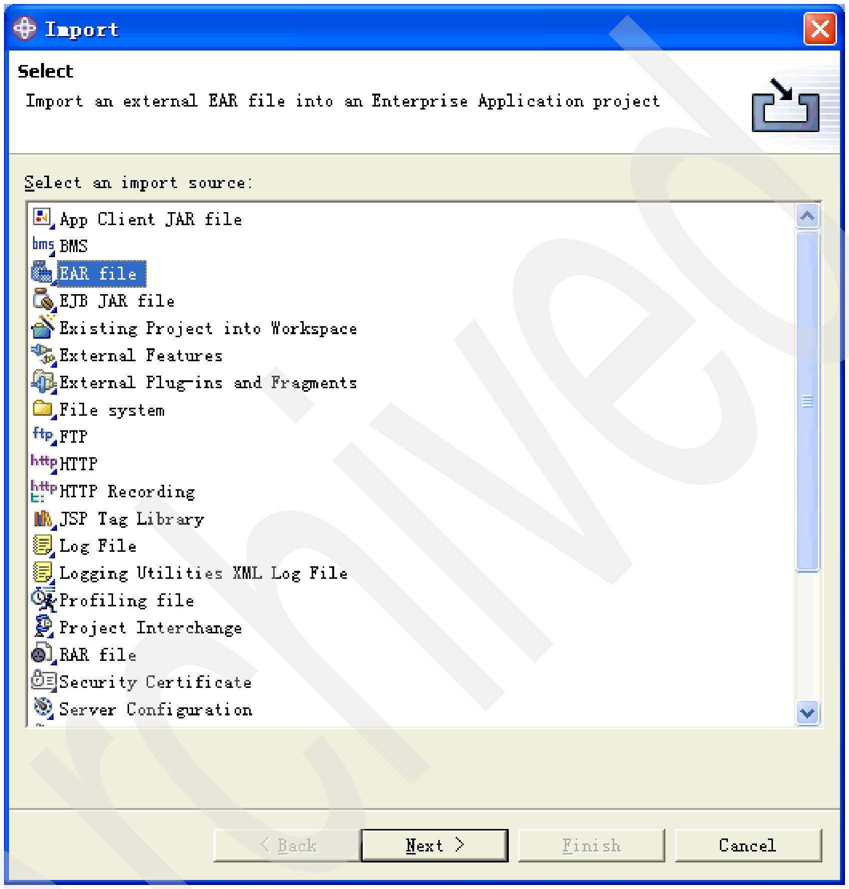
6.2 Importing the required EARs for version 5.1

This section describes all the EAR files that must be imported into our application to complete the migration.

6.2.1 Importing the BTTFormatter.ear

To import the BTTFOrmatter.ear file, follow these steps:

1. From the main menu, select **File** → **Import...**, select the EAR File and browse to the installation package path, for example, C:\IBM\WebSphere Studio\ Branch Transformation Toolkit 5.1\ear. Choose the file **BTTFormatter.ear** and click **Finish**. See Figure 6-4 and Figure 6-5 on page 126.


Import

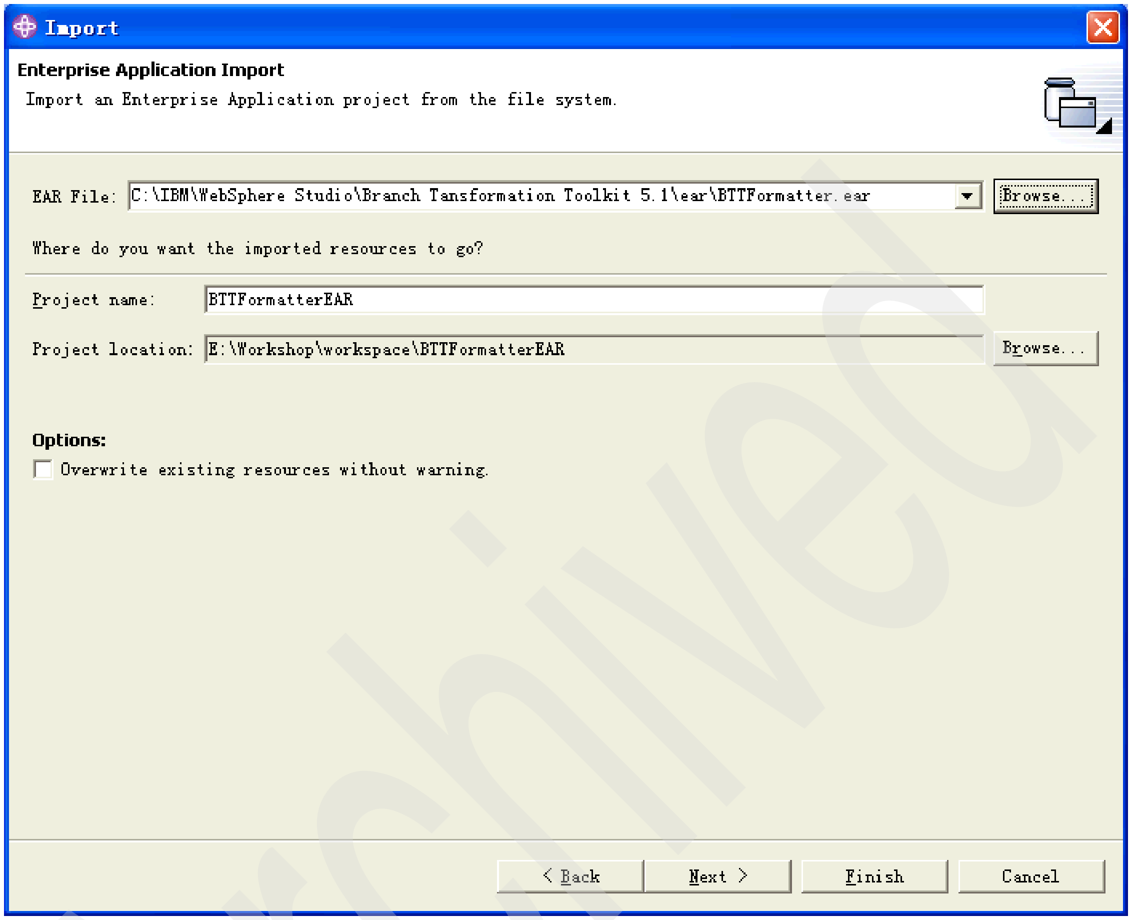
Select

Import an external EAR file into an Enterprise Application project

Select an import source:

App Client JAR file
BMS
EAR file
EJB JAR file
Existing Project into Workspace
External Features
External Plug-ins and Fragments
File system
FTP
HTTP
HTTP Recording
JSP Tag Library
Log File
Logging Utilities XML Log File
Profiling file
Project Interchange
RAR file
Security Certificate
Server Configuration

Figure 6-4 Import EAR file


Import

Enterprise Application Import

Import an Enterprise Application project from the file system.

EAR File: C:\IBM\WebSphere Studio\Branch Tansformation Toolkit 5.1\ear\BTTFormatter.ear

Where do you want the imported resources to go?

Project name: BTTFormatterEAR

Project location: E:\Workshop\workspace\BTTFormatterEAR

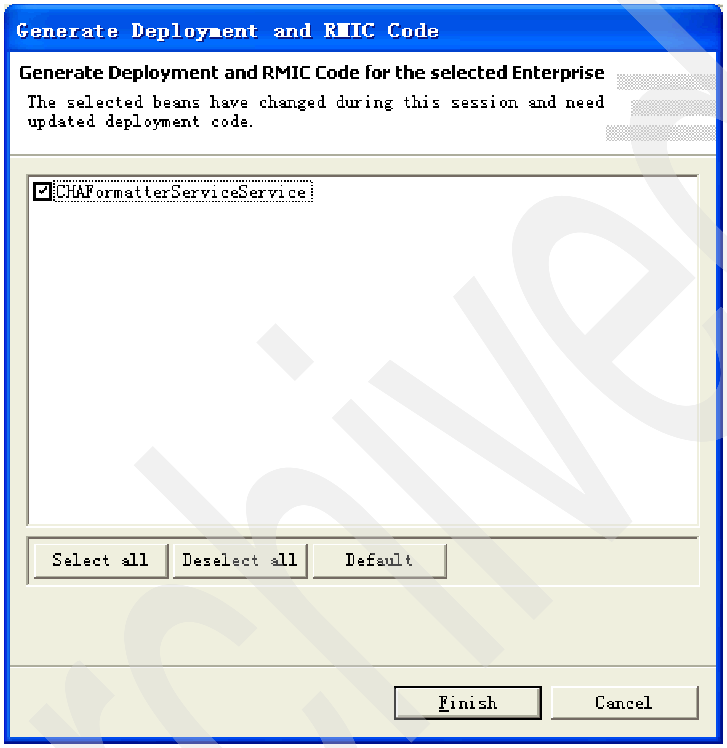
Options:

☐ Overwrite existing resources without warning.

Figure 6-5 Import BTTFormatter.ear

2. Deploy the EJB code.

   In the J2EE perspective, notice that a few BTTFormatter-related projects are created. Right-click the **BTTFormatterEJB** project, and select **Generate → Deployment and RMIC Code...** in the pop-up menu. In the next window, click **Select All**, and then click **Finish**. See Figure 6-6 on page 127.


Generate Deployment and RMIC Code

Generate Deployment and RMIC Code for the selected Enterprise

The selected beans have changed during this session and need updated deployment code.

[✓] CHAFormatterServiceService

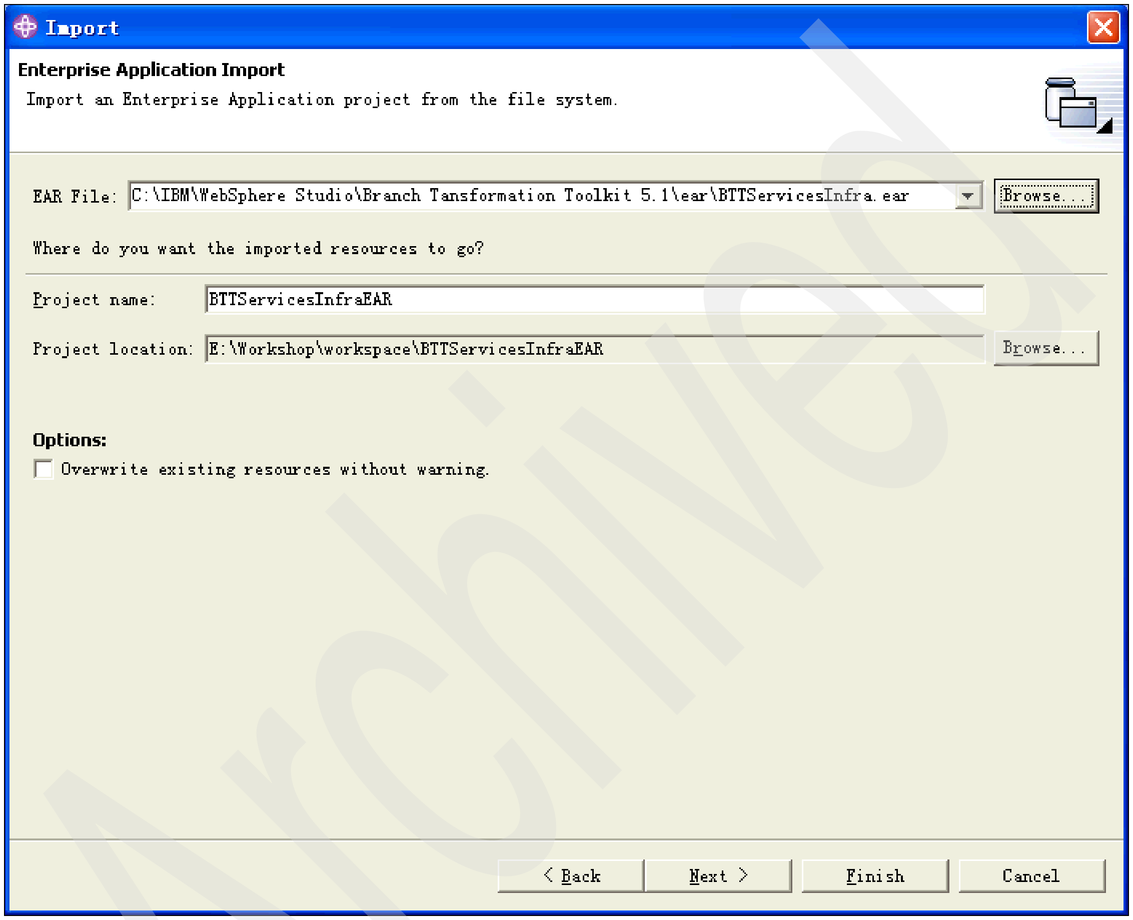
Figure 6-6 Generate EJB deployment code

Similarly, three more EAR files are imported in the next sections.

6.2.2 Importing BTTServicesInfra.ear

To import BTTServiceInfra.ear, follow these steps:

1. From the main menu, select **File** → **Import...**, select the EAR File and browse to the installation packaging path such as C:\IBM\WebSphere Studio\ Branch Transformation Toolkit 5.1\ear, choose the file **BTTServicesInfra.ear**, and click **Finish**. See Figure 6-7.



Enterprise Application Import

Import an Enterprise Application project from the file system.

EAR File: C:\IBM\WebSphere Studio\Branch Tansformation Toolkit 5.1\ear\BTTServicesInfra.ear

Where do you want the imported resources to go?

Project name: BTTServicesInfraEAR

Project location: E:\Workshop\workspace\BTTServicesInfraEAR

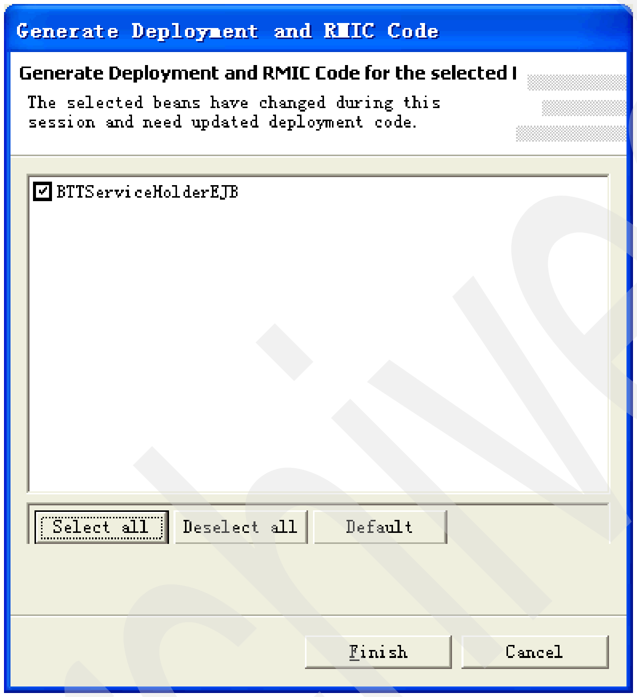
Options:

- [ ] Overwrite existing resources without warning.

Figure 6-7 Import BTTServicesInfra.ear

2. Deploy the EJB code.

   In the J2EE perspective, notice that a few BTTFormatter-related projects are created. Right-click the **bttsvcinfra** project, and select **Generate** → **Deployment and RMIC Code...** in the pop-up menu. In the next window, click **Select All**, and then click **Finish**. See Figure 6-8 on page 129.


Generate Deployment and RMIC Code

Generate Deployment and RMIC Code for the selected I

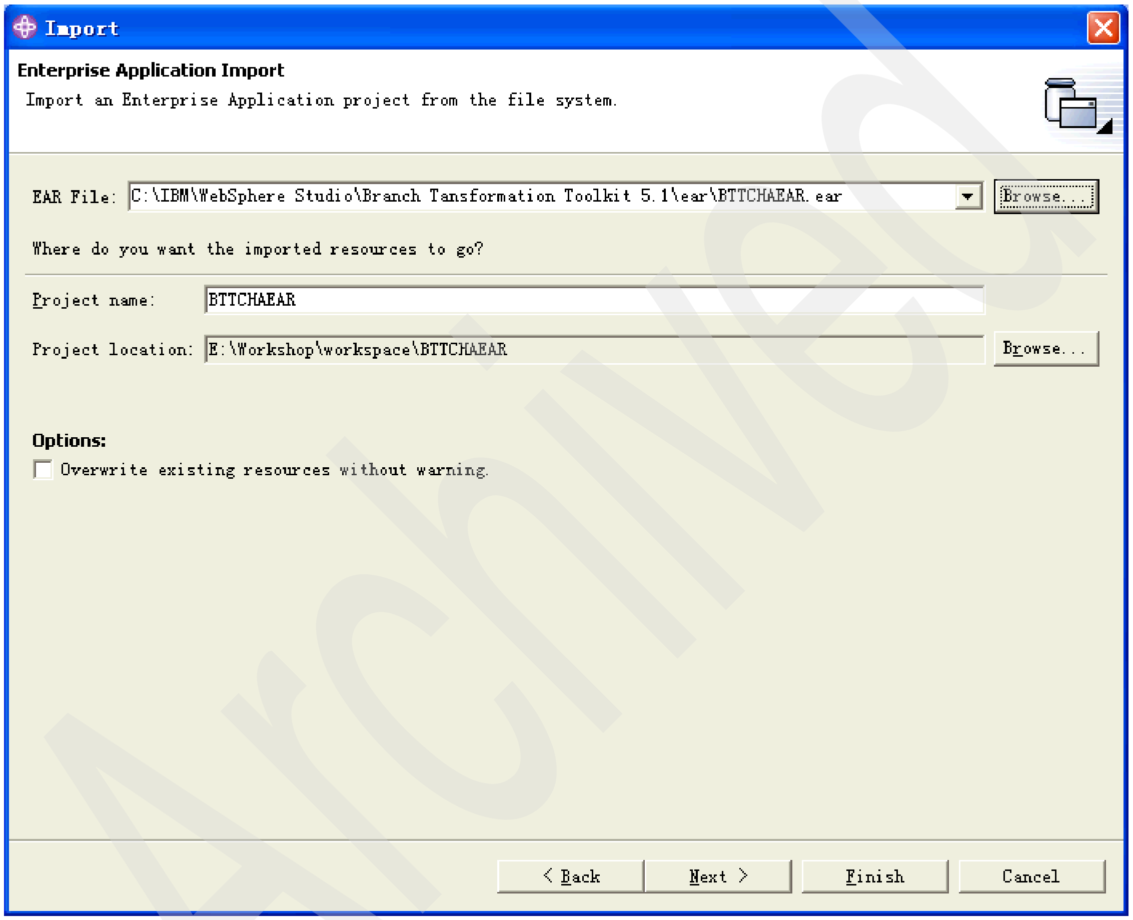
The selected beans have changed during this session and need updated deployment code.

☑ BTTServiceHolderEJB

Figure 6-8 Generate EJB deploy code for BTTServicesInfra.ear

6.2.3 Importing BTTCHAEAR.ear

From the main menu, select **File** → **Import...**, select the EAR File and browse to the installation packaging path such as C:\IBM\WebSphere Studio\ Branch Transformation Toolkit 5.1\ear, choose the file **BTTCHAEAR.ear**, and click **Finish**. See Figure 6-9.



Enterprise Application Import

Import an Enterprise Application project from the file system.

EAR File: C:\IBM\WebSphere Studio\Branch Tansformation Toolkit 5.1\ear\BTTCHAEAR.ear

Where do you want the imported resources to go?

Project name: BTTCHAEAR

Project location: E:\Workshop\workspace\BTTCHAEAR

Options:

☐ Overwrite existing resources without warning.

Figure 6-9 Import BTTCHAEAR.ear,

6.2.4 Importing dummysnalu0.rar

From the main menu, select **File** → **Import...**, select the RAR File and browse to the installation packaging path such as C:\IBM\WebSphere Studio\ Branch Transformation Toolkit 5.1\jars, choose the file **dummysnalu0.rar**. Make sure the "Standalone connector project" option is checked, and click **Finish**. See Figure 6-10 and Figure 6-11 on page 132.

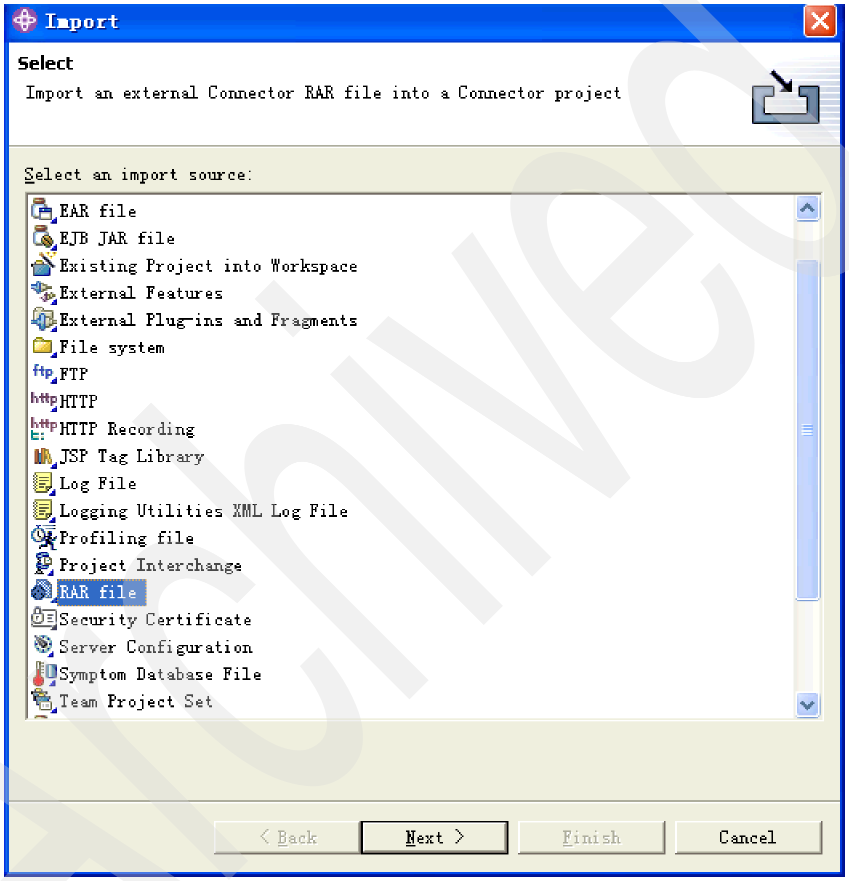
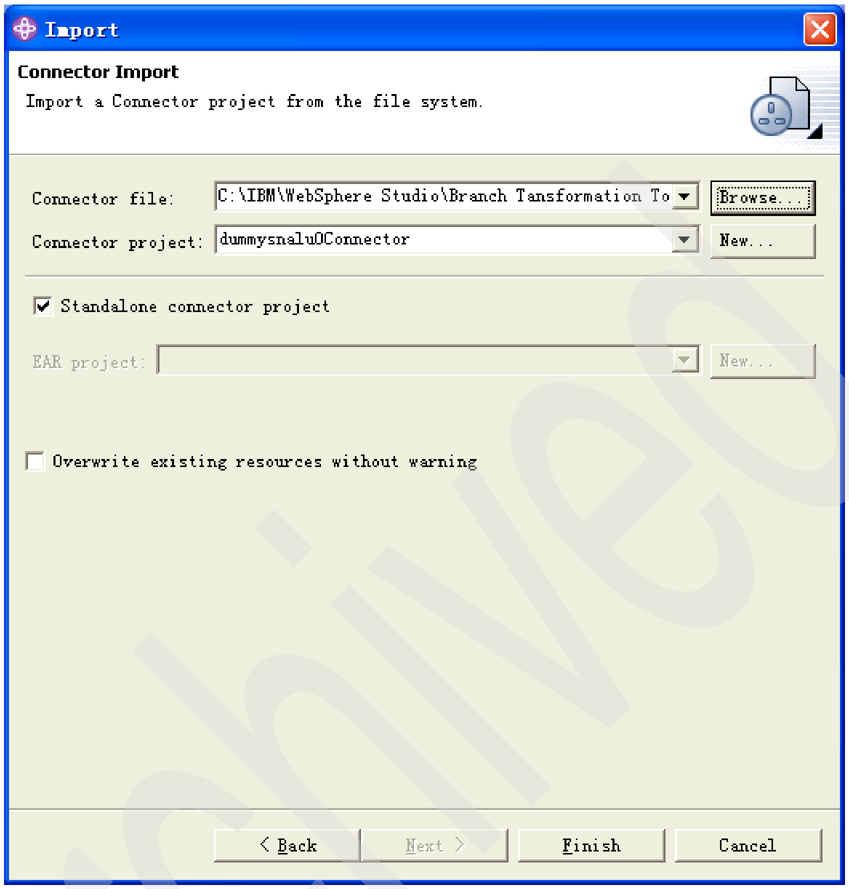


Figure 6-10 Import connector



Connector file: C:\IBM\WebSphere Studio\Branch Tansformation To

Connector project: dummysnalu0Connector

[✓] Standalone connector project

EAR project:

[ ] Overwrite existing resources without warning

Figure 6-11 Import dummysnalu0.rar

6.3 Preparing the client and the server

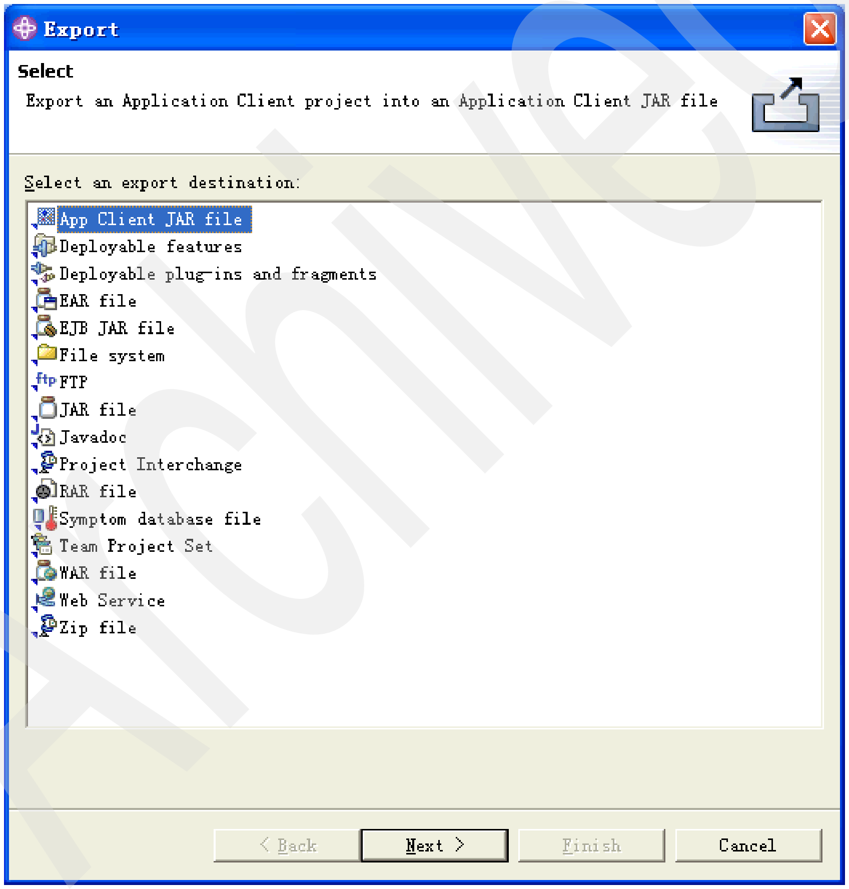
In this phase, you should create the application client and server for use by the Branch Transformation Toolkit 5.1 version of the application.

6.3.1 Creating the client

To create the client, follow these steps:

1. Export the application client from the Branch Transformation Toolkit 4.3 workspace.

   Open the Branch Transformation Toolkit 4.3 workspace of WebSphere Studio Application Developer Integration Edition. In this workspace, right-click **DSE_SampleApplicationClient** project. From the pop-up menu, choose **Export...** For the type, choose **App Client JAR file**. Click **Next**. In the next window, choose where to save, for example, C:\temp, and check **Export source files**. Click **Finish**. See Figure 6-12 and Figure 6-13 on page 134


Export

Select

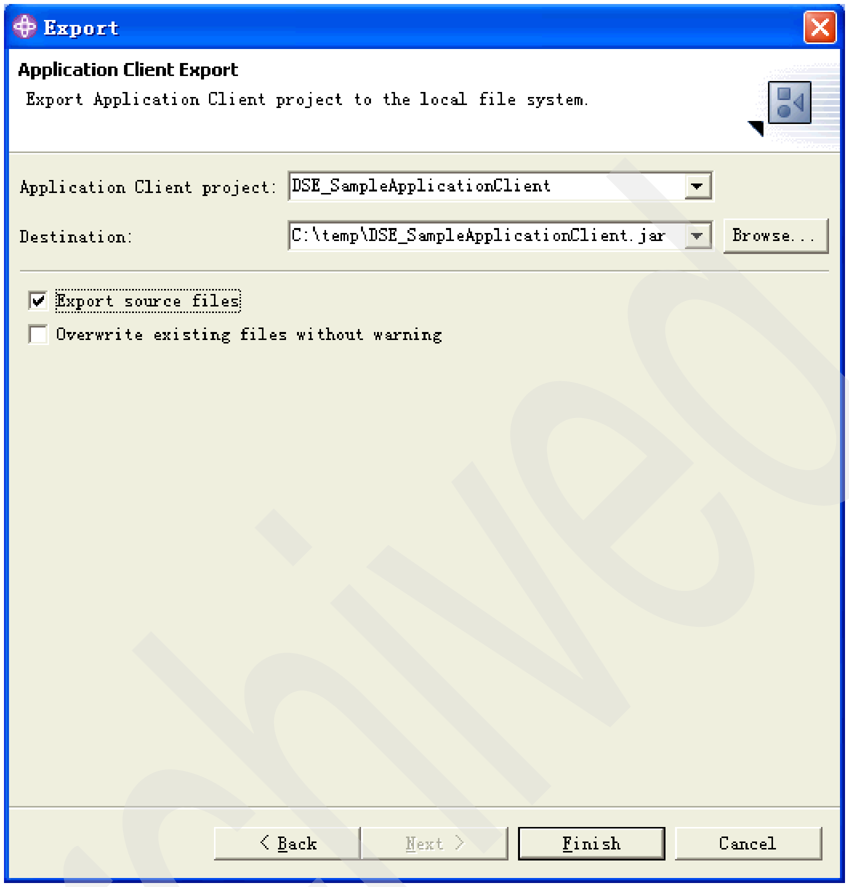
Export an Application Client project into an Application Client JAR file

Select an export destination:

- App Client JAR file
- Deployable features
- Deployable plug-ins and fragments
- EAR file
- EJB JAR file
- File system
- FTP
- JAR file
- Javadoc
- Project Interchange
- RAR file
- Symptom database file
- Team Project Set
- WAR file
- Web Service
- Zip file

< Back | Next > | Finish | Cancel

Figure 6-12 Exporting the application client


Export
Application Client Export
Export Application Client project to the local file system.
Browse...
< Back
Next >
Finish
Cancel

Application Client project: DSE_SampleApplicationClient

Destination: C:\temp\DSE_SampleApplicationClient.jar

[✓] Export source files

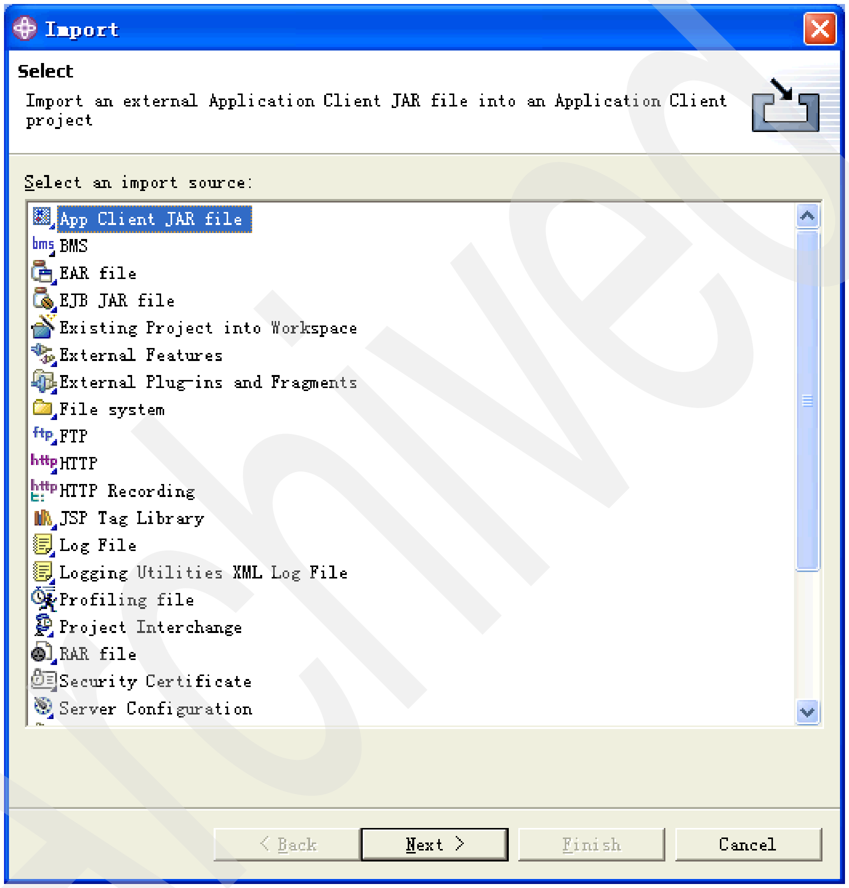
[ ] Overwrite existing files without warning

Figure 6-13 Export application client source files

Note: Do not close the Branch Transformation Toolkit 4.3 workspace of WebSphere Studio Application Developer Integration Edition. It will be used later.

2. Import the application client into the migration workspace.

   In the Branch Transformation Toolkit 5.1 migration workspace, from the main menu, select **File** → **Import...**, choose **App Client JAR file**, and click **Next**. In the next window, browse to the **DSE_SampleApplicationClient.jar** file, and from the EAR project drop-down list, select **BaseSample**. Click **Finish**. See Figure 6-14 and Figure 6-15 on page 136.



Import

Select

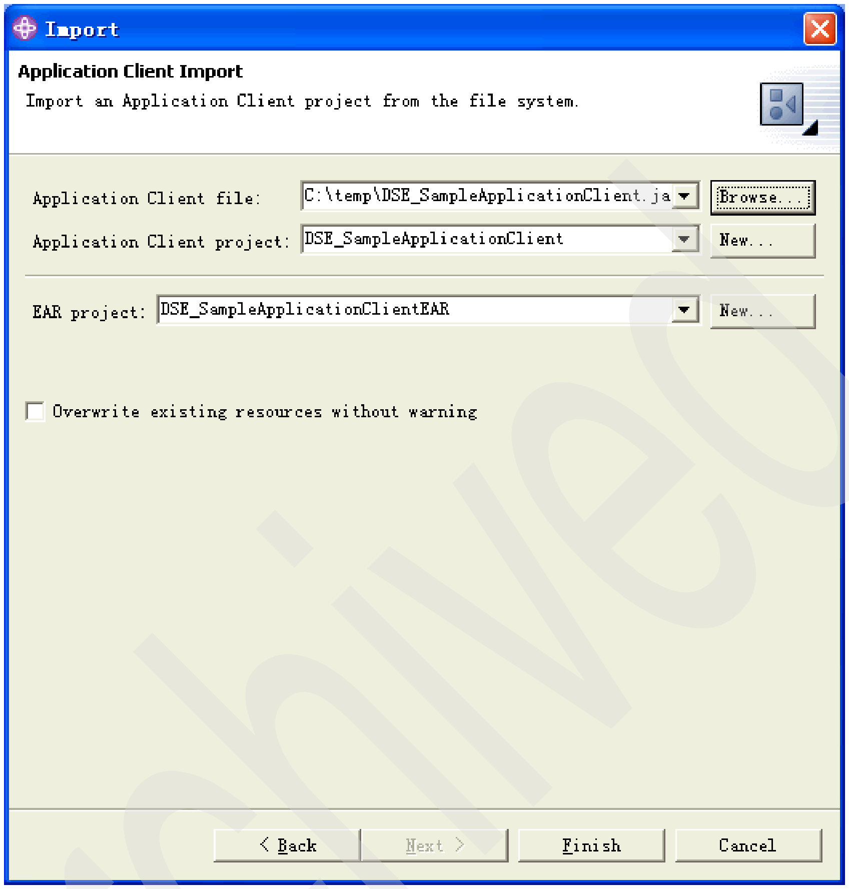
Import an external Application Client JAR file into an Application Client project

Select an import source:

App Client JAR file
BMS
EAR file
EJB JAR file
Existing Project into Workspace
External Features
External Plug-ins and Fragments
File system
FTP
HTTP
HTTP Recording
JSP Tag Library
Log File
Logging Utilities XML Log File
Profiling file
Project Interchange
RAR file
Security Certificate
Server Configuration

< Back | Next > | Finish | Cancel

Figure 6-14 Import application client


Import
Application Client Import
Import an Application Client project from the file system.

Application Client file: C:\temp\DSE_SampleApplicationClient.ja

Application Client project: DSE_SampleApplicationClient

EAR project: DSE_SampleApplicationClientEAR

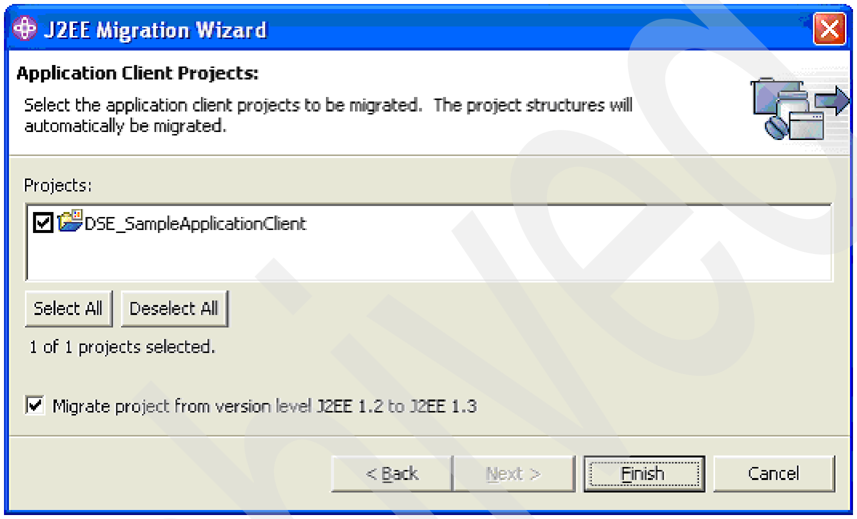
☐ Overwrite existing resources without warning

Browse...
New...
New...
< Back
Next >
Finish
Cancel

Figure 6-15 Import application client from the file system

3. Migrate the sample client to a Branch Transformation Toolkit 5.1 client project.

   Right-click the **DSE_SampleApplicationClient** project in the Branch Transformation Toolkit 5.1 migration workspace. In the pop-up menu, select **Migrate** → **J2EE Migration Wizard...**, read the information, and click **Next**. In the next window, make sure the DSE_SampleApplicationClient project is selected, and click **Finish**. See Figure 6-16


J2EE Migration Wizard

Application Client Projects:

Select the application client projects to be migrated. The project structures will automatically be migrated.

Projects:

☑ DSE_SampleApplicationClient

1 of 1 projects selected.

☑ Migrate project from version level J2EE 1.2 to J2EE 1.3

Figure 6-16 Migrate sample client project

6.3.2 Copying client definition files

Copy the Branch Transformation Toolkit 4.3 application client definition files to the migration project for future use.

1. In the J2EE perspective of the migration project, expand the **BaseSampleWeb** project. Under the WebContent folder, create a new folder named `dse`. Within the dse folder, create two other folders, `client` and `desktop`. The resulting file structure should look as displayed in Figure 6-17:

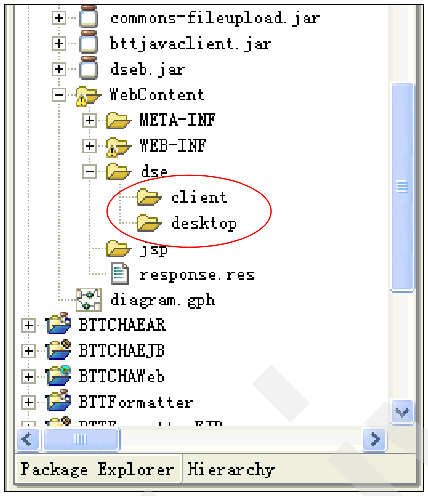


Figure 6-17 File structure for client project

2. In the J2EE perspective of the Branch Transformation Toolkit 4.3 project, expand the **DSE_SampleApplicationWeb** project. Copy the following files from DSE_SampleApplication\DSE_SampleApplicationWeb\WebContent\dse\ client to BaseSampleWeb\WebContent\dse\client of the migration workspace:
   - **dsectxt.xml**
   - **dsedata.xml**
   - **dsefmts.xml**
   - **dseoper.xml**
   - **dsesrvce.xml**
   - **dsetype.xml**

The resulting file structure should look as shown in Figure 6-18:

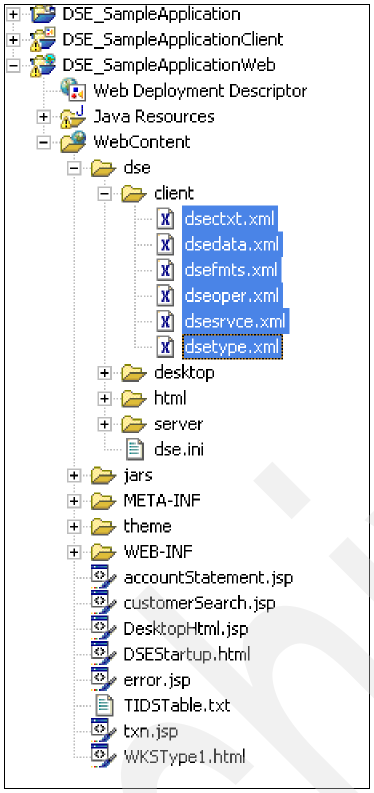

```
DSE_SampleApplication
DSE_SampleApplicationClient
DSE_SampleApplicationWeb
    Web Deployment Descriptor
    Java Resources
    WebContent
        dse
            client
                dsectxt.xml
                dsedata.xml
                dsefmts.xml
                dseoper.xml
                dsesrvce.xml
                dsetype.xml
            desktop
            html
            server
            dse.ini
        jars
        META-INF
        theme
        WEB-INF
        accountStatement.jsp
        customerSearch.jsp
        DesktopHtml.jsp
        DSEStartup.html
        error.jsp
        TIDSTable.txt
        txn.jsp
        WKSType1.html
```

Figure 6-18 Files for DSE client

3. In the J2EE perspective of the Branch Transformation Toolkit 4.3 project, expand the **DSE_SampleApplicationWeb** project. Copy the following files from DSE_SampleApplication\DSE_SampleApplicationWeb\WebContent\dse\desktop to BaseSampleWeb\WebContent\dse\desktop of the migration workspace:

   – **desktop.dtd**
   – **desktop.xml**

The resulting file structure should look as shown in Figure 6-19:

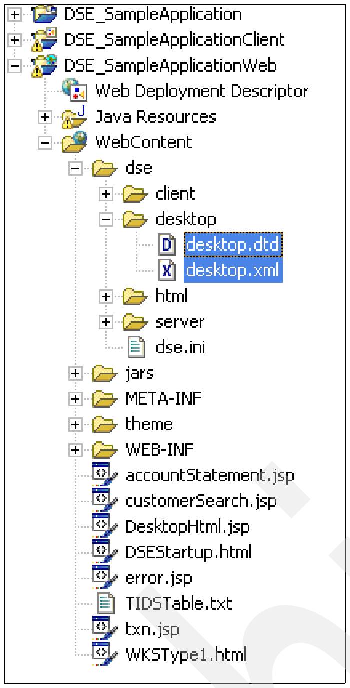

DSE_SampleApplication
DSE_SampleApplicationClient
DSE_SampleApplicationWeb
Web Deployment Descriptor
Java Resources
WebContent
dse
client
desktop
desktop.dtd
desktop.xml
html
server
dse.ini
jars
META-INF
theme
WEB-INF
accountStatement.jsp
customerSearch.jsp
DesktopHtml.jsp
DSEStartup.html
error.jsp
TIDSTable.txt
txn.jsp
WKSType1.html

Figure 6-19 Desktop files

4. In the J2EE perspective of the Branch Transformation Toolkit 4.3 project, expand the **DSE_SampleApplicationWeb** project. Copy the file **dse.ini** from DSE_SampleApplication\DSE_SampleApplicationWeb\WebContent\dse to BaseSampleWeb\WebContent\dse of the migration workspace.

The file structure should look as shown in Figure 6-20:

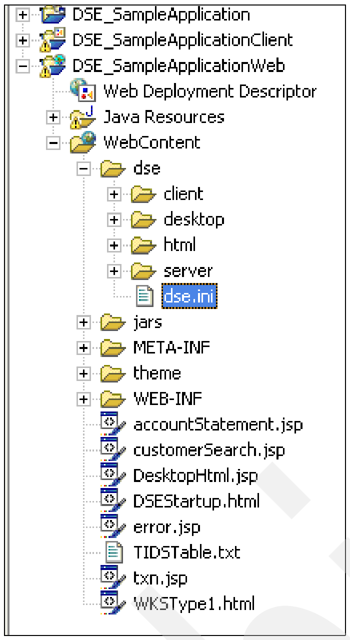


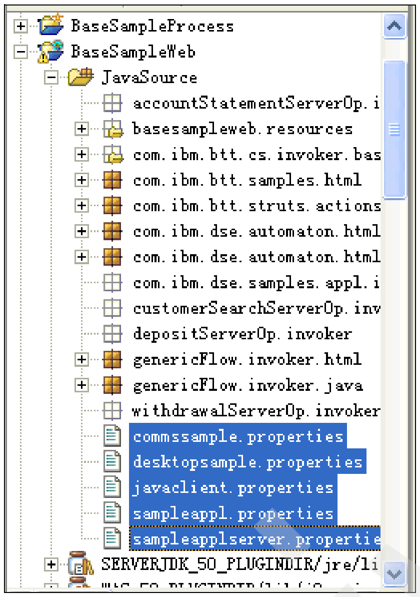
Figure 6-20 File location for dse.ini

5. In the J2EE perspective of the Branch Transformation Toolkit 4.3 project, expand the **DSE_SampleApplicationWeb** project. Copy the following files from Branch Transformation Toolkit 5.1 installation packaging's path such as C:\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1\ samples\JavaSampleApplication\StandAlone\ BTTJavaSample.ear to BaseSampleWeb\Java Resources of the migration workspace:

   – **commssample.properties**
   – **desktopsample.properties**
   – **javaclient.properties**
   – **sampleappl.properties**
   – **sampleappserver.properties**

   Note: Extract **BTTJavaSampleClient.jar** in BTTJavaSample.ear to get the properties files.

The resulting file structure should look as shown in Figure 6-21 on page 142.



```
BaseSampleProcess
BaseSampleWeb
  JavaSource
    accountStatementServerOp.i
    basesampleweb.resources
    com.ibm.btt.cs.invoker.bas
    com.ibm.btt.samples.html
    com.ibm.btt.struts.actions
    com.ibm.dse.automaton.html
    com.ibm.dse.automaton.html
    com.ibm.dse.samples.appl.i
    customerSearchServerOp.inv
    depositServerOp.invoker
    genericFlow.invoker.html
    genericFlow.invoker.java
    withdrawalServerOp.invoker
    commssample.properties
    desktopsample.properties
    javaclient.properties
    sampleappl.properties
    sampleapplserver.propertie
  SERVERJDK_50_PLUGINDIR/jre/li
```

Figure 6-21 Properties file location

6. Modify the definitions in the dse.ini file.

   Since Branch Transformation Toolkit 5.1 sever client structure is different from Branch Transformation Toolkit 4.3, some definitions have to be changed.

   Open **BaseSampleWeb\WebContent\dse**, and in the dse.ini file, make changes in the Field elements according to that shown in Table 6-1.

Table 6-1 Field elements in the des.ini

Field ID	Old Value	New Value
CsAssignServletName	/servlet/com.ibm.dse.cs.servlet.CSAssignServiceIdAndServerTIDProtocolServlet	/servlet/com.ibm.btt.cs.servlet.CSAssignServiceIdAndServerTIDProtocolServlet
csReqProtocolServletName	/servlet/com.ibm.dse.cs.servlet.CSReqProtocolServlet	/servlet/com.ibm.btt.cs.servlet.CSReqProtocolServlet
csNotifClToSrvServletName	/servlet/com.ibm.dse.cs.servlet.CSNotifClToSrvProtocolServlet	/servlet/com.ibm.btt.event.CSNotifClToSrvProtocolServlet

Field ID	Old Value	New Value
csNotifSrvToClServletName	/servlet/com.ibm.dse.cs.servlet.CSNotifSrvToClProtocolServlet	/servlet/com.ibm.btt.event.CSNotifSrvToClProtocolServlet

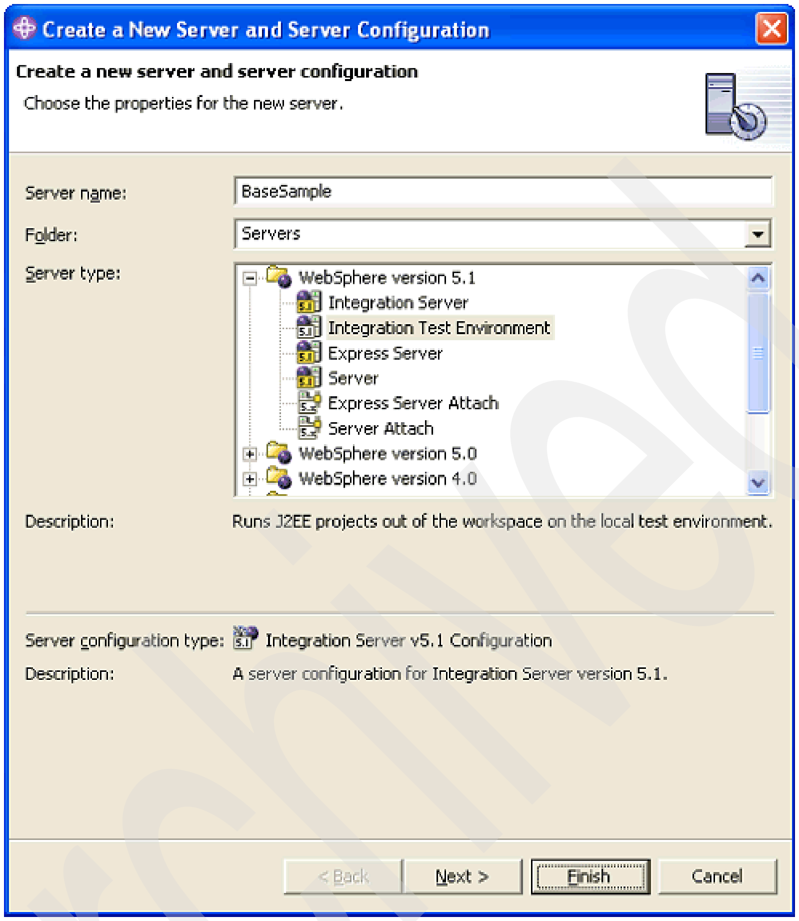
6.3.3 Creating a server

To create a server environment to test your migrated project, you should create a server and its configuration, and then deploy the CHA EJB code.

Server and configuration

To create the server and its configuration, follow these steps:

1. In the migration workspace, change to the Server perspective.
2. From the main menu, select **File** → **New** → **Server and Server Configuration**. In the next window, input `BaseSample` for the Server name, and select **Integration Test Environment** for the Server type. Click **Finish**. See Figure 6-22 on page 144.


Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: BaseSample

Folder: Servers

Server type:

- WebSphere version 5.1
  - Integration Server
  - Integration Test Environment
  - Express Server
  - Server
  - Express Server Attach
  - Server Attach
- WebSphere version 5.0
- WebSphere version 4.0

Description: Runs J2EE projects out of the workspace on the local test environment.

Server configuration type: Integration Server v5.1 Configuration

Description: A server configuration for Integration Server version 5.1.

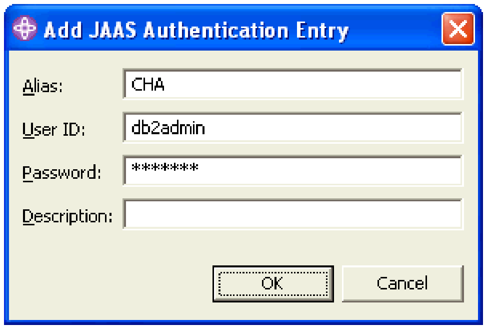
Figure 6-22 Creat a new server

3. Double-click **BaseSample** server in the Server panel to configure.
4. In the Security tab, click **Add** to insert two JAAS authentication entries as shown in Table 6-2.

Table 6-2 JAAS authentication entries

Alias	User ID	Password
CHA	DB2_username	DB2_password
sna	sna	sna

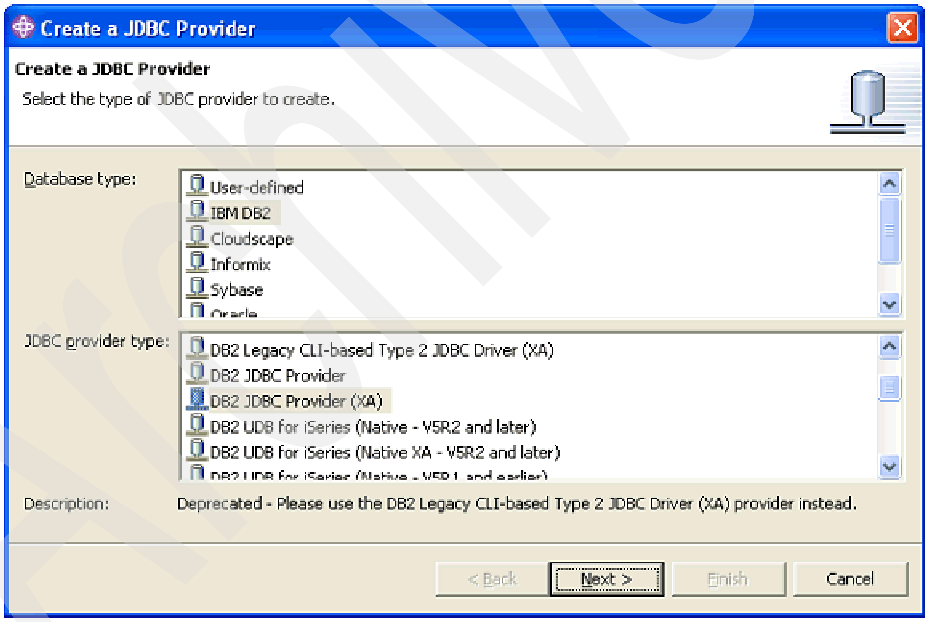
Figure 6-23 shows the dialog used to create a JAAS authentication entry.



Alias: CHA
User ID: db2admin
Password: ********
Description:

Figure 6-23 Add JAAS authentication

5. In the Data Source tab, click **Add** to insert an item in the JDBC provider list. For Database type, select **IBM DB2**, and for JDBC provider type, select **DB2 JDBC Provider (XA)**, and click **Next**. See Figure 6-24.



Create a JDBC Provider

Select the type of JDBC provider to create.

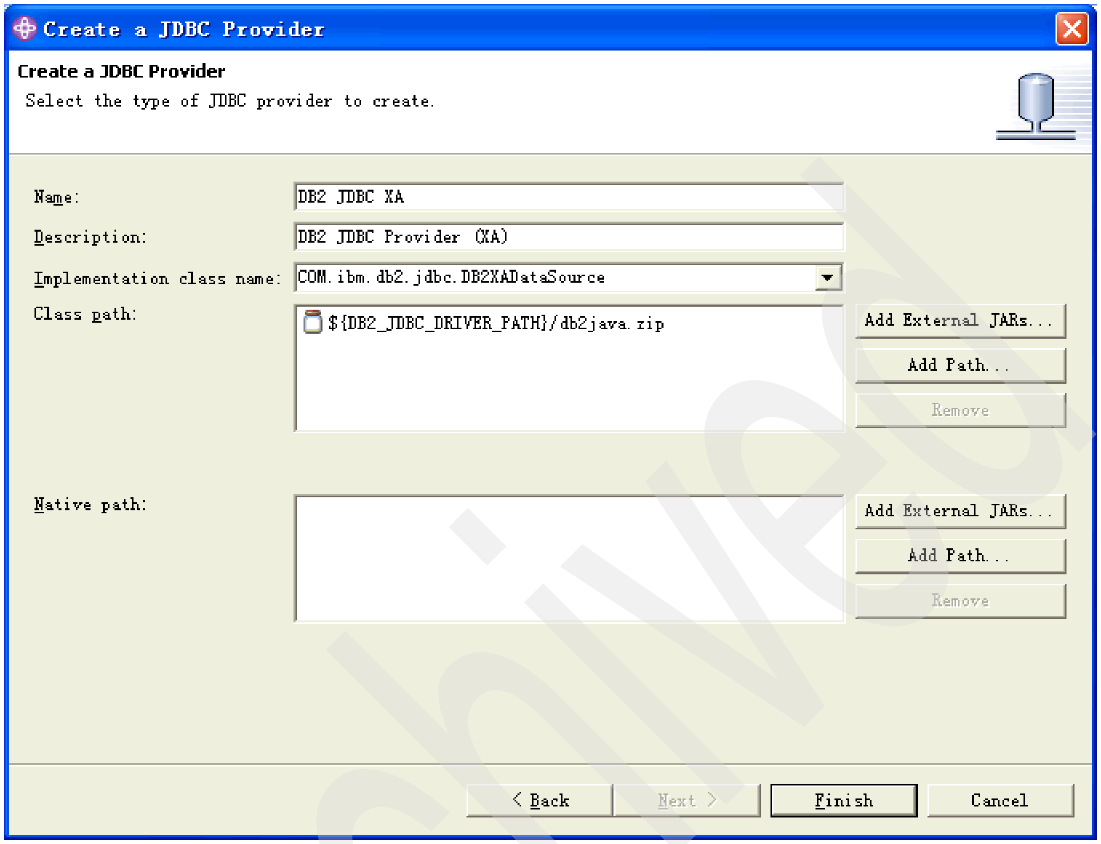
Database type: User-defined, IBM DB2 (selected), Cloudscape, Informix, Sybase, Oracle

JDBC provider type: DB2 Legacy CLI-based Type 2 JDBC Driver (XA), DB2 JDBC Provider, DB2 JDBC Provider (XA) (selected), DB2 UDB for iSeries (Native - V5R2 and later), DB2 UDB for iSeries (Native XA - V5R2 and later), DB2 UDB for iSeries (Native - V5R1 and earlier)

Description: Deprecated - Please use the DB2 Legacy CLI-based Type 2 JDBC Driver (XA) provider instead.

Figure 6-24 Create a JDBC provider

6. In the next window, eneter **`DB2 JDBC XA`** for the name. Click **Finish**. See Figure 6-25 on page 146.


Create a JDBC Provider

Create a JDBC Provider

Select the type of JDBC provider to create.

Name: DB2 JDBC XA

Description: DB2 JDBC Provider (XA)

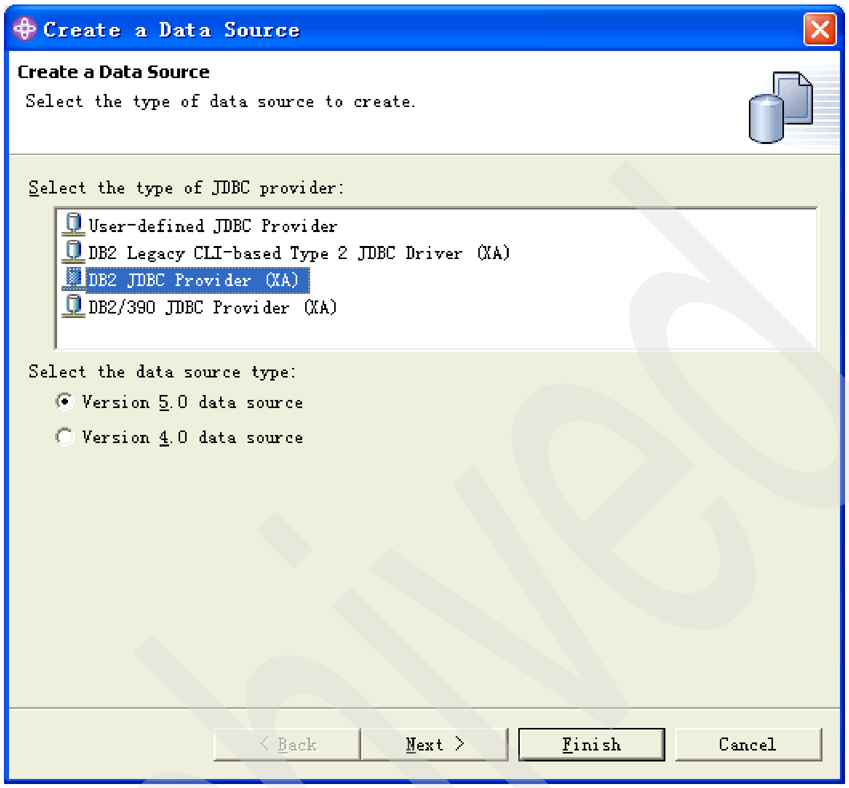
Implementation class name: COM.ibm.db2.jdbc.DB2XADataSource

Class path: ${DB2_JDBC_DRIVER_PATH}/db2java.zip

Native path:

Figure 6-25 JDBC provider details

7. In the Data Source tab, select **DB2 JDBC XA** from the JDBC provider list, and click **Add** to create a data source. In the pop-up window, select **DB2 JDBC Provider (XA)** for the type of JDBC provider, and select **version 5.0 data source** for the data source type. Click **Next**. See Figure 6-26 on page 147.


Create a Data Source

Create a Data Source

Select the type of data source to create.

Select the type of JDBC provider:

- User-defined JDBC Provider
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
- DB2 JDBC Provider (XA)
- DB2/390 JDBC Provider (XA)

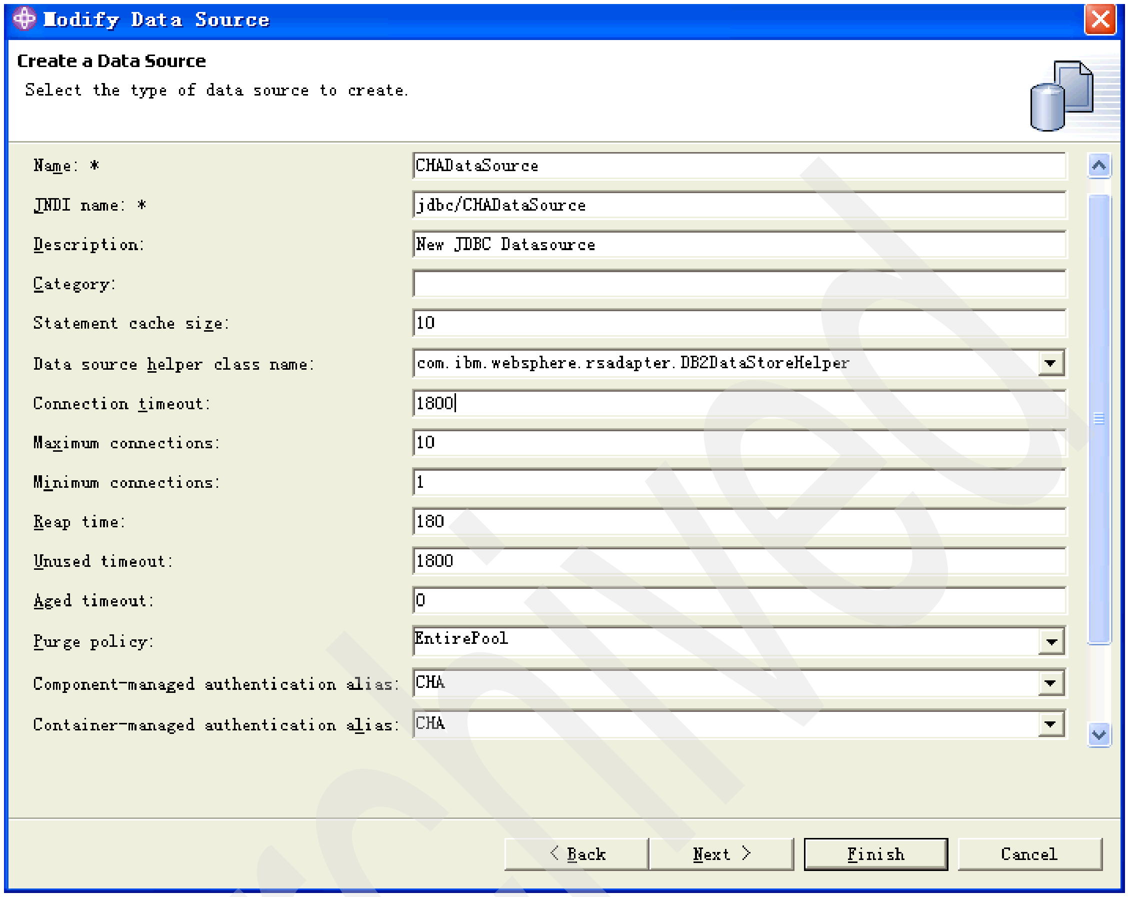
Select the data source type:

[✓ Version 5.0 data source]

Version 4.0 data source

Figure 6-26 Create a DataSource

8. In the next window, input `CHADataSource` as the name, `jdbc/CHADataSource` as the JNDI name, and from the drop-down list, choose **CHA** for both component-managed and container-managed authentication alias. Leave the remaining values as default. Click **Finish**. See Figure 6-27 on page 148.


Modify Data Source

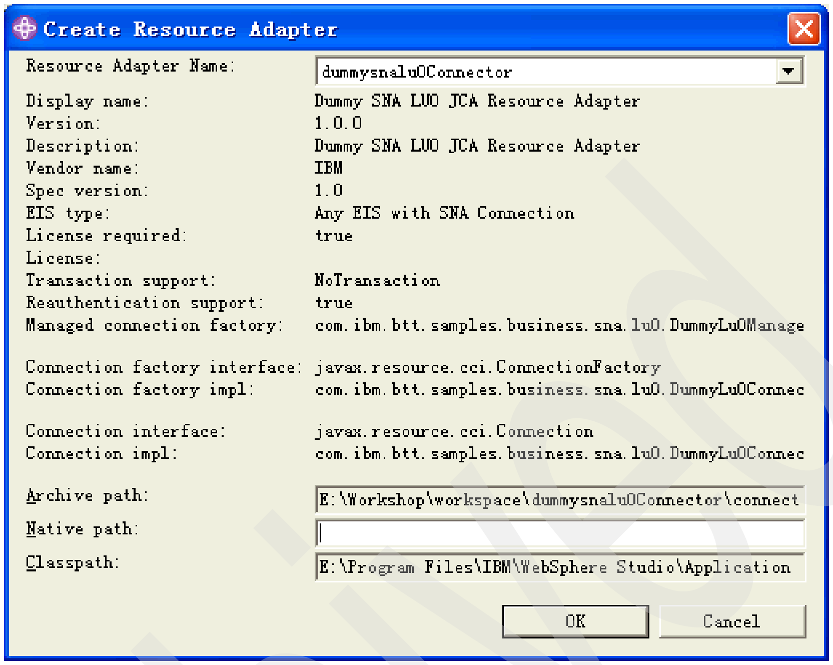
Create a Data Source

Select the type of data source to create.

Name: *	CHADataSource
JNDI name: *	jdbc/CHADataSource
Description:	New JDBC Datasource
Category:	
Statement cache size:	10
Data source helper class name:	com.ibm.websphere.rsadapter.DB2DataStoreHelper
Connection timeout:	1800
Maximum connections:	10
Minimum connections:	1
Reap time:	180
Unused timeout:	1800
Aged timeout:	0
Purge policy:	EntirePool
Component-managed authentication alias:	CHA
Container-managed authentication alias:	CHA

Figure 6-27   DataSource details

9. In the J2C tab, click **Add** to insert an item in the J2C Resource Adapters. Ensure that dummysnalu0Connector is displayed as the Resource Adapter Name, and click **OK**. See Figure 6-28 on page 149.

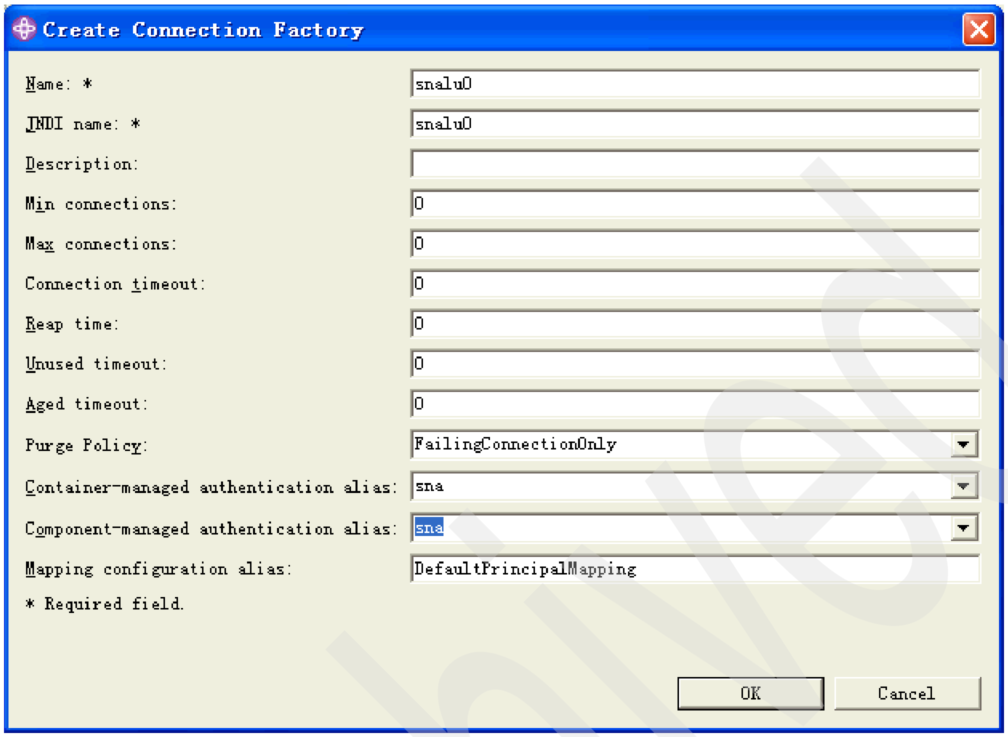


Create Resource Adapter

Resource Adapter Name:	dummysnalu0Connector
Display name:	Dummy SNA LU0 JCA Resource Adapter
Version:	1.0.0
Description:	Dummy SNA LU0 JCA Resource Adapter
Vendor name:	IBM
Spec version:	1.0
EIS type:	Any EIS with SNA Connection
License required:	true
License:	
Transaction support:	NoTransaction
Reauthentication support:	true
Managed connection factory:	com.ibm.btt.samples.business.sna.lu0.DummyLu0Manage
Connection factory interface:	javax.resource.cci.ConnectionFactory
Connection factory impl:	com.ibm.btt.samples.business.sna.lu0.DummyLu0Connec
Connection interface:	javax.resource.cci.Connection
Connection impl:	com.ibm.btt.samples.business.sna.lu0.DummyLu0Connec
Archive path:	E:\Workshop\workspace\dummysnalu0Connector\connect
Native path:	
Classpath:	E:\Program Files\IBM\WebSphere Studio\Application

Figure 6-28 Create a resource adapter

10. In the J2C tab, select **dummysnalu0Connector** from the J2C Resource Adapters, and click **Add** to create an item in the J2C Connection Factories. In the pop-up window, type `snalu0` as the name and the JNDI name, select **FailingConnectionOnly** as the Purge Policy, and from the drop-down list, select **sna** as container-managed and component-managed authentication aliases. Leave the remaining values as default. Click **OK**. See Figure 6-29 on page 150.



Create Connection Factory

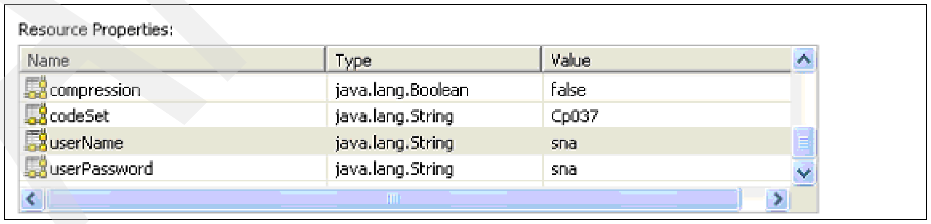
Name: *	snalu0
JNDI name: *	snalu0
Description:	
Min connections:	0
Max connections:	0
Connection timeout:	0
Reap time:	0
Unused timeout:	0
Aged timeout:	0
Purge Policy:	FailingConnectionOnly
Container-managed authentication alias:	sna
Component-managed authentication alias:	sna
Mapping configuration alias:	DefaultPrincipalMapping

* Required field.

Figure 6-29 Create a connection factory

11. In the same tab, change the content of Resource Properties to `dummysnalu0Connector`. Change the value of both userName and userPassword to `sna`, and change the value of TestFile to `http://127.0.0.1:9080/BaseSampleWeb/response.res`.

    See Figure 6-30.



Resource Properties:

Name	Type	Value
compression	java.lang.Boolean	false
codeSet	java.lang.String	Cp037
userName	java.lang.String	sna
userPassword	java.lang.String	sna

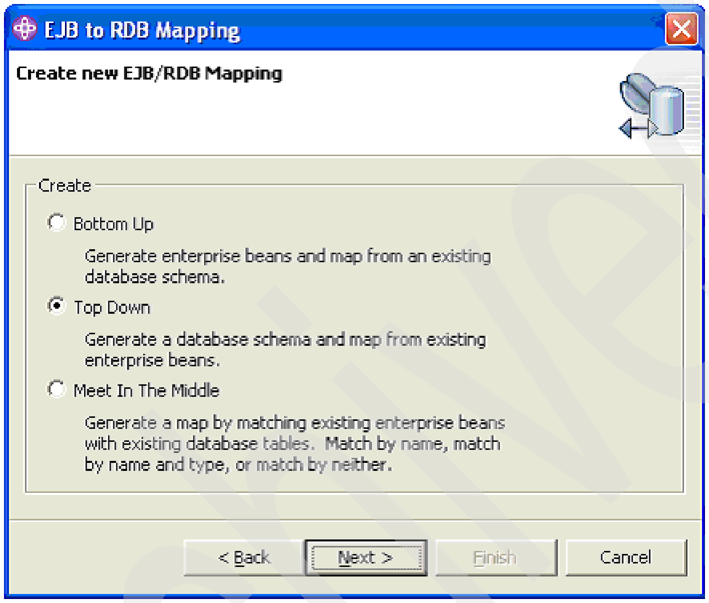
Figure 6-30 Resource properties

12. Save the server configuration and close the **Server Configuration Content Editor**.

CHA deployment

To deploy the CHA, follow these steps:

1. Change the Project Navigator view to J2EE perspective. Right-click the **BTTCHAEJB** project, and from the the pop-up menu, select **Generate** → **EJB to RDB Mapping...**, then click **Next** to create a new backend folder. Select **Top Down** in the next window. Click **Next**. See Figure 6-31



( ) Bottom Up

Generate enterprise beans and map from an existing database schema.

(•) Top Down

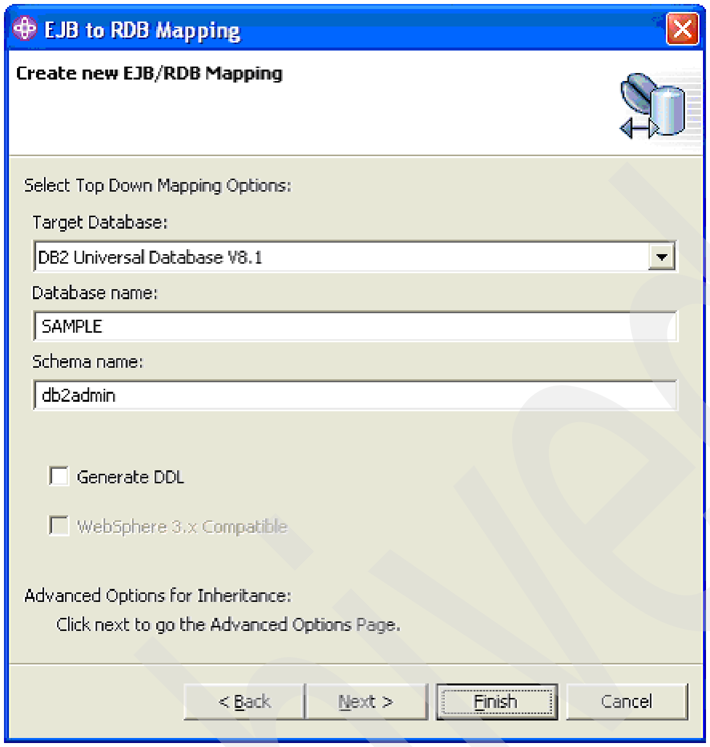
Generate a database schema and map from existing enterprise beans.

( ) Meet In The Middle

Generate a map by matching existing enterprise beans with existing database tables. Match by name, match by name and type, or match by neither.

Figure 6-31 Generate EJB to RDB mapping

In the next window, select the database type and input the schema you installed on your server. Click **Finish**. See Figure 6-32 on page 152.



EJB to RDB Mapping

Create new EJB/RDB Mapping

Select Top Down Mapping Options:

Target Database: DB2 Universal Database V8.1

Database name: SAMPLE

Schema name: db2admin

☐ Generate DDL

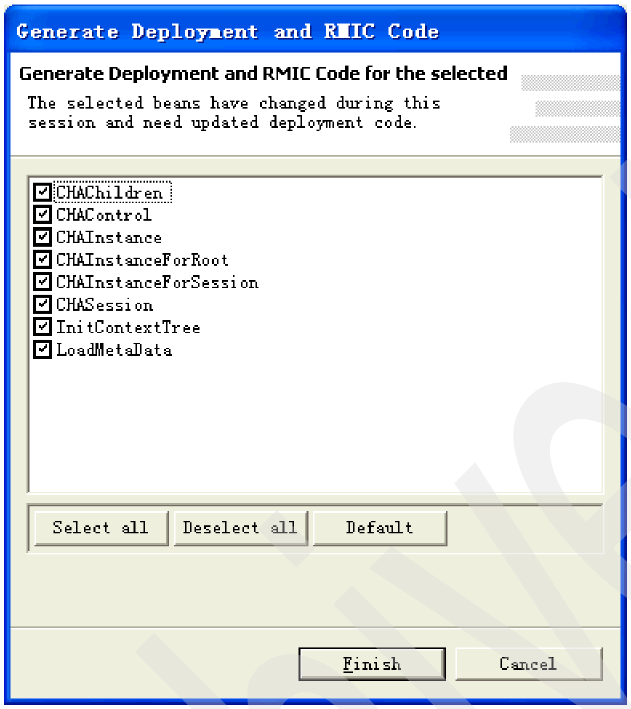
☐ WebSphere 3.x Compatible

Advanced Options for Inheritance:

Click next to go the Advanced Options Page.

Figure 6-32 Mapping details

2. Right-click the **BTTCHAEJB** project. From the pop-up menu select **Generate** → **Deployment and RMIC Code...**. Click **Select All** and **Finish**. See Figure 6-33 on page 153



Generate Deployment and RMIC Code for the selected

The selected beans have changed during this session and need updated deployment code.

- ☑ CHAChildren
- ☑ CHAControl
- ☑ CHAInstance
- ☑ CHAInstanceForRoot
- ☑ CHAInstanceForSession
- ☑ CHASession
- ☑ InitContextTree
- ☑ LoadMetaData

Figure 6-33 Generate deployment code

6.4 Adding and modifying code

In this section we describe some of the code additions and modifications you must make to successfully complete the migration.

6.4.1 Business logic layer

This section describes how the business logic layer needs to be altered in order to complete the migration.

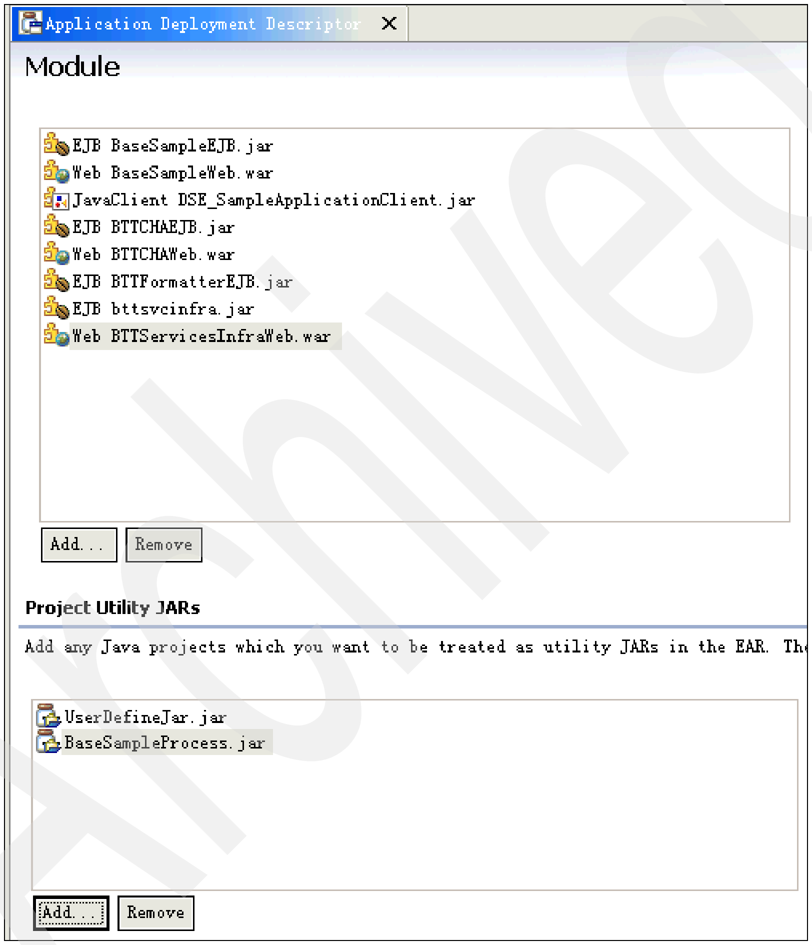
Configuring the business process

To configure the business process, follow these steps:

1. Open the **application.xml** file in the META-INF folder of the BaseSample project in J2EE perspective. In the Module tab, click **Add** to include following modules:
   - **BTTCHAEJB**
   - **BTTCHAWeb**

- **BTTFormatterEJB**
- **bttsvcinfra**
- **BTTServiceInfraWeb**
- **DSE_SampleApplicationClient**

2. Click **Add** in the lower panel to add Project Utility JARs. Select **UserDefineJar** and **BaseSampleProcess** project separately. Save and close the file. See Figure 6-34.



Application Deployment Descriptor

Module

EJB BaseSampleEJB.jar
Web BaseSampleWeb.war
JavaClient DSE_SampleApplicationClient.jar
EJB BTTCHAEJB.jar
Web BTTCHAWeb.war
EJB BTTFormatterEJB.jar
EJB bttsvcinfra.jar
Web BTTServicesInfraWeb.war

Project Utility JARs

Add any Java projects which you want to be treated as utility JARs in the EAR. Th

UserDefineJar.jar
BaseSampleProcess.jar

Figure 6-34 Add utility JARs

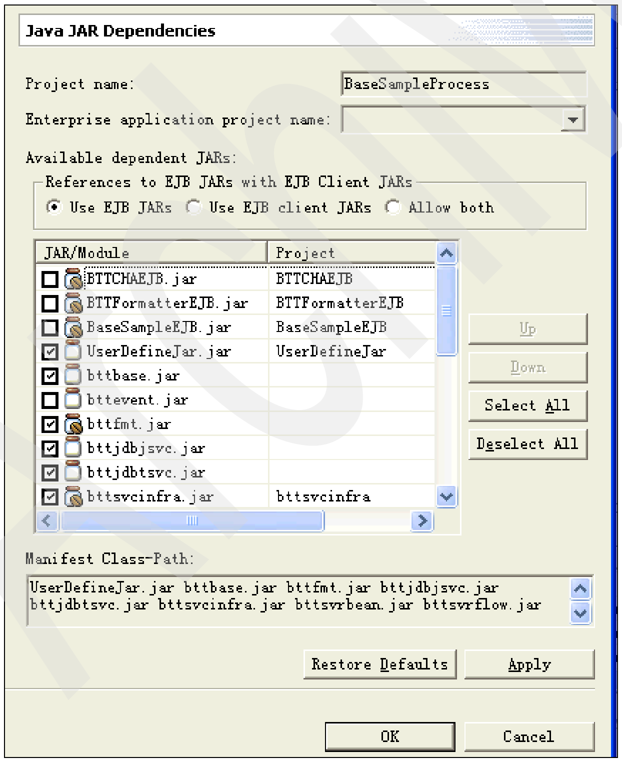
3. Right-click the **BaseSampleProcess** project. In the pop-up menu, select **New** → **Package** to create a package named `com.ibm.btt.samples.services`. Then create two Java classes named `DummyJournal` and `DummyJournalImpl` in that package. See the source code in

Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493.

4. Right-click the **BaseSampleProcess** project, and in the pop-up menu, select **Properties**. Select **Java JAR Dependencies**, and add the following JARs as dependencies:

   – **bttbase.jar**
   – **bttfmt.jar**
   – **bttjdbjsvc.jar**
   – **bttjdbtsvc.jar**
   – **bttsvcinfra.jar**
   – **bttsvrbean.jar**
   – **bttsvrflow.jar**
   – **sn0dummy.jar**
   – **UserDefineJar.jar**

   Click **OK**. See Figure 6-35



Java JAR Dependencies

Project name: BaseSampleProcess

Enterprise application project name:

Available dependent JARs:

References to EJB JARs with EJB Client JARs

[✓ Use EJB JARs] Use EJB client JARs Allow both

	JAR/Module	Project
☐	BTTCHAEJB.jar	BTTCHAEJB
☐	BTTFormatterEJB.jar	BTTFormatterEJB
☐	BaseSampleEJB.jar	BaseSampleEJB
☑	UserDefineJar.jar	UserDefineJar
☑	bttbase.jar	
☐	bttevent.jar	
☑	bttfmt.jar	
☑	bttjdbjsvc.jar	
☑	bttjdbtsvc.jar	
☑	bttsvcinfra.jar	bttsvcinfra

Manifest Class-Path:

```
UserDefineJar.jar bttbase.jar bttfmt.jar bttjdbjsvc.jar
bttjdbtsvc.jar bttsvcinfra.jar bttsvrbean.jar bttsvrflow.jar
```

Figure 6-35 JAR dependencies

5. Implement the snippets for the customerSearchServerOp business process.

   Expand the **BaseSampleProcess** project, and find the package named customerSearchServerOp.snippets. There are three Java files inside: initial.java, state2.java, and state3.java. Migration Tools only provides template for these classes. You should implement them depending on their business logic.

   – Code modification of initial.java

     Add the required import class code in Example 6-1 on top:

Example 6-1 Import class code for initial.java

```
import com.ibm.btt.base.Context;
import com.ibm.btt.formatter.client.FormatElement;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.dse.samples.appl.HostField;
```

In execute() method, add this code in Example 6-2:

Example 6-2 Additional code for initial.java execute() method

```
try{
    Context bpContext = getContext() ;
    if(bpContext.getParent() == null){
      Context parent =
Context.getContextByInstanceID(getSystemData().getInstanceId()) ;
      bpContext.chainTo(parent);
    }

    Journal journal;
    String hostBuff =
((FormatElement)getFormat("customerSearchSendFmt")).format(getContext())
;
    setValueAt("HostBuff",hostBuff);
         // writes to the journal using the appropriate format
    journal = (Journal)getService("JournalService");
    journal.addRecord(getContext(),"preSendJournalFmt");//$NON-NLS-1$
    journal.releaseServiceRequester();
    } catch(Exception e){
    e.printStackTrace();
    }
```

`Change "return 0" to "return 1".`

   – Code modification of state2.java

     Add the required import class code in Example 6-3 on top.

Example 6-3 Import class code for state2.java

```
import java.util.Hashtable;
import javax.resource.cci.*;
import com.ibm.btt.base.*;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0ConnectionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0InteractionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0Record;
import com.ibm.btt.services.*;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.btt.formatter.client.FormatElement;
```

In execute() method, add the code in Example 6-4.

Example 6-4 Additional code for state2.java execute() method

```
try{
      javax.naming.Context initialContext = null;
      ConnectionFactory connectionFactory = null;

      if (initialContext == null) {
        initialContext = new javax.naming.InitialContext();
        connectionFactory = (ConnectionFactory)
initialContext.lookup("snalu0");
      }
      // START TRANSACTION
      long begin = System.currentTimeMillis();


      // For testing Component-managed authentication.
      DummyLu0ConnectionSpec lu0ConnectionSpec = new
DummyLu0ConnectionSpec();
      lu0ConnectionSpec.setUserName("sna");
      lu0ConnectionSpec.setPassword("sna");
      Connection connection =
connectionFactory.getConnection(lu0ConnectionSpec);
      System.out.println("connection created...");

      // Beginning of testing SYNC_SEND_RECEIVE
      Interaction interaction = connection.createInteraction();
      System.out.println("interaction created...");
      DummyLu0InteractionSpec interactionSpec = new
DummyLu0InteractionSpec();

      DummyLu0Record in = new DummyLu0Record();
      DummyLu0Record out = new DummyLu0Record();

      in.setData((String) getValueAt("HostBuff"));

interactionSpec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE);
```

```
      interactionSpec.setExecutionTimeout(10000);

      interaction.execute(interactionSpec, in, out);
      System.out.println("data from host: " + out.getData());

      interaction.close();
      System.out.println("interaction closed...");
      connection.close();
      System.out.println("connection closed...");

      System.out.println("Before using customerSearchRecFmt to do the
format") ;
      com.ibm.btt.formatter.client.FormatElement fromHost =
(com.ibm.btt.formatter.client.FormatElement)
getFormat("customerSearchRecFmt"); //$NON-NLS-1$

      System.out.println("After using customerSearchRecFmt to do the
format") ;

      fromHost.unformat(out.getData(),getContext());

System.out.println("CustomerName==============="+getContext().tryGetElem
entAt("CustomerName"));
      System.out.println("session context
value=="+getContext().getParent().getKeyedCollection());
      System.out.println("AccountTransfer : executeSendHostStep ok!");

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

Change "`return 0`" to "`return 1`".

– Code modification of state3.java

Add the required import class code in Example 6-5 on top:

Example 6-5 Import class code for state3.java

```
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

In execute() method, add the code in Example 6-6:

Example 6-6 Additional code for state3.java execute() method

```
try{
  Journal journal;
  journal = (Journal)getService("JournalService");
    //journal.addRecord(getContext(),
"customerSearchRecFmt");//$NON-NLS-1$
  journal.addRecord(getContext(), "afterRecJournalFmt");//$NON-NLS-1$
  journal.releaseServiceRequester();
}catch(Exception e){
  e.printStackTrace();
}
```

Change "`return 0`" to" `return 1`";

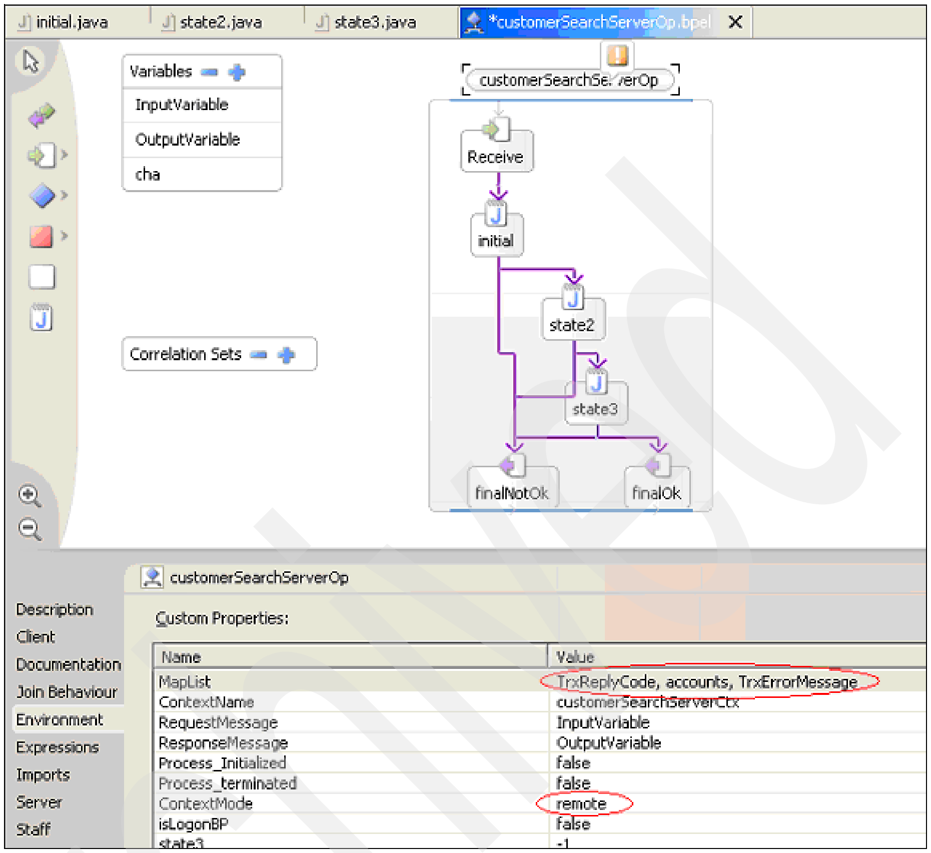
For the source code, refer to Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493.

6. Modify the BPEL file for the customerSearchServerOp Business Process.

   Find the package named *customerSearchServerOp*, and double-click **customerSearchServerOp.bpel** to edit.

   Select **customerSearchServerOp**. Its properties are displayed in the lower panel. Select **Environment**, and change the ContextMode to `remote`. Also add `TrxReplyCode`, `accounts`, `TrxErrorMessage` to the MapList. See Figure 6-36 on page 160.

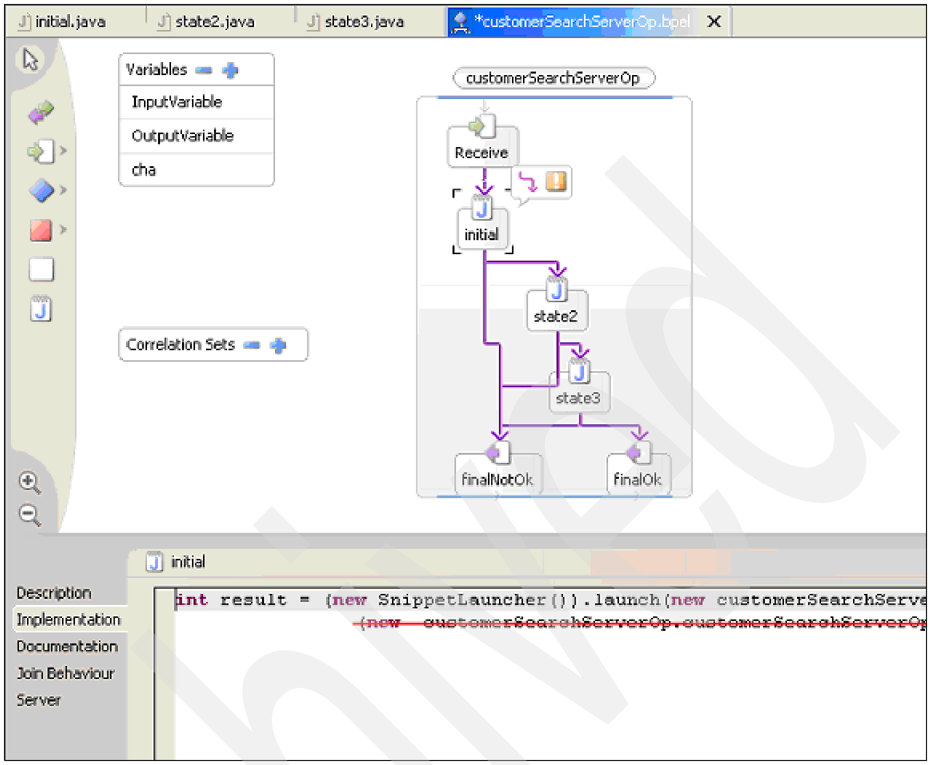
   Note: Do *not* leave white spaces after the comma.



Name	Value
MapList	TrxReplyCode, accounts, TrxErrorMessage
ContextName	customerSearchServerCtx
RequestMessage	InputVariable
ResponseMessage	OutputVariable
Process_Initialized	false
Process_terminated	false
ContextMode	remote
isLogonBP	false
state3	-1

Figure 6-36 Modify the business process environment properties

In the editor, select **initial** → **Implementation**. Delete the second line of code. See Figure 6-37 on page 161.



```
int result = (new SnippetLauncher()).launch(new customerSearchServe
                ~~(new  customerSearchServerOp.customerSearchServerOp~~
```

Figure 6-37   *Modify implementation for the initial activity*

In the same way, select **state2** → **Implementation**. Delete the second line of the code. Save and close the BPEL file.

7. Modify the WSDL file for customerSearchServerOp business process.

   In the same package, open **customerSearchServerOpInterface.wsdl** with Source Editor. Add a new line in the message tag named InputMessage:

```
<part name="CustomerId" type="xsd:string" />
```

   The result looks as follows:

```
<message name="InputMessage">
  <part name="systemData" type="xsd1:BTTSystemData"/>
  <part name="CustomerId" type="xsd:string" />
</message>
```

   Save and close **customerSearchServerOpInterface.wsdl**.

8. Implement the snippets for depositServerOp business process.

   Expand the **BaseSampleProcess** project, and find the package named depositServerOp.snippets. Edit three Java files inside: **initial.java**, **state2.java**, and **state3.java**.

   For depositServerOp.snippets.initial, add the code in Example 6-7 to the import class code on top.

Example 6-7 Import class code for depositServerOP.snippets.initial

```
import com.ibm.btt.base.Context;
import com.ibm.btt.formatter.client.FormatElement;
import com.ibm.btt.server.flow.al.workarea.BPWorkArea;
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.websphere.workarea.UserWorkArea;
```

   In execute() method add the code in Example 6-8.

Example 6-8 Additional code for depositServerOP.snippets.initial execute() method

```
try{
  if(getContext().getParent()==null)

getContext().chainTo(Context.getContextByInstanceID(getSystemData().getInst
anceId()));
    Journal journal;

setValueAt("HostBuff",((FormatElement)getFormat("depositSendFmt")).format(g
etContext()));//$NON-NLS-2$//$NON-NLS-1$
    journal = (Journal)getService("JournalService");
    journal.addRecord(getContext(),"preSendJournalFmt ");//$NON-NLS-1$
    journal.releaseServiceRequester();
    System.out.println("Deposit : executeJournalHostRequestDataStep ok!");
  } catch(Exception e){
    e.printStackTrace();
  }
```

   Change "`return 0`" to" `return 1`".

   For depositServerOp.snippets.state2, add the code in Example 6-9 to import class code on top:

Example 6-9 Import class code for depositServerOp.snippets.state2

```
import java.util.Hashtable;
import javax.resource.cci.*;
import com.ibm.btt.base.*;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0ConnectionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0InteractionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0Record;
import com.ibm.btt.services.*;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.btt.formatter.client.FormatElement;
```

In execute() method add the code in Example 6-10.

Example 6-10 Additional code for depositServerOp.snippets.state2 execute() method

```
try{
      javax.naming.Context initialContext = null;
      ConnectionFactory connectionFactory = null;

      if (initialContext == null) {
        initialContext = new javax.naming.InitialContext();
        connectionFactory = (ConnectionFactory)
initialContext.lookup("snalu0");
      }
      // START TRANSACTION
      long begin = System.currentTimeMillis();

      DummyLu0ConnectionSpec lu0ConnectionSpec = new
DummyLu0ConnectionSpec();
      lu0ConnectionSpec.setUserName("sna");
      lu0ConnectionSpec.setPassword("sna");
      Connection connection =
connectionFactory.getConnection(lu0ConnectionSpec);
      System.out.println("connection created...");


      // Beginning of testing SYNC_SEND_RECEIVE
      Interaction interaction = connection.createInteraction();
      System.out.println("interaction created...");
      DummyLu0InteractionSpec interactionSpec = new
DummyLu0InteractionSpec();

      DummyLu0Record in = new DummyLu0Record();
      DummyLu0Record out = new DummyLu0Record();

      in.setData((String) getValueAt("HostBuff"));


interactionSpec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE);
      interactionSpec.setExecutionTimeout(10000);

      interaction.execute(interactionSpec, in, out);

      System.out.println("data from host: " + out.getData());
```

```
        interaction.close();
        System.out.println("interaction closed...");
        connection.close();
        System.out.println("connection closed...");

        com.ibm.btt.formatter.client.FormatElement fromHost =
(com.ibm.btt.formatter.client.FormatElement) getFormat("depositRecFmt");
//$NON-NLS-1$
        fromHost.unformat(out.getData(),getContext());

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

Change "`return 0`" to" `return 1`".

For depositServerOp.snippets.state3, add the import class code in Example 6-11 on top:

Example 6-11 Import class code for depositServerOpsnippets.state3

```
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

In execute() method add code in Example 6-12.

Example 6-12 Additional code for depositServerOpsnippets.state3

```
try{
    Journal journal;
    journal = (Journal)getService("JournalService");
    journal.addRecord(getContext(), " afterRecJournalFmt");//$NON-NLS-1$
    journal.releaseServiceRequester();

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

Change "`return 0`" to"`return 1`".

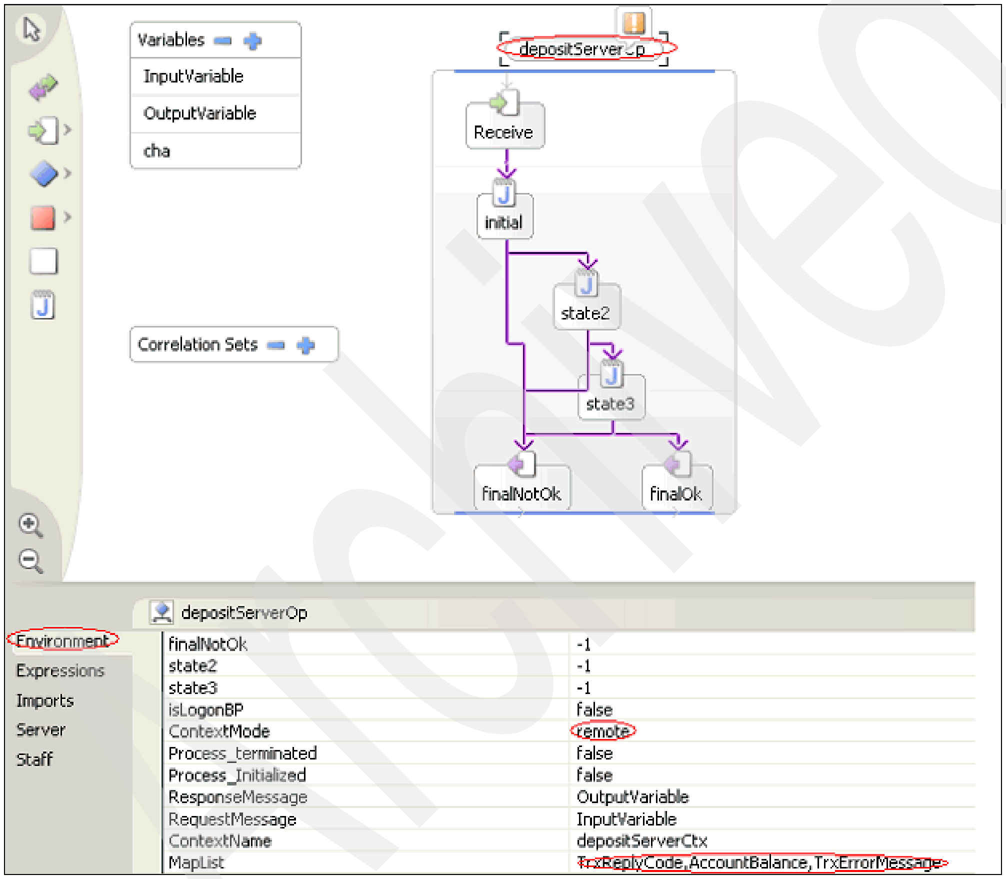
For the complete source code, refer to Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493.

9. Modify the BPEL file for depositServerOp business process.
   a. Find the package named depositServerOp, and double-click **depositServerOp.bpel** to edit.
   b. Select **depositServerOp**. Its properties are displayed in the lower panel. Select **Environment**, change the ContextMode to `remote`, and add

"`TrxReplyCode`, `AccountBalance`, `TrxErrorMessage`" to the MapList. See Figure 6-38.

- In the editor, select **initial** → **Implementation**. Delete the second line of the code.
- In the same way, select **state2** → **Implementation**. Delete the second line of the code.

Save and close the depositServerOp.bpel file.



depositServerOp

finalNotOk	-1
state2	-1
state3	-1
isLogonBP	false
ContextMode	remote
Process_terminated	false
Process_Initialized	false
ResponseMessage	OutputVariable
RequestMessage	InputVariable
ContextName	depositServerCtx
MapList	TrxReplyCode,AccountBalance,TrxErrorMessage

Figure 6-38 Modify environment property

10. Modify the *.wsdl file for the depositServerOp business process.

    In the same package, open **depositServerOpInterface.wsdl** with Source Editor. Add two new lines in the message tag named InputMessage, as in Example on page 166.

```
<part name="Amount" type="xsd:string" />
<part name="AccountNumber" type="xsd:string" />
```

The result should look as follows:

```
<message name="InputMessage">
  <part name="systemData" type="xsd1:BTTSystemData"/>
  <part name="Amount" type="xsd:string" />
  <part name="AccountNumber" type="xsd:string" />
</message>
```

Save and close **depositServerOpInterface.wsdl**..

11. Implement the snippets for the withdrawalServerOp business process.

    Expand the **BaseSampleProcess** project, and find the package named withdrawalServerOp.snippets. Edit three Java files inside: **initial.java**, **state2.java**, and **state3.java**.

    For withdrawalServerOp.snippets.initial, add the import class code in Example 6-13 on top.

Example 6-13 Added import code for withdrawalServerOp.snippets.initial.java

```
import com.ibm.btt.base.Context;
import com.ibm.btt.formatter.client.FormatElement;
import com.ibm.btt.server.flow.al.workarea.BPWorkArea;
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.websphere.workarea.UserWorkArea;
```

In the execute() method, add the code in Example 6-14.

Example 6-14 Code for execute() method of withdrawalServerOp.snippets.initial.java

```
try{

  if(getContext().getParent()==null)
  //getContext().chainTo(Context.getContextNamed("javaSessionCtx"));

getContext().chainTo(Context.getContextByInstanceID(getSystemData().getInst
anceId()));
  //      JDBCJournal journal;
   Journal journal;
setValueAt("HostBuff",((FormatElement)getFormat("withdrawalSendFmt")).forma
t(getContext()));//$NON-NLS-2$//$NON-NLS-1$

   // writes to the journal using the appropriate format
   journal = (Journal)getService("JournalService");
   journal.addRecord(getContext(),"preSendJournalFmt ");//$NON-NLS-1$
   journal.releaseServiceRequester();
   System.out.println("Withdrawal : executeJournalHostRequestDataStep
ok!");


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

Change "`return 0`" to" `return 1`";

For withdrawalServerOp.snippets.state2, add the needed import class code in Example 6-15 on top:

Example 6-15 Added import class code for withdrawalServerOp.snippets.state2

```
import java.util.Hashtable;
import javax.resource.cci.*;
import com.ibm.btt.base.*;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0ConnectionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0InteractionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0Record;
import com.ibm.btt.services.*;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.btt.formatter.client.FormatElement;
```

In the execute() method, add the code in Example 6-16:

Example 6-16 Additional code for withdrawalServerOp.snippets.state2 execute() method

```
try{
  javax.naming.Context initialContext = null;
  ConnectionFactory connectionFactory = null;
  if (initialContext == null) {
  initialContext = new javax.naming.InitialContext();
    connectionFactory = (ConnectionFactory)
initialContext.lookup("snalu0");
  }
  // START TRANSACTION
  long begin = System.currentTimeMillis();
  // For testing Component-managed authentication.
  DummyLu0ConnectionSpec lu0ConnectionSpec = new DummyLu0ConnectionSpec();
  lu0ConnectionSpec.setUserName("sna");
  lu0ConnectionSpec.setPassword("sna");
  Connection connection =
connectionFactory.getConnection(lu0ConnectionSpec);
  System.out.println("connection created...");

  // Beginning of testing SYNC_SEND_RECEIVE
  Interaction interaction = connection.createInteraction();
      System.out.println("interaction created...");
      DummyLu0InteractionSpec interactionSpec = new
DummyLu0InteractionSpec();

      DummyLu0Record in = new DummyLu0Record();
      DummyLu0Record out = new DummyLu0Record();

      in.setData((String) getValueAt("HostBuff"));


interactionSpec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE);
      interactionSpec.setExecutionTimeout(10000);

      interaction.execute(interactionSpec, in, out);

      System.out.println("data from host: " + out.getData());

      interaction.close();
      System.out.println("interaction closed...");
      connection.close();
      System.out.println("connection closed...");

      com.ibm.btt.formatter.client.FormatElement fromHost =
(com.ibm.btt.formatter.client.FormatElement) getFormat("withdrawalRecFmt");
//$NON-NLS-1$
      fromHost.unformat(out.getData(),getContext());


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

Change "`return 0`" to" `return 1`".

For withdrawalServerOp.snippets.state3, add the import class code in Example 6-17 on page 168 on top:

Example 6-17 Import class code for withdrawalServerOp.snippets.state3

```
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

In execute() method add the code in Example 6-18.

Example 6-18 Execute() method code for withdrawalServerOp.snippets.state3

```
try{
    Journal journal;
    journal = (Journal)getService("JournalService");
    journal.addRecord(getContext(), " afterRecJournalFmt ");//$NON-NLS-1$
    journal.releaseServiceRequester();
  }catch(Exception e){
    e.printStackTrace();
  }
```

Change "`return 0`" to "`return 1`";

For the source code, refer to Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493.

12. Modify the BPEL file for the withdrawalServerOp business process.

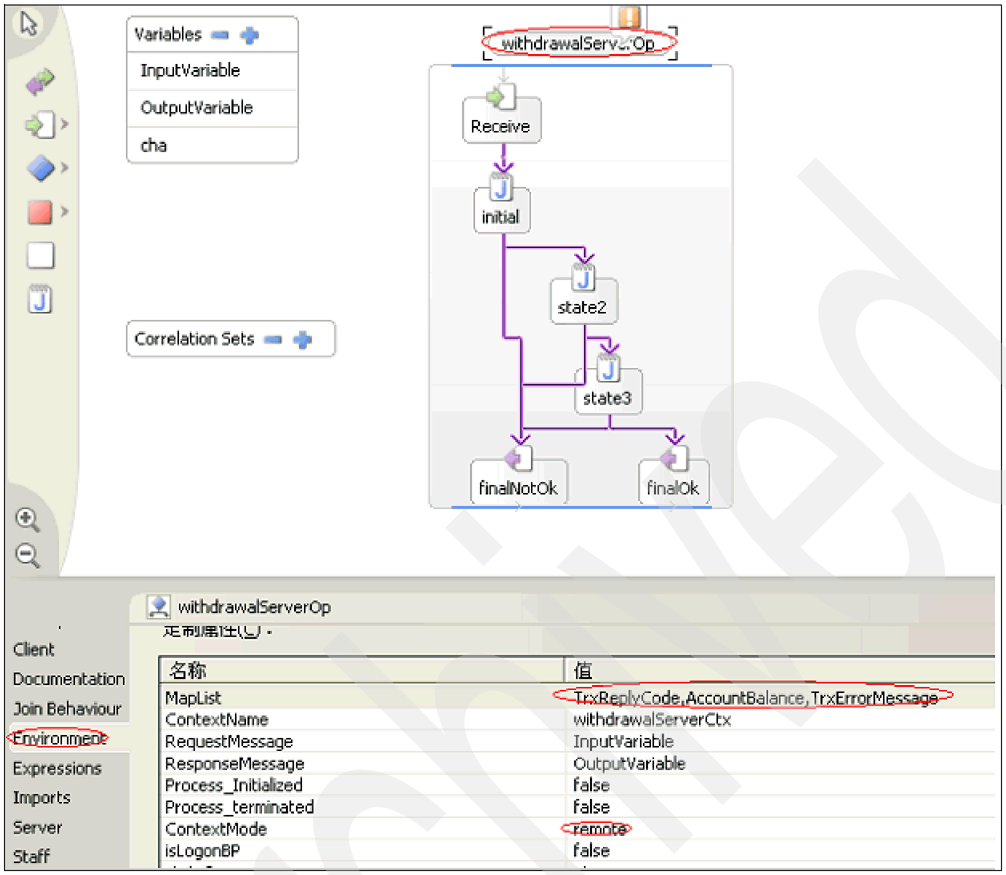
    Find the package named withdrawalServerOp, and double-click **withdrawalServerOp.bpel** to edit.

    a. Select **withdrawalServerOp**. It’s properties are displayed in the lower panel. Select Environment, and change the ContextMode to `remote`, also add "`TrxReplyCode`, `AccountBalance`, `TrxErrorMessage`" to the MapList. See Figure 6-39 on page 170.

       In the editor, select **initial** → **Implementation**. Delete the second line of the code.

    b. In the same way, select **state2** → **Implementation**. Delete the second line of the code.

    c. Save and close the BPEL file.



名称	值
MapList	TrxReplyCode,AccountBalance,TrxErrorMessage
ContextName	withdrawalServerCtx
RequestMessage	InputVariable
ResponseMessage	OutputVariable
Process_Initialized	false
Process_terminated	false
ContextMode	remote
isLogonBP	false

Figure 6-39 Modify environment properties for withdrawalServerOp.bpel

13.Modify the wsdl file for the withdrawalServerOp business process.

In the same package, open **withdrawalServerOpInterface.wsdl** with Source Editor. Add two new lines in the message tag named InputMessage:

```
<part name="Amount" type="xsd:string" />
<part name="AccountNumber" type="xsd:string" />
```

Save and close **withdrawalServerOpInterface.wsdl**.

14.Implement the snippets for the accountStatementServerOp business process.

Expand the **BaseSampleProcess** project, and find the package named accountStatementServerOp.snippets. Edit three Java files inside, **initial.java**, **state2.java**, and **state3.java**.

For accountStatementServerOp.snippets.initial, add the need to import class code on top:

```
import com.ibm.btt.base.Context;
import com.ibm.btt.formatter.client.FormatElement;
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

In execute() method add code:

```
  try{
    if(getContext().getParent()==null)
//getContext().chainTo(Context.getContextNamed("javaSessionCtx"));

getContext().chainTo(Context.getContextByInstanceID(getSystemData().getInst
anceId()));
    Journal journal;

setValueAt("HostBuff",((FormatElement)getFormat("accountStatementSendFmt"))
.format(getContext()));//$NON-NLS-2$//$NON-NLS-1$

     // writes to the journal using the appropriate format
    journal = (Journal)getService("JournalService");
    journal.addRecord(getContext(),"preSendJournalFmt ");//$NON-NLS-1$
    journal.releaseServiceRequester();
  } catch(Exception e){
    e.printStackTrace();
  }
```

Change "`return 0`" to" `return 1`".

For accountStatementServerOp.snippets.state2, add the need to import class code on top:

```
import java.util.Hashtable;
import javax.resource.cci.*;
import com.ibm.btt.base.*;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0ConnectionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0InteractionSpec;
import com.ibm.btt.samples.business.sna.lu0.DummyLu0Record;
import com.ibm.btt.services.*;
import com.ibm.btt.services.jdbcjournalservice.Journal;
import com.ibm.btt.formatter.client.FormatElement;
```

In execute() method add code:

```
try{
    javax.naming.Context initialContext = null;
    ConnectionFactory connectionFactory = null;
    if (initialContext == null) {
      initialContext = new javax.naming.InitialContext();
      connectionFactory = (ConnectionFactory)
initialContext.lookup("snalu0");
```

```
    }
    // START TRANSACTION
    long begin = System.currentTimeMillis();

    // For testing Component-managed authentication.
    DummyLu0ConnectionSpec lu0ConnectionSpec = new
DummyLu0ConnectionSpec();
    lu0ConnectionSpec.setUserName("sna");
    lu0ConnectionSpec.setPassword("sna");
    Connection connection =
connectionFactory.getConnection(lu0ConnectionSpec);
    System.out.println("connection created...");

    // Beginning of testing SYNC_SEND_RECEIVE
    Interaction interaction = connection.createInteraction();
    System.out.println("interaction created...");
    DummyLu0InteractionSpec interactionSpec = new
DummyLu0InteractionSpec();

    DummyLu0Record in = new DummyLu0Record();
    DummyLu0Record out = new DummyLu0Record();

    in.setData((String) getValueAt("HostBuff"));

    interactionSpec.setInteractionVerb(InteractionSpec.SYNC_SEND_RECEIVE);
    interactionSpec.setExecutionTimeout(10000);

    interaction.execute(interactionSpec, in, out);

    System.out.println("data from host: " + out.getData());

    interaction.close();
    System.out.println("interaction closed...");
    connection.close();
    System.out.println("connection closed...");

    com.ibm.btt.formatter.client.FormatElement fromHost =
(com.ibm.btt.formatter.client.FormatElement)
getFormat("accountStatementRecFmt"); //$NON-NLS-1$
    fromHost.unformat(out.getData(),getContext());
  }catch(Exception e){
    e.printStackTrace();
  }
```

Change "`return 0`" to" `return 1`".

For the accountStatementServerOp.snippets.state3, add the need to import class code on top:

```
import com.ibm.btt.services.jdbcjournalservice.JDBCJournal;
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

In execute() method add code:

```
try{
    Journal journal;
    journal = (Journal)getService("JournalService");
    journal.addRecord(getContext(), " afterRecJournalFmt ");//$NON-NLS-1$
    journal.releaseServiceRequester();

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

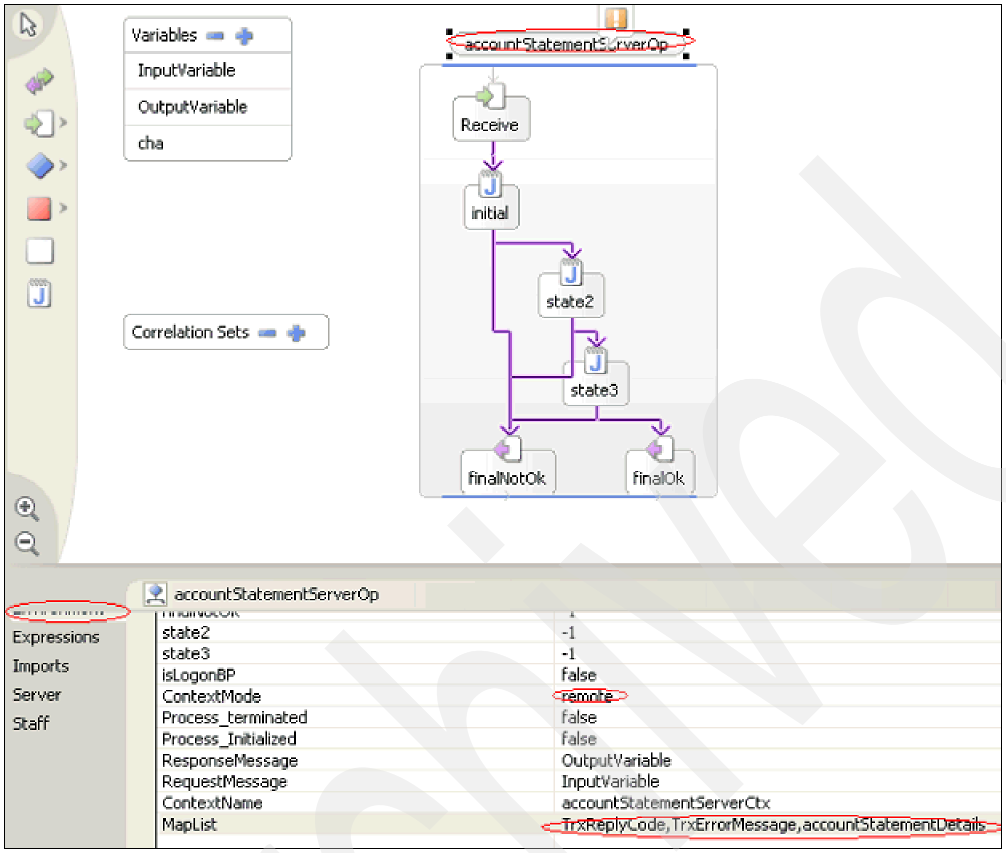
Change "`return 0`" to" `return 1`".

For the source code, please refer to the Appendix (Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493).

15. Modify the BPEL file for accountStatementServerOp business process.

    Find the package named accountStatementServerOp, and double-click **accountStatementServerOp.bpel** to edit.

    Select **accountStatementServerOp**. It’s properties are displayed in the lower panel. Select **Environment**, and change the ContextMode to `remote`, and add "`TrxReplyCode`, `accountStatementDetails`, `TrxErrorMessage`" to the MapList. See Figure 6-40 on page 174.



state2	-1
state3	-1
isLogonBP	false
ContextMode	remote
Process_terminated	false
Process_Initialized	false
ResponseMessage	OutputVariable
RequestMessage	InputVariable
ContextName	accountStatementServerCtx
MapList	TrxReplyCode,TrxErrorMessage,accountStatementDetails

Figure 6-40 Modify environment properties for accountStatementServerOp.bpel

In the editor, select **initial** → **Implementation**. Delete the second line of the code.

In the same way, select **state2** → **Implementation**. Delete the second line of the code.

Save and close the BPEL file.

16. Modify the wsdl file for the accountStatementServerOp business process.

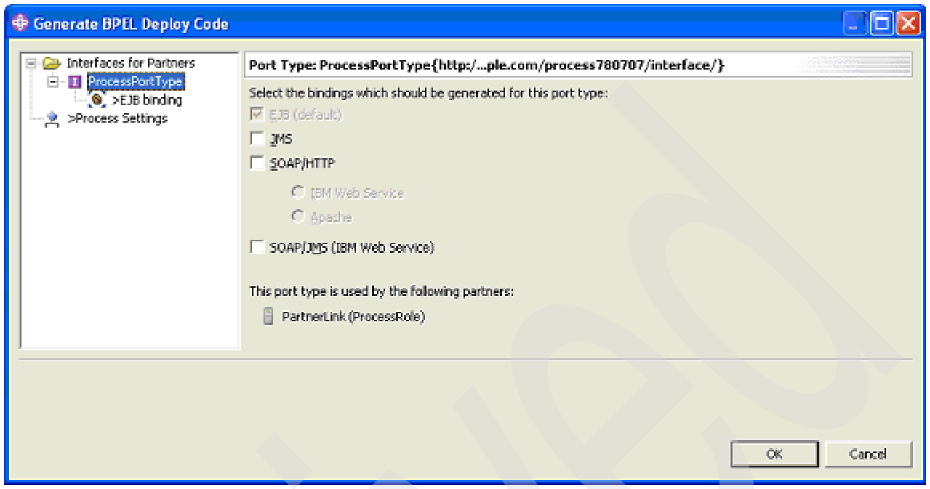
    In the same package, open **accountStatementServerOpInterface.wsdl** with Source Editor. Add a new line in the message tag named InputMessage:

    ```
    <part name="AccountNumber" type="xsd:string" />
    ```

    Save and close **accountStatementServerOpInterface.wsdl**.

17. Deploy the BP codes. For the four BPEL files modified, right-click in the J2EE perspective and select **Enterprise Services** → **Generate Deploy Code**.

Click **OK** in the pop-up window. All the four BPEL files should be deployed separately. See Figure 6-41.


Generate BPEL Deploy Code

Interfaces for Partners
ProcessPortType
>EJB binding
>Process Settings

Port Type: ProcessPortType{http:/...ple.com/process780707/interface/}

Select the bindings which should be generated for this port type:

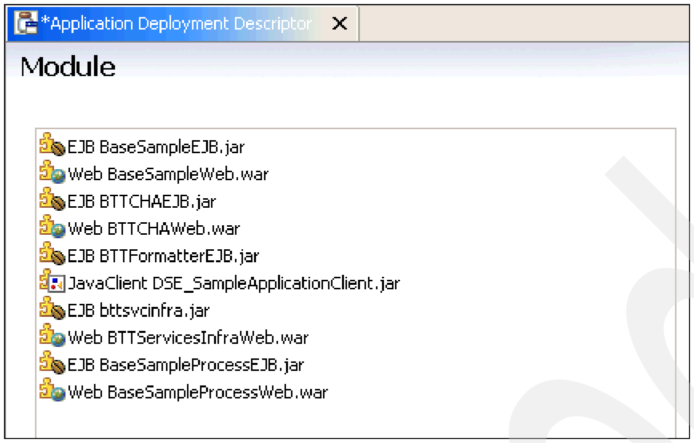
[✓] EJB (default)
[ ] JMS
[ ] SOAP/HTTP
( ) IBM Web Service
( ) Apache
[ ] SOAP/JMS (IBM Web Service)

This port type is used by the following partners:
PartnerLink (ProcessRole)

Figure 6-41 Generate BPEL deploy code

18. After the deployment, a new project named BaseSampleProcessEAR is created automatically. Copy the following JAR files from the BaseSample project into BaseSampleProcessEAR.
    - **bttbase.jar**
    - **bttevent.jar**
    - **bttfmt.jar**
    - **bttjdbjsvc.jar**
    - **bttjdbtsvc.jar**
    - **bttsvcinfra.jar**
    - **bttsvrbean.jar**
    - **bttsvrflow.jar**
    - **sn0dummy.jar**

    Since BaseSampleProcessEJB needs these jar files, remove the compile error needed to copy these jar files.
19. Open the **application.xml** file in the META-INF folder of the BaseSample project in the J2EE perspective. In the Module tab, click **Add** to include BaseSampleProcessEJB and BaseSampleProcessWeb projects. Save and close the file. See Figure 6-42 on page 176



EJB BaseSampleEJB.jar
Web BaseSampleWeb.war
EJB BTTCHAEJB.jar
Web BTTCHAWeb.war
EJB BTTFormatterEJB.jar
JavaClient DSE_SampleApplicationClient.jar
EJB bttsvcinfra.jar
Web BTTServicesInfraWeb.war
EJB BaseSampleProcessEJB.jar
Web BaseSampleProcessWeb.war

Figure 6-42 Add modules to the application

20. Add the services' properties files to BaseSampleProcessEJB project.

    Get BTTJavaSample.ear from BTT5.1 installpackaging's path such as C:\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1\samples\JavaSampleApplication\StandAlone.

    Extract **BTTJavaSampleEJB.jar** in BTTJavaSample.ear and copy the following properties files:

    – **DummyJournal.properties**
    – **LocalJava.properties**
    – **RemoteEJB.properties**
    – **ServiceRequesterIDs.properties**
    – **WSIFEJB.properties**
    – **WSIFSoap.properties**
    – **sampleapplserver.properties**

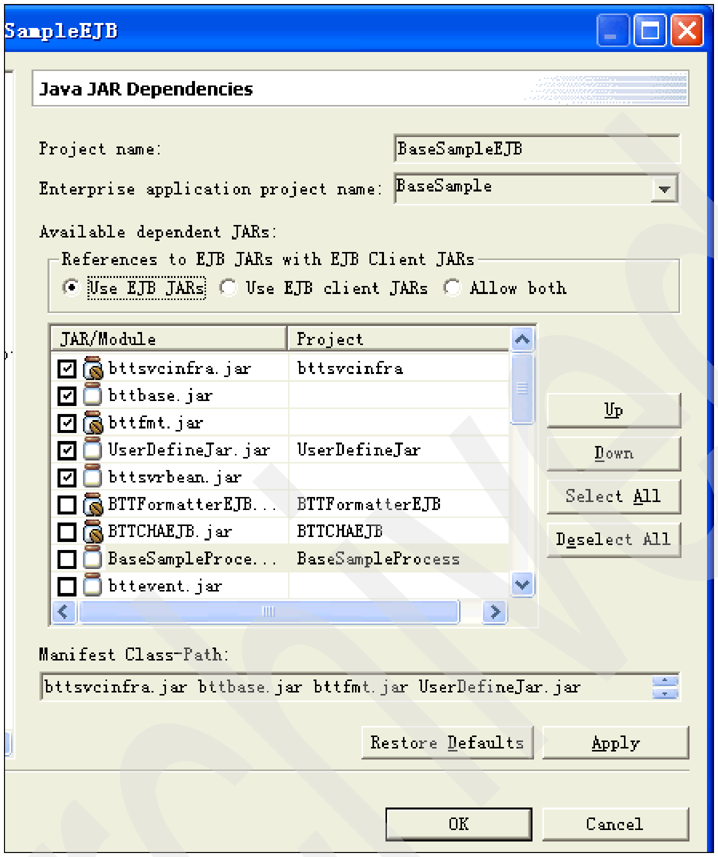
    In the J2EE perspective, paste the properties files into the ejbModule folder of the BaseSampleProcessEJB project.

Configuring Single Action EJBs

1. Right-click the BaseSampleEJB project. In the pop-up menu, select **Properties**. Select **Java JAR Dependencies**, and add the following JARs as dependencies.

    – **bttbase.jar**
    – **bttfmt.jar**
    – **bttsvcinfra.jar**
    – **bttsvrbean.jar**
    – **UserDefineJar.jar**

Click **OK**. See Figure 6-43.


SampleEJB

Java JAR Dependencies

Project name: BaseSampleEJB

Enterprise application project name: BaseSample

Available dependent JARs:

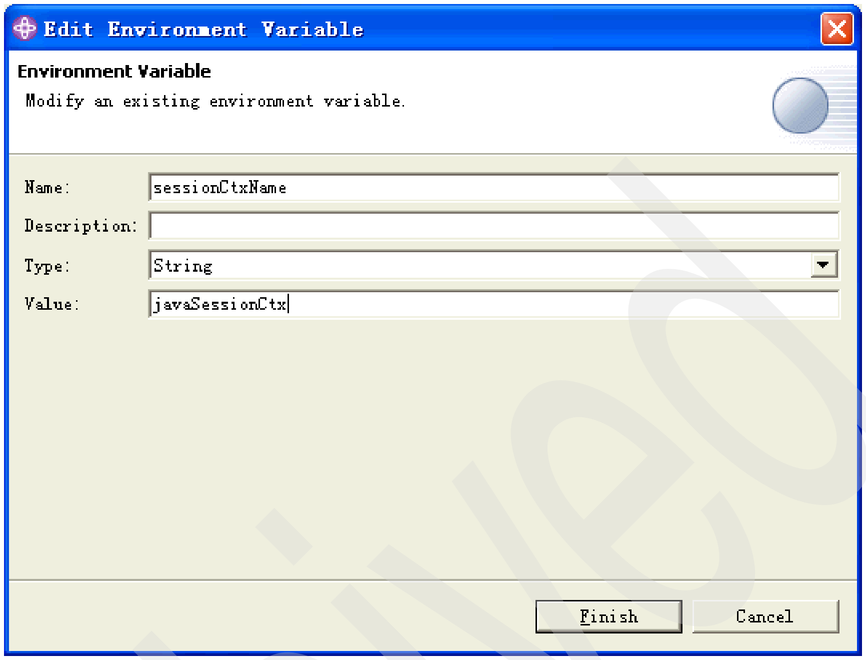
References to EJB JARs with EJB Client JARs: [✓ Use EJB JARs] Use EJB client JARs Allow both

JAR/Module	Project
☑ bttsvcinfra.jar	bttsvcinfra
☑ bttbase.jar	
☑ bttfmt.jar	
☑ UserDefineJar.jar	UserDefineJar
☑ bttsvrbean.jar	
☐ BTTFormatterEJB...	BTTFormatterEJB
☐ BTTCHAEJB.jar	BTTCHAEJB
☐ BaseSampleProce...	BaseSampleProcess
☐ bttevent.jar	

Manifest Class-Path: bttsvcinfra.jar bttbase.jar bttfmt.jar UserDefineJar.jar

Figure 6-43 Add Java JAR dependencies

2. Open the **ejb-jar.xml** file in the META-INF folder. In the Beans tab, choose **startupServerOp** and edit its Environment Variables. Select **sessionCtxName** from the variables list and click **Edit**. Then change its value to `javaSessionCtx`. Click **Finish**, and save and close the file. See Figure 6-44 on page 178.


Edit Environment Variable
Environment Variable
Modify an existing environment variable.

Name: sessionCtxName
Description:
Type: String
Value: javaSessionCtx

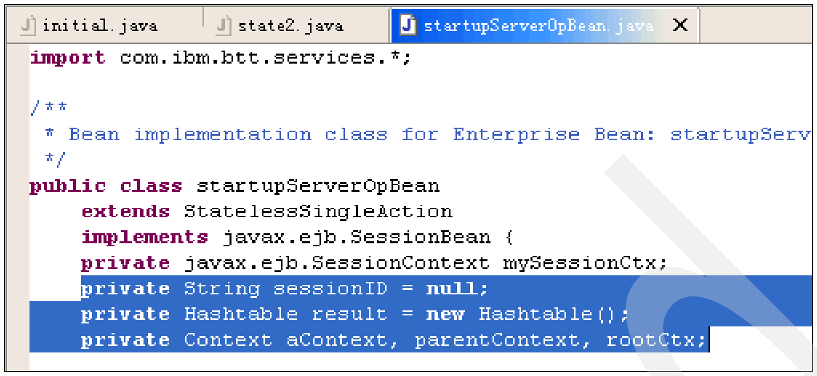
Figure 6-44 Edit environment variables

3. Expand the **com.ibm.dse.samples.appl** package to find the startupServerOpBean.java and startupServerOp.java file. Implement their execute( ) methods:

   For com.ibm.dse.samples.appl.startupServerOpBean.java, add

   ```
   private String sessionID = null;
   private Hashtable result = new Hashtable();
   private Context aContext, parentContext, rootCtx;
   ```

   See Figure 6-45 on page 179.


initial.java | state2.java | startupServerOpBean.java

```
import com.ibm.btt.services.*;

/**
 * Bean implementation class for Enterprise Bean: startupServ
 */
public class startupServerOpBean
    extends StatelessSingleAction
    implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext mySessionCtx;
    private String sessionID = null;
    private Hashtable result = new Hashtable();
    private Context aContext, parentContext, rootCtx;
```

Figure 6-45   Implement execute methods

Modify the execute() method. Change the following:

```
public Hashtable execute() throws BTTSAEException, Exception {
    Hashtable result = new Hashtable();

    //user code here

    return result;
  }
```

This should now look as follows:

```
  public Hashtable execute(BTTSystemData sysData, String wksContext,String
wksParentContext) throws BTTSAEException, Exception {
    Hashtable result = new Hashtable();

    //user code here
    try {
          initialize(sysData);
          setupSessionContext();
          result.put("InstanceID", getInstanceId());

          close();

      } catch (BTTSAEException ex) {
          ex.printStackTrace();
          throw new DSEInvalidRequestException(DSEException.critical,
                                getClass().getName(),
                                "Not able to create the session context " +
" in " + getName() + " \n" + ex.toString());
            }
    return result;
  }
```

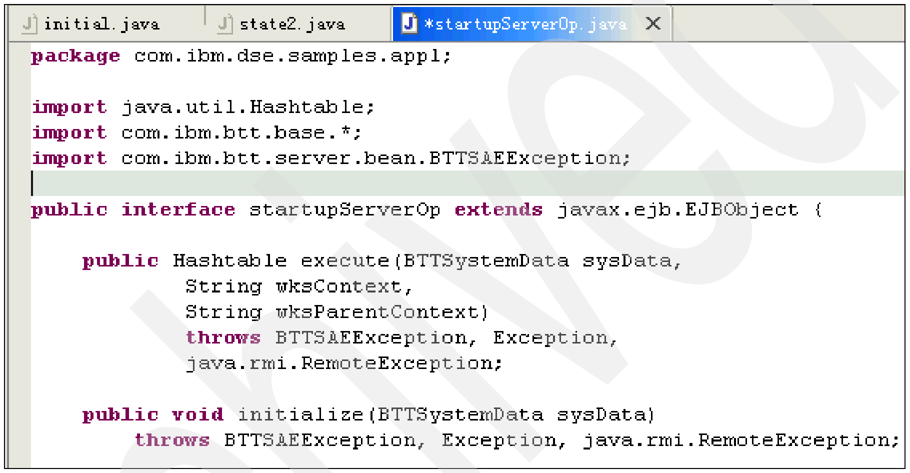
For startupServerOp.java, modify the execute() method. Change the following:

```
public Hashtable execute()
throws BTTSAEException, Exception, java.rmi.RemoteException;
```

This should look as follows:

```
public Hashtable execute(BTTSystemData sysData,String wksContext,String
wksParentContext) throws BTTSAEException, Exception,
java.rmi.RemoteException;
```

See Figure 6-46



```
package com.ibm.dse.samples.appl;

import java.util.Hashtable;
import com.ibm.btt.base.*;
import com.ibm.btt.server.bean.BTTSAEException;

public interface startupServerOp extends javax.ejb.EJBObject {

    public Hashtable execute(BTTSystemData sysData,
            String wksContext,
            String wksParentContext)
            throws BTTSAEException, Exception,
            java.rmi.RemoteException;

    public void initialize(BTTSystemData sysData)
        throws BTTSAEException, Exception, java.rmi.RemoteException;
```

Figure 6-46   Execute method for startupServerOp.java

For the source code, refer to the Appendix (Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493).

4. In the same way, implement the execute( ) methods of the endSessionServerOpBean.java and endSessionServerOp.java file:

   For endSessionServerOpBean.java, modify the execute() method. Change the following:

```
public Hashtable execute() throws BTTSAEException, Exception {
    Hashtable result = new Hashtable();

    //user code here

    return result;
  }
```

This should look as follows:

```
  public Hashtable execute(BTTSystemData sysData) throws
BTTSAEException,Exception {

    initialize(sysData);

    Hashtable result = new Hashtable();
    Context sessionCtx ;

    //if instanceID != null
    if( getInstanceId() != null ){
      // Removes the context and its parent (the parent session context)
      sessionCtx = Context.getContextByInstanceID(getInstanceId());
      if(sessionCtx instanceof Context)
        sessionCtx.prune();
    }

    close();

    return result;
  }
```

For endSessionServerOp.java, modify the execute() method. Change the following:

```
public Hashtable execute()
    throws BTTSAEException, Exception, java.rmi.RemoteException;
```

This should look as follows:

```
public Hashtable execute(BTTSystemData sysData)
    throws BTTSAEException, Exception, java.rmi.RemoteException;
```

For the source code, refer to the Appendix (Appendix A, "Branch Transformation Toolkit development and runtime requirements" on page 477 and Appendix B, "Setting up a Branch Transformation Toolkit sample application" on page 493).

5. Save all the changes. If any error exists in the BaseSampleEJB project, choose the **Java Build Path** in the Properties window of the project and add the JARs **bttbase.jar**, **bttfmt.jar**, and **bttsvrbean.jar** to the Libraries tab and select the **bttsvcinfra** and **UserDefineJar** projects in the Projects tab.
6. Make sure that all the errors in the BaseSampleEJB project have been fixed before deploying. Right-click the project, and from the pop-up menu, select **Generate → Deployment and RMIC Code...**, click **Select All** and **Finish**.

6.4.2 Presentation layer

In this section we describe how you configure and test the presentation layer of our sample application.

HTML client

1. Configuring the HTML client.

   First, create the servlets needed and then configure them by performing the following tasks:

   a. Expand the **BaseSampleWeb** project. Right-click the **Java Resources** folder and select **New** → **Package** to create a new package named `com.ibm.btt.samples.appl`.

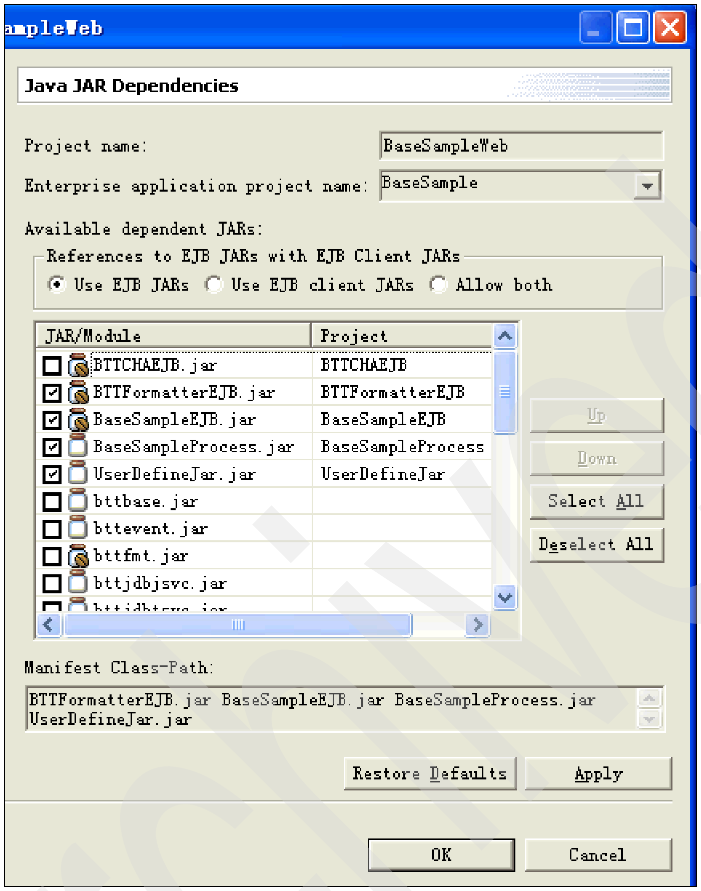
   b. Right-click the **com.ibm.btt.samples.appl** package and select **New** → **Class** to create a Java class named `StartServerServlet`.

      For the source code, refer to the Appendix (Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493).

   c. Copy the **bttevent.jar** file from the BaseSample project to the /WebContent/WEB-INF/lib folder of the BaseSampleWeb project.

   d. Configure the project to fix the compile errors. Right-click the project and choose **Properties** from the pop-up menu to open the Properties window.

   e. Select **Java JAR Dependencies** from the left panel and check **BTTFormatterEJB.jar**, **BaseSampleEJB.jar**, **BaseSampleProcess.jar**, **BaseSampleProcessEJB.jar**, and **UserDefineJar.jar** from the list. See Figure 6-47 on page 183.


ampleWeb

Java JAR Dependencies

Project name: BaseSampleWeb

Enterprise application project name: BaseSample

Available dependent JARs:

References to EJB JARs with EJB Client JARs
(● Use EJB JARs) ○ Use EJB client JARs ○ Allow both

JAR/Module	Project
☐ BTTCHAEJB.jar	BTTCHAEJB
☑ BTTFormatterEJB.jar	BTTFormatterEJB
☑ BaseSampleEJB.jar	BaseSampleEJB
☑ BaseSampleProcess.jar	BaseSampleProcess
☑ UserDefineJar.jar	UserDefineJar
☐ bttbase.jar	
☐ bttevent.jar	
☐ bttfmt.jar	
☐ bttjdbjsvc.jar	

Manifest Class-Path:

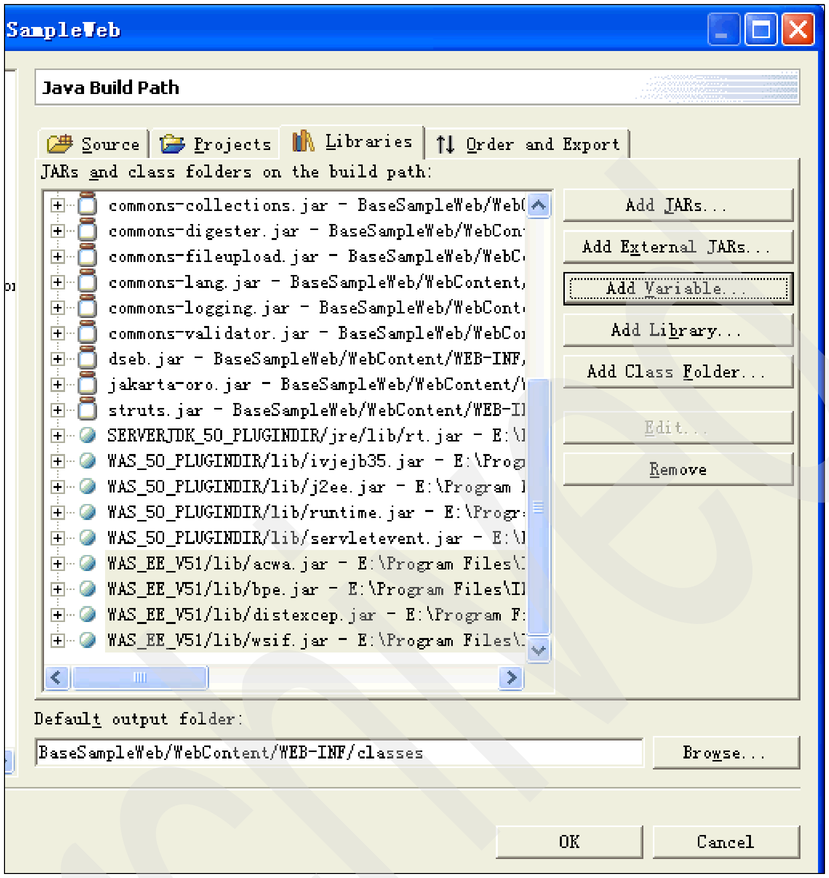
```
BTTFormatterEJB.jar BaseSampleEJB.jar BaseSampleProcess.jar
UserDefineJar.jar
```

Figure 6-47 Java JAR depencies for BaseSampleWeb

f. Select **Java Build Path** from the left panel and select the **Libraries** tab. Click **Add Variable** and select **WAS_EE_V51** from the list. Click **Extend** and expand the **Libraries** folder. Select the following JAR files to add to the build path. Click **OK** to close the window.

- **acwa.jar**
- **bpe.jar**
- **distexcep.jar**
- **wsif.jar**

See Figure 6-48 on page 184.


SampleWeb

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

```
commons-collections.jar - BaseSampleWeb/WebC
commons-digester.jar - BaseSampleWeb/WebCon
commons-fileupload.jar - BaseSampleWeb/WebC
commons-lang.jar - BaseSampleWeb/WebContent/
commons-logging.jar - BaseSampleWeb/WebCont
commons-validator.jar - BaseSampleWeb/WebCo
dseb.jar - BaseSampleWeb/WebContent/WEB-INF/
jakarta-oro.jar - BaseSampleWeb/WebContent/
struts.jar - BaseSampleWeb/WebContent/WEB-I
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - E:\
WAS_50_PLUGINDIR/lib/ivjejb35.jar - E:\Prog
WAS_50_PLUGINDIR/lib/j2ee.jar - E:\Program
WAS_50_PLUGINDIR/lib/runtime.jar - E:\Progr
WAS_50_PLUGINDIR/lib/servletevent.jar - E:\
WAS_EE_V51/lib/acwa.jar - E:\Program Files\
WAS_EE_V51/lib/bpe.jar - E:\Program Files\I
WAS_EE_V51/lib/distexcep.jar - E:\Program F
WAS_EE_V51/lib/wsif.jar - E:\Program Files\
```

Default output folder:

BaseSampleWeb/WebContent/WEB-INF/classes

Figure 6-48 Java build partn for BaseSampleWeb

2. Edit the JSP files.

   a. Expand the **BaseSampleWeb** project. Remove all the four JSP files, that is, **accountStatement.jsp**, **customerSearch.jsp**, **error.jsp**, and **txn.jsp** from the WebContent/jsp folder, and reorganize them in separate folders.

   b. Within the WebContent/jsp folder, create new folders and new JSP files:

      - Create new folder `startUp`. Create new JSPs, that is, `welcome.jsp` and `errorpage.jsp`.

        For welcome.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
```

```
<btt:html>

<head>
<title>Welcome</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body text=#ffffff bgColor=#3366ff
background="/BaseSampleWeb/jsp/images/back_interior.gif" >
<FONT face=Arial color=#ffffff size=5>Demo Bank - Welcome</FONT>
<br>
<br>
<HR>
<center>
<btt:form action="/prepareCustomerSearch">
<TABLE cellSpacing=0 cellPadding=0 width="60%" border=0>
 <TBODY>
 <TR>
  <TD align=center><FONT face=Verdana color=#ffffff
size=1><html:submit property="Click to
start"/></FONT></TD></TR></TBODY></TABLE>
<BR>
<BR>
</center>
</btt:form>
<br>
<br>
</body>
</btt:html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

- Create a new folder `customerSearchBP`. Create new JSPs, `customerSearch.jsp`, `customerSearchList.jsp`, and `errorpage.jsp`.

  For customerSearch.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<btt:html>

<head>
<title>Customer log on</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body bgcolor="#3366FF" text="#FFFFFF"
background="/BaseSampleWeb/jsp/images/back_interior.gif" >

<btt:errors/>
<btt:form action="/customerSearchBP">
<br>
<br>
<center>
<table align="center" border="1" cellspacing="1" cellpadding="0"
width=60%>
  <TR>
    <TD height="40" bgcolor="#002184" width="40%"
align="center"><FONT face="Verdana" size="2"
color="#FFFFFF"><STRONG>Customer ID:     </STRONG></FONT></TD>
    <TD bgcolor="#0A35B8" align="center"><FONT face="Verdana"
size="2" color="#FFFFFF"><input type=text name="customerId"
value=""></FONT></td><td></TD>
  </TR>
</table>
<BR>
<BR>

<table align="center" border="0" cellspacing="0" cellpadding="0"
width=60%>
<TR>
<TD align="center">
<FONT face="Verdana" size="1" color="#FFFFFF">
<input type=submit value="Search">
</FONT>
</TD>
</TR>
</table>
</center>
</btt:form>

<br>
<br>
</body>
</btt:html>
```

For customerSearchList.jsp:

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

<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<btt:html>
<HEAD>
<title>Demo Bank - Transactions Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<body bgcolor="#3366FF" text="#FFFFFF"
background="/BaseSampleWeb/jsp/images/back_interior.gif">
<FONT size="5" color="#FFFFFF" face="Arial">Demo Bank - Transactions
Page</FONT>
<script language="JavaScript">
function submitAction(actionName){
customerSearchForm.action=actionName;
customerSearchForm.submit();

}
</script>
<br>
<p>
<btt:form action="/customerSearchBP">
<center>
<table border="1" cellspacing="1" cellpadding="1" width=60%>
  <TR>
    <TD height="40" bgcolor="#002184" align="center"
colspan="4"><FONT face="Verdana" size="2"
color="#FFFFFF"><STRONG>Customer Name: </STRONG>
    <btt:label dataName="CustomerName"/>
</FONT></TD>
  </TR>
  <TR>
    <TD height="35" bgcolor="#002184" width="35%" align="left"><FONT
face="Verdana" size="2" color="#FFFFFF"><STRONG>Account
Number</STRONG></FONT></TD>
    <TD bgcolor="#0A35B8" align="right"><FONT face="Verdana" size="2"
color="#FFFFFF">
    <btt:combo dataName="AccountNumber" dataNameForList="accounts"
value="AccountNumber" item="Name"/>
      </FONT></TD>
  </TR>
  <TR>
    <TD height="35" bgcolor="#002184" width="35%" align="left"><FONT
face="Verdana" size="2"
color="#FFFFFF"><STRONG>Amount</STRONG></FONT></TD>
```

```
    <TD bgcolor="#0A35B8" align="right"><FONT face="Verdana" size="2"
color="#FFFFFF">
    <btt:text property="Amount"/></FONT></TD>
  </TR>

</table>
<table align="center" border="0" cellspacing="0" cellpadding="0"
width=60%>
<TR align="center" valign="baseline">
<TD align="center"><input type=button align="middle" value="Deposit"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/depositBP/deposi
tBP.do');"></TD>
<TD align="center"><input type=button align="middle"
value="Withdrawal"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/withdrawalBP/wit
hdrawalBP.do');"></TD>
<TD align="center"><input type=button align="middle" value="Account
Statement"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/accountStatement
BP/accountStatementBP.do');"></TD>
<TD align="center"><input type=button align="middle" value="Cancel"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/endSession/endSe
ssion.do');" ></TD>
</TR>
</table>
</center>
</btt:form>
<br>
<br>
<br>
</body>
</btt:html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

- Create a new folder, `depositBP`. In the new folder, create new JSPs `depositList.jsp`, `errorpage.jsp`.

  For depositList.jsp:

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

<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<btt:html>
<HEAD>
<title>Demo Bank - Transactions Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<body bgcolor="#3366FF" text="#FFFFFF"
background="/BaseSampleWeb/jsp/images/back_interior.gif">
<FONT size="5" color="#FFFFFF" face="Arial">Demo Bank - Transactions
Page</FONT>
<script language="JavaScript">
function submitAction(actionName){
depositForm.action=actionName;
depositForm.submit();

}

</script>
<br>
<p>
<btt:form action="/depositBP">
<center>
<table border="1" cellspacing="1" cellpadding="1" width=60%>
  <TR>
    <TD height="40" bgcolor="#002184" align="center"
colspan="4"><FONT face="Verdana" size="2"
color="#FFFFFF"><STRONG>Customer Name: </STRONG>
    <btt:label dataName="CustomerName"/>
</FONT></TD>
  </TR>
  <TR>
    <TD height="35" bgcolor="#002184" width="35%" align="left"><FONT
face="Verdana" size="2" color="#FFFFFF"><STRONG>Account
Number</STRONG></FONT></TD>
    <TD bgcolor="#0A35B8" align="right"><FONT face="Verdana" size="2"
color="#FFFFFF">
    <btt:combo dataName="AccountNumber" dataNameForList="accounts"
value="AccountNumber" item="Name"/>
      </FONT></TD>
  </TR>
  <TR>
```

```
    <TD height="35" bgcolor="#002184" width="35%" align="left"><FONT
face="Verdana" size="2"
color="#FFFFFF"><STRONG>Amount</STRONG></FONT></TD>
    <TD bgcolor="#0A35B8" align="right"><FONT face="Verdana" size="2"
color="#FFFFFF">
    <btt:text property="Amount"/></FONT></TD>
  </TR>

</table>
<table align="center" border="0" cellspacing="0" cellpadding="0"
width=60%>
<TR align="center" valign="baseline">
<TD align="center"><input type=button align="middle" value="Deposit"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/depositBP/deposi
tBP.do');"></TD>
<TD align="center"><input type=button align="middle"
value="Withdrawal"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/withdrawalBP/wit
hdrawalBP.do');"></TD>
<TD align="center"><input type=button align="middle" value="Account
Statement"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/accountStatement
BP/accountStatementBP.do');"></TD>
<TD align="center"><input type=button align="middle" value="Cancel"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/endSession/endSe
ssion.do');" ></TD>
</TR>
</table>
</center>
</btt:form>

<br>
<br>
<table>
<TR> <TD align=left><FONT face="Verdana" size="2"
color="#FFFFFF">Balance: </FONT></TD>
<TD align=left><FONT face="Verdana" size="2"
color="#FFFFFF"><btt:label
dataName="AccountBalance"/></FONT></TD></TR>
<TR> <TD align=left><FONT face="Verdana" size="2"
color="#FFFFFF">Message: </FONT></TD>
<TD align=left><FONT face="Verdana" size="2" color="#FFFFFF">
Transaction Successful </FONT></TD></TR>
</table>
<br>
</body>
</btt:html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

- Create a new folder, `withdrawalBP`. In the new folder, create JSPs `withdrawalList.jsp` and `errorpage.jsp`.

  For withdrawalList.jsp:

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

<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<btt:html>
<HEAD>
<title>Demo Bank - Transactions Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<body bgcolor="#3366FF" text="#FFFFFF"
background="/BaseSampleWeb/jsp/images/back_interior.gif">
<FONT size="5" color="#FFFFFF" face="Arial">Demo Bank - Transactions
Page</FONT>
<script language="JavaScript">
function submitAction(actionName){
withdrawalBPForm.action=actionName;
withdrawalBPForm.submit();

}

</script>
<br>
<p>
<btt:form action="/withdrawalBP">
<center>
<table border="1" cellspacing="1" cellpadding="1" width=60%>
  <TR>
    <TD height="40" bgcolor="#002184" align="center"
colspan="4"><FONT face="Verdana" size="2"
```

```
color="#FFFFFF"><STRONG>Customer Name: </STRONG>
    <btt:label dataName="CustomerName"/>
</FONT></TD>
  </TR>
  <TR>
    <TD height="35" bgcolor="#002184" width="35%" align="left"><FONT
face="Verdana" size="2" color="#FFFFFF"><STRONG>Account
Number</STRONG></FONT></TD>
    <TD bgcolor="#0A35B8" align="right"><FONT face="Verdana" size="2"
color="#FFFFFF">
    <btt:combo dataName="AccountNumber" dataNameForList="accounts"
value="AccountNumber" item="Name"/>
      </FONT></TD>
  </TR>
  <TR>
    <TD height="35" bgcolor="#002184" width="35%" align="left"><FONT
face="Verdana" size="2"
color="#FFFFFF"><STRONG>Amount</STRONG></FONT></TD>
    <TD bgcolor="#0A35B8" align="right"><FONT face="Verdana" size="2"
color="#FFFFFF">
    <btt:text property="Amount"/></FONT></TD>
  </TR>

</table>
<table align="center" border="0" cellspacing="0" cellpadding="0"
width=60%>
<TR align="center" valign="baseline">
<TD align="center"><input type=button align="middle" value="Deposit"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/depositBP/deposi
tBP.do');"></TD>
<TD align="center"><input type=button align="middle"
value="Withdrawal"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/withdrawalBP/wit
hdrawalBP.do');"></TD>
<TD align="center"><input type=button align="middle" value="Account
Statement"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/accountStatement
BP/accountStatementBP.do');"></TD>
<TD align="center"><input type=button align="middle" value="Cancel"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/endSession/endSe
ssion.do');" ></TD>
</TR>
</table>
</center>
</btt:form>

<br>
<br>
<table>
```

```
<TR> <TD align=left><FONT face="Verdana" size="2"
color="#FFFFFF">Balance: </FONT></TD>
<TD align=left><FONT face="Verdana" size="2"
color="#FFFFFF"><btt:label
dataName="AccountBalance"/></FONT></TD></TR>
<TR> <TD align=left><FONT face="Verdana" size="2"
color="#FFFFFF">Message: </FONT></TD>
<TD align=left><FONT face="Verdana" size="2" color="#FFFFFF">
Transaction Successful </FONT></TD></TR>
</table>
<br>
</body>
</btt:html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

- Create a new folder, `accountStatementBP`. In the new folder, create JSPs `accountStatementList.jsp` and `errorpage.jsp`.

For accountStatementList.jsp:

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

<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<btt:html>
<HEAD>
<title>Demo Bank: accountStatement</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<body bgcolor="#3366FF" text="#FFFFFF"
background="/BaseSampleWeb/jsp/images/back_interior.gif" >
<FONT size="5" color="#FFFFFF" face="Arial">Demo Bank - Account
Statement</FONT>
<script language="JavaScript">
```

```
function submitAction(actionName){
accountstatementForm.action=actionName;
accountstatementForm.submit();

}
</script>
<btt:form action="/accountStatementBP">


<br>
<center>
<input type=hidden name=CustomerId value="123" SIZE=15>
<btt:table dataNameForList="accountStatementDetails"
headers="{OpnDate,OpnDescription,OpnAmount,OpnBalance}"  border="1"
cellspacing="1" cellpadding="0" width="70%" headerBGColor="#002184"
colAlignments="left,center,center,center" showHeaders="yes"
headerAlignment="center"/>

<br>
<br>
<br>
<br>
<input type=button value="Previous"
onClick="javascript:submitAction('/BaseSampleWeb/jsp/customerSearchBP
/customerSearchBP.do');">
</center>
</btt:form>
</body>
</btt:html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

- Create a new folder, `endSession`. In the new folder, create JSPs `logoffPage.jsp` and `errorpage.jsp`.

  For logoffPage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@page import = "java.util.ResourceBundle"%>
<html>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Studio">
</HEAD>
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<body BGCOLOR="#EEEEEE">

<p>
<p>
logoff success!
</body>
</html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

- In the jsp folder, create `doubleClickError.jsp` and `errorpage.jsp`.

  For doubleClickError.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<head>
<title>Demo Bank - Out of order</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">

<h3 align=center><btt:label text="Double click Error"/></h3>
<br>
<br>
<P>We apologize ,you double click or reload.</P>
```

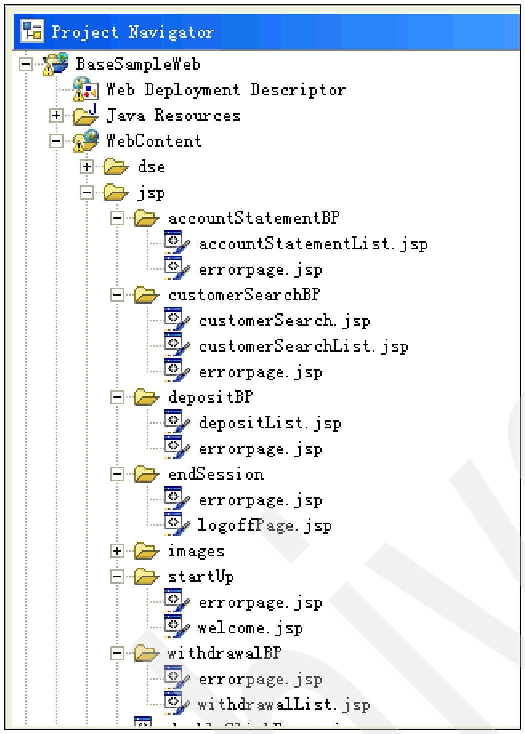
```
<table>
<tr>
<td align="center">

</td>

</tr>
</table>

<br>
<br>
<br>
<center>
<table border=0 cellpadding=0 cellspacing=0>
<tr>
<td align=center>

</td>
</tr>
</table>
</center>
</body>
</html>
```

For errorpage.jsp:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Message Page</title>
<META name="GENERATOR" content="IBM WebSphere Studio">
</head>
<body BGCOLOR="#EEEEEE">
<h3 align=center>System Message</h3>
<P></P>
<P>You may go back and correct the application. If you have any
question, please call <B>99-9-999-999-9999</B> and we will be happy
to help you to take your application.</P>
</body>
</html>
```

The image file back_interior.gif can be found in the DSE_SampleApplicationWeb project in the Branch Transformation Toolkit 4.3 workspace, WebContent/dse/html folder.

The overall file structure should look as shown in Figure 6-49 on page 197.



```
Project Navigator
BaseSampleWeb
    Web Deployment Descriptor
    Java Resources
    WebContent
        dse
        jsp
            accountStatementBP
                accountStatementList.jsp
                errorpage.jsp
            customerSearchBP
                customerSearch.jsp
                customerSearchList.jsp
                errorpage.jsp
            depositBP
                depositList.jsp
                errorpage.jsp
            endSession
                errorpage.jsp
                logoffPage.jsp
            images
            startUp
                errorpage.jsp
                welcome.jsp
            withdrawalBP
                errorpage.jsp
                withdrawalList.jsp
```

Figure 6-49 File structure for BaseSampleWeb

3. Create the ActionForms and Validators.

   Create a package named `com.ibm.btt.samples.html` in the Java Resources folder of the BaseSampleWeb project.

   In the com.ibm.btt.samples.html package, create the following classes:

   – **StartUpForm.java**
   – **CustomerSearchForm.java**
   – **CustomerSearchXVal.java**
   – **DepositBPForm.java**
   – **DepositXVal.java**
   – **WithdrawalBPForm.java**
   – **WithdrawalXVal.java**
   – **AccountStatementForm.java**
   – **EndSessionForm.java**

   Their contents can be found in the Appendix (Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and B.1, “Setting up the Java sample application” on page 494).

4. Create Struts Actions.

   Create a package named `com.ibm.btt.struts.actions` in the Java Resources folder of the BaseSampleWeb project.

   In the com.ibm.btt.struts.actions package, create the following classes:

   – **CustomerSearchAction.java**
   – **EJBLogOffAction.java**

   Their contents can be found in the Appendix (Appendix A, "Branch Transformation Toolkit development and runtime requirements" on page 477 and B.1, "Setting up the Java sample application" on page 494).

5. Modify the invoker.

   Modify the invoker classes according to Table 6-3, in the Java Resources folder of the BaseSampleWeb project. Their contents can be found in the Appendix (Appendix A, "Branch Transformation Toolkit development and runtime requirements" on page 477 and B.1, "Setting up the Java sample application" on page 494).

Table 6-3 BaseSampleWen invoker classes

Package	File
com.ibm.dse.samples.appl.invoker.html	startupHtmlServerOpInvoker.java endSessionServerOpInvoker.java
com.ibm.dse.samples.appl.invoker.java	startupServerOpInvoker.java endSessionServerOpInvoker.java
customerSearchServerOp.invoker.java	customerSearchServerOpInvoker.java
depositServerOp.invoker.java	depositServerOpInvoker.java
withdrawalServerOp.invoker.java	withdrawalServerOpInvoker.java
accountStatementServerOp.invoker.java	accountStatementServerOpInvoker.java
com.ibm.btt.cs.invoker.base	BeanInvokerRegistryMapper.properties

6. Modify the project configuration file.

   Expand the **BaseSampleWeb** project. Locate the web.xml file in the WebContent/WEB-INF folder. Open it in the XML Editor and add the following definitions. Add initial parameters at the end of the servlet definition with the name `Action` before the line: `<load-on-startup>2</load-on-startup>`.

```
<init-param>
  <param-name>config/jsp/startUp</param-name>
  <param-value>/WEB-INF/struts-config_startUp.xml</param-value>
</init-param>
<init-param>
```

```
  <param-name>config/jsp/customerSearchBP</param-name>
  <param-value>/WEB-INF/struts-config_customerSearchForBP.xml</param-value>
</init-param>
<init-param>
  <param-name>config/jsp/withdrawalBP</param-name>
  <param-value>/WEB-INF/struts-config_withdrawalForBP.xml</param-value>
</init-param>
<init-param>
  <param-name>config/jsp/depositBP</param-name>
  <param-value>/WEB-INF/struts-config_depositForBP.xml</param-value>
</init-param>
<init-param>
  <param-name>config/jsp/accountStatementBP</param-name>

<param-value>/WEB-INF/struts-config_accountstatementForBP.xml</param-value>
</init-param>
<init-param>
  <param-name>config/jsp/endSession</param-name>
  <param-value>/WEB-INF/struts-config_endSession.xml</param-value>
</init-param>
```

Add a new servlet definition with the name `StartServerServlet` after the first servlet definition.

```
<servlet>
  <servlet-name>StartServerServlet</servlet-name>
  <display-name>StartServerServlet</display-name>

<servlet-class>com.ibm.btt.samples.appl.StartServerServlet</servlet-class>
  <load-on-startup>-1</load-on-startup>
</servlet>
```

Add a new environment entry named `dseIniPath` at the end of the tag-lib list.

```
<env-entry>
  <env-entry-name>dseIniPath</env-entry-name>
  <env-entry-value>c:\\dse\\dse.ini</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

The resulting web.xml file can be found in the Appendix (Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493).

7. Create WSDL definitions.

   Create a folder named `wsdl`in the WebContent folder of the BaseSampleWeb project. See Figure 6-50 on page 200

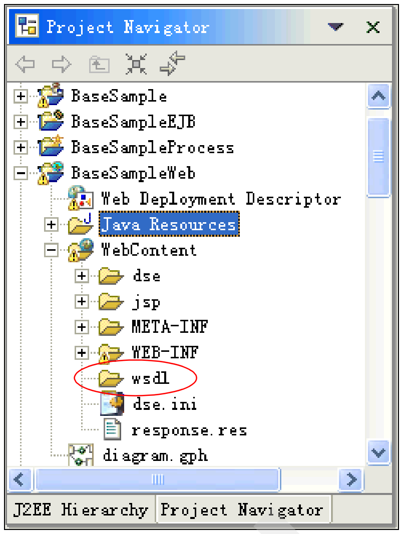


Figure 6-50 WSDL folder for BaseSampleWeb

Copy the definition files from the BaseSampleProcess project to this folder according to Table 6-4.

Table 6-4 Definition files needed for BaseSampleWeb

Source folder	File name
customerSearchServerOp	BTTSystemData.xsd cha.wsdl customerSearchServerOp.wsdl customerSearchServerOp_ProcessPortType_EJB.wsdl customerSearchServerOpInterface.wsdl
DepositServerOp	depositServerOp.wsdl depositServerOp_ProcessPortType_EJB.wsdl depositServerOpInterface.wsdl
withdrawalServerOp	withdrawalServerOp.wsdl withdrawalServerOp_ProcessPortType_EJB.wsdl withdrawalServerOpInterface.wsdl

Source folder	File name
accountStatementServerOp	accountStatementServerOp.wsdl accountStatementServerOp_ProcessPort Type_EJB.wsdl accountStatementServerOpInterface.wsd l

The resulting file structure in this folder should look as shown in Figure 6-51.

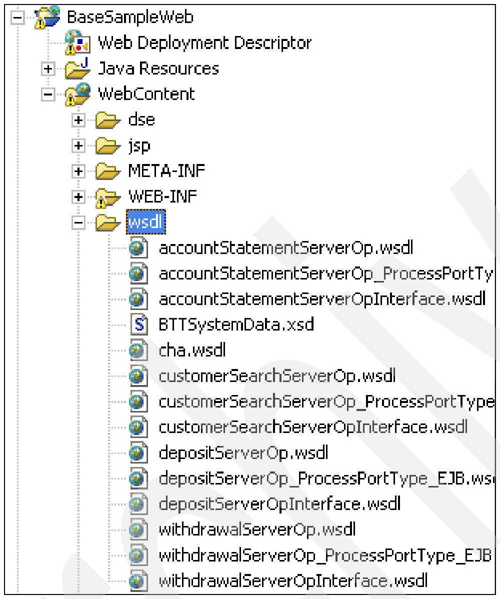

BaseSampleWeb
Web Deployment Descriptor
Java Resources
WebContent
dse
jsp
META-INF
WEB-INF
wsdl
accountStatementServerOp.wsdl
accountStatementServerOp_ProcessPortTy
accountStatementServerOpInterface.wsdl
BTTSystemData.xsd
cha.wsdl
customerSearchServerOp.wsdl
customerSearchServerOp_ProcessPortType
customerSearchServerOpInterface.wsdl
depositServerOp.wsdl
depositServerOp_ProcessPortType_EJB.ws
depositServerOpInterface.wsdl
withdrawalServerOp.wsdl
withdrawalServerOp_ProcessPortType_EJB
withdrawalServerOpInterface.wsdl

Figure 6-51 Completed WSDL folder for BaseSampleWeb

8. Create the Struts definition files.

   In the WebContent/WEB-INF folder of the BaseSampleWeb project, create the following definition files.

   – **struts-config_startUp.xml**
   – **struts-config_customerSearchForBP.xml**
   – **struts-config_depositForBP.xml**
   – **struts-config_withdrawalForBP.xml**
   – **struts-config_endSession.xml**
   – **struts-config_accountStatementForBP.xml**

   Their contents can be found in the Appendix (Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 and Appendix B, “Setting up a Branch Transformation Toolkit sample application” on page 493).

The resulting file structure in the WEB-INF folder should look as shown in Figure 6-52.

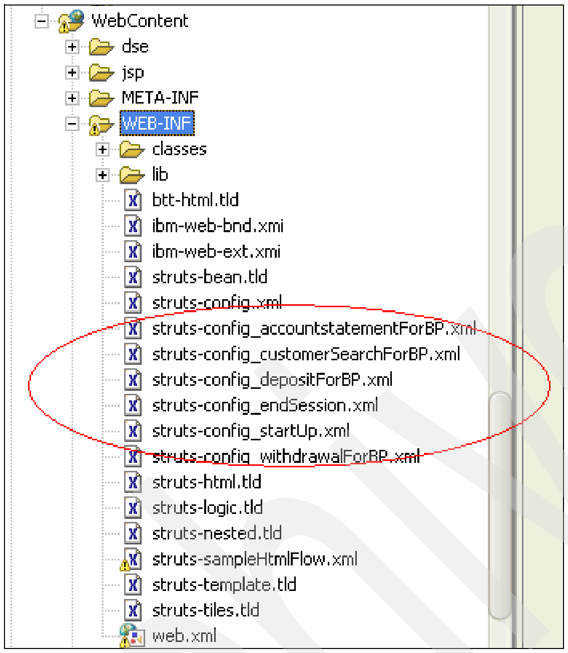

WebContent
dse
jsp
META-INF
WEB-INF
classes
lib
btt-html.tld
ibm-web-bnd.xmi
ibm-web-ext.xmi
struts-bean.tld
struts-config.xml
struts-config_accountstatementForBP.xml
struts-config_customerSearchForBP.xml
struts-config_depositForBP.xml
struts-config_endSession.xml
struts-config_startUp.xml
struts-config_withdrawalForBP.xml
struts-html.tld
struts-logic.tld
struts-nested.tld
struts-sampleHtmlFlow.xml
struts-template.tld
struts-tiles.tld
web.xml

Figure 6-52 WEB_INF folder for BaseSampleWeb

Java client

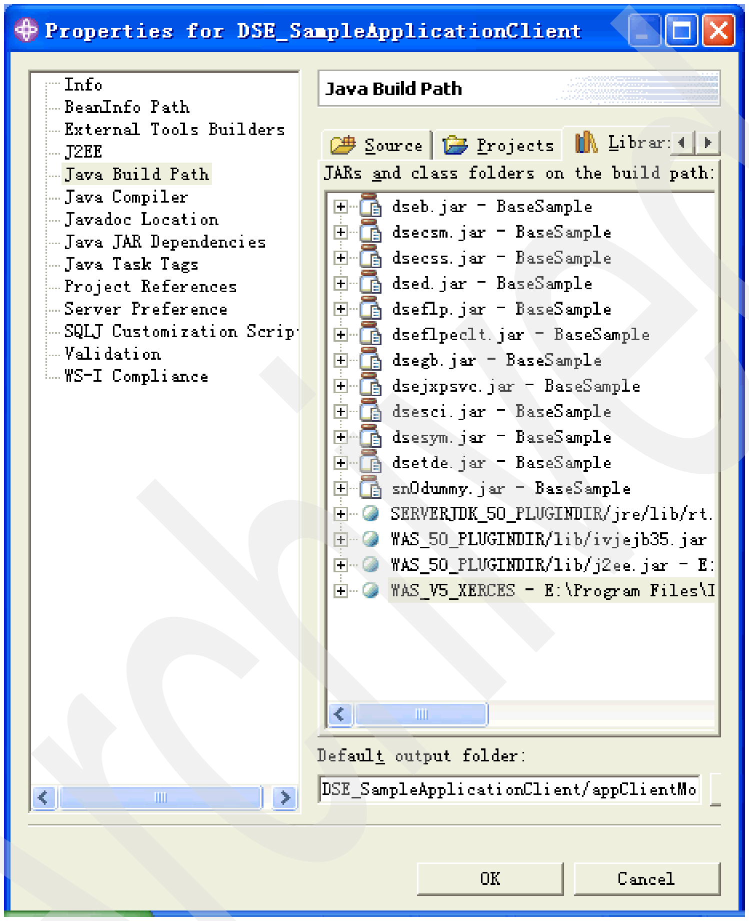
1. To configure the build path of the DSE_SampleApplicationClient project, right-click the project and open its **Properties** window. Select **Java Build Path** from the left panel.

   In the Libraries tab, click **Add JARs**. Expand the **BaseSample** project and select the following JAR files:

   – **dseb.jar**
   – **dsecsm.jar**
   – **dsecss.jar**
   – **dsed.jar**
   – **dseflp.jar**
   – **dseflpeclt.jar**
   – **dsegb.jar**
   – **dsejxpsvc.jar**
   – **dsesci.jar**

– **dsesym.jar**
– **dsetde.jar**

In the Libraries tab, click **Add Variable**. Choose **WAS_V5_XERCES** from the list and click **OK**. See Figure 6-53



Properties for DSE_SampleApplicationClient

Java Build Path

JARs and class folders on the build path:

```
dseb.jar - BaseSample
dsecsm.jar - BaseSample
dsecss.jar - BaseSample
dsed.jar - BaseSample
dseflp.jar - BaseSample
dseflpeclt.jar - BaseSample
dsegb.jar - BaseSample
dsejxpsvc.jar - BaseSample
dsesci.jar - BaseSample
dsesym.jar - BaseSample
dsetde.jar - BaseSample
sn0dummy.jar - BaseSample
SERVERJDK_50_PLUGINDIR/jre/lib/rt.
WAS_50_PLUGINDIR/lib/ivjejb35.jar
WAS_50_PLUGINDIR/lib/j2ee.jar - E:
WAS_V5_XERCES - E:\Program Files\I
```

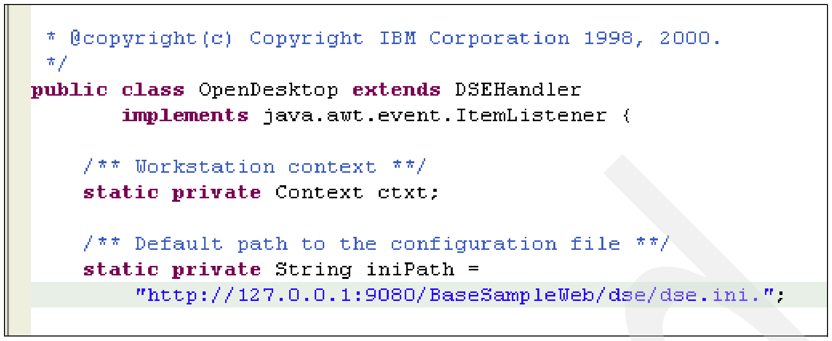
Default output folder:

DSE_SampleApplicationClient/appClientMo

Figure 6-53 Java build path for client application

2. Modify the OpenDesktop.java class in the com.ibm.dse.samples.appl package of the DSE_SampleApplicationClient project. Change the initial value of the variable iniPath to http://127.0.0.1:9080/BaseSampleWeb/dse/dse.ini.

   See Figure 6-54 on page 204

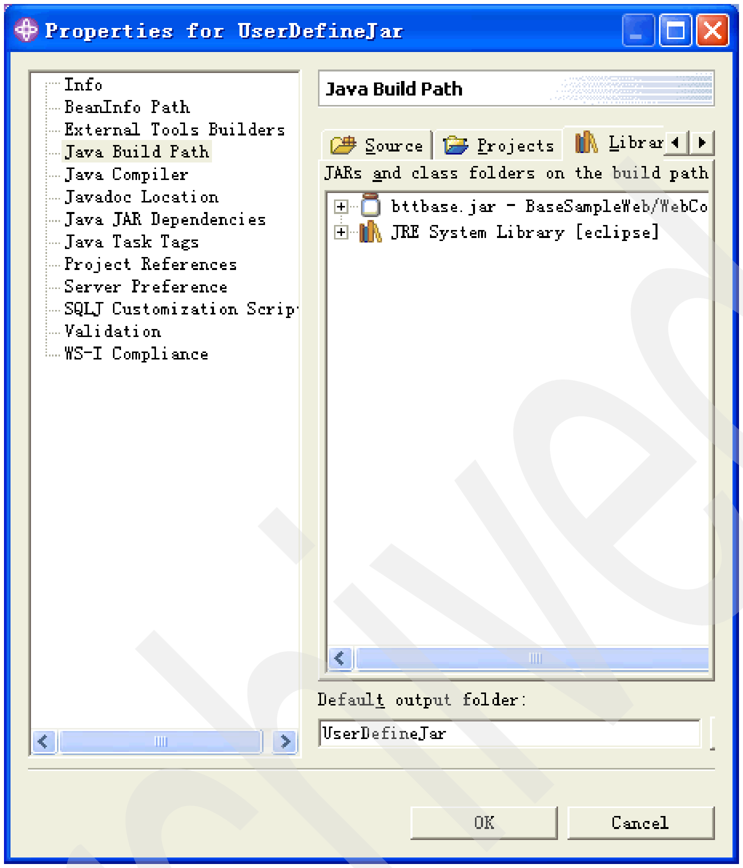


```
 * @copyright(c) Copyright IBM Corporation 1998, 2000.
 */
public class OpenDesktop extends DSEHandler
       implements java.awt.event.ItemListener {

    /** Workstation context **/
    static private Context ctxt;

    /** Default path to the configuration file **/
    static private String iniPath =
        "http://127.0.0.1:9080/BaseSampleWeb/dse/dse.ini.";
```

Figure 6-54 Set configuration file path

3. Configure the UserDefineJar project. Right-click the **UserDefineJar** project to bring its Properties window.

   Select **Java Build Path** from the left panel. In the Libraries tab, click **Add JARs**.In the pop-up window, expand the **BaseSample** project to choose the **bttbase.jar** file. Click **OK** to add. See Figure 6-55 on page 205.


Properties for UserDefineJar

- Info
- BeanInfo Path
- External Tools Builders
- Java Build Path
- Java Compiler
- Javadoc Location
- Java JAR Dependencies
- Java Task Tags
- Project References
- Server Preference
- SQLJ Customization Scrip
- Validation
- WS-I Compliance

Java Build Path

Source | Projects | Librar

JARs and class folders on the build path

- bttbase.jar - BaseSampleWeb/WebCo
- JRE System Library [eclipse]

Default output folder:

UserDefineJar

Figure 6-55   Add JARs to build path for client application

Modify the HostField.java class in the com.ibm.dse.samples.appl package. Change the following line:

```
import com.ibm.dse.base.*;
```

The line should appear as:

```
import com.ibm.btt.base.*;
```

Add two methods in this class:

```
public void writeExternal(java.io.ObjectOutput s) throws
java.io.IOException {
  super.writeExternal(s);

  s.writeUTF(hostIdentifier);

}
```

```
/**
 * Invokes the object creation from an ObjectInput.
 * @param s java.io.ObjectInput
 * @exception java.io.IOException.
 * @exception java.lang.ClassNotFoundException.
 */
public void readExternal(java.io.ObjectInput s) throws java.io.IOException,
java.lang.ClassNotFoundException {

  super.readExternal(s);

  //hostIdentifier
  hostIdentifier = s.readUTF();
}
```

Modify the HostDecorator.java class in the com.ibm.dse.samples.appl package. Change the following line:

```
import com.ibm.dse.base.*;
```

It should appear as follows:

```
import com.ibm.btt.base.*;
```

Modify the HostStringFormat.java class in the com.ibm.dse.samples.appl package. Change the following line:

```
import com.ibm.dse.base.*;
```

It should appear as follows:

```
import com.ibm.btt.base.*;
```

Change the definition of the method formatField( ) in this class:

```
public String formatField( DataField aDataField) throws
com.ibm.btt.base.DSEInvalidClassException { ... }
```

Surround the statement `aDataField.setValue(tmpString);` with try/catch block as follows:

```
 try {
   aDataField.setValue(tmpString);
 } catch (Exception e) {
   e.printStackTrace();
 }
```

4. Add the Base Sample Application to the test environment.

   Since you use the WSAD test environment to deploy the BaseSample server, add the Base Sample Application.

   Change to the Server perspective. Right-click the **BaseSample** server in the Server Configuration panel. Select **Add and remove projects** in the pop-up menu. In the next window, select **Base Sample** project from the list in the left panel, and click **Add** to configure it on the server. Click **Finish**. See Figure 6-56

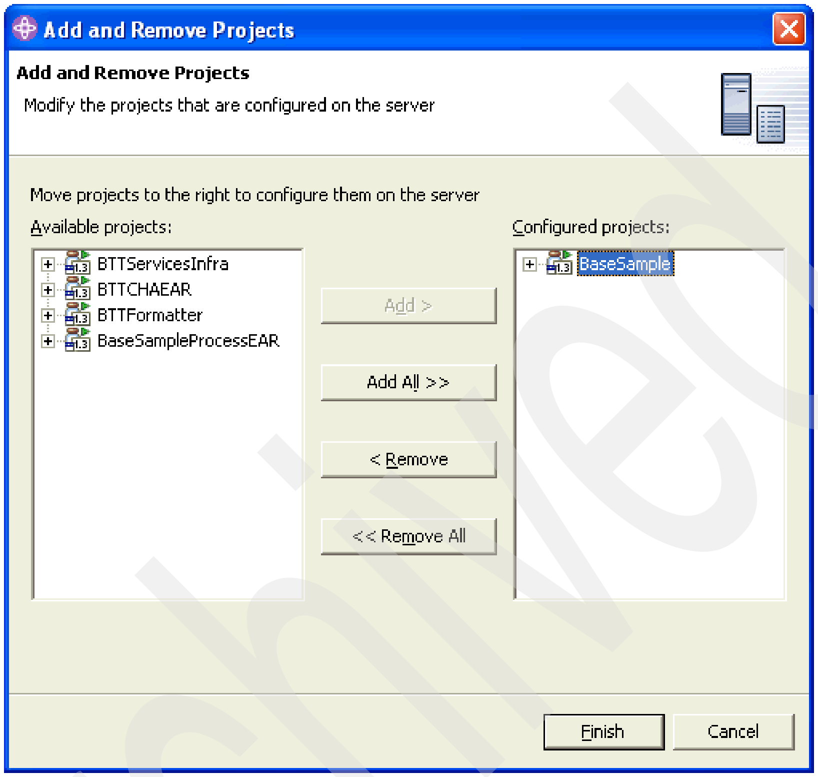


Figure 6-56 Add project to test server

7

Testing and deployment

This chapter discusses testing the migrated Base Sample application. In our sample, we used two types of clients to access the services and operations provided by the server, one a pure Java client and the other an HTML client. After testing, you should deploy the newly created application. Branch Transformation Toolkit 5.1 supports two kinds of platforms; WebSphere Application Server and)WebSphere Business Integration Server Foundation. Depending on what features your application uses you need to decide what platform to which to deploy.

Base on the features of our sam,ple application, we deploy it on WebSphere Business Integration Server Foundation.

This chapter discusses the following topics:

7.1 Configuring the test environment

Although the migration work is nearly compete, before using the migrated project, the application should be tested, deployed, and run. In this phase, the server is created and some configuration carried out.

CHA provides context functionality in a distributed server environment. To fulfil the functionality of the CHA in Branch Transformation Toolkit 5.1 applications, a database server is also required. Although most kinds of database systems are supported as described in Chapter 5, "Migrating an application", IBM DB2 Universal Database 8.2 is used as the default in this book.

In our sample, the WebSphere Studio Application Developer Integration Edition v5.1.1 test environment was used as the runtime platform and DB2 Universal 8.2 was used as the database server.

7.1.1 Preparing the CHA database

During the migration process, you created a database named *SAMPLE*. Check the database status before proceeding. If you created the database correctly, skip this phase. Otherwise, you have to create it again or fix it. For detailed information, refer to 5.3.2, "Creating a CHA database" on page 116.

7.1.2 Creating a server

In 6.3, "Preparing the client and the server" on page 132, you created the server named `BaseSample`. Check the server status before proceeding. For this configuration, the security and database information is required. If you have done this correctly, skip this step. Otherwise, reorganize the server again. For information about the detailed method to do this, refer to 6.3.3, "Creating a server" on page 143.

Once you finish this, deploy CHA to fulfil the EJB to RDB Mapping functionality. If you have already done this correctly following the instruction given in "CHA deployment" on page 151, you can skip this step. Otherwise, you have do it again to fix any errors.

7.1.3 Adding application to the test environment

Because you will be publishing the BaseSample application by deploying the BaseSample server, add the application to the server, by performing the following steps:

1. Change to **Server** perspective.

2. Right-click the **BaseSample** server in the Server Configuration panel.
3. Select **Add and remove projects** in the pop-up menu.
4. In the next window, select the **Base Sample** project from the list in the left panel, and click **Add** to configure it on the server.
5. Click **Finish**.

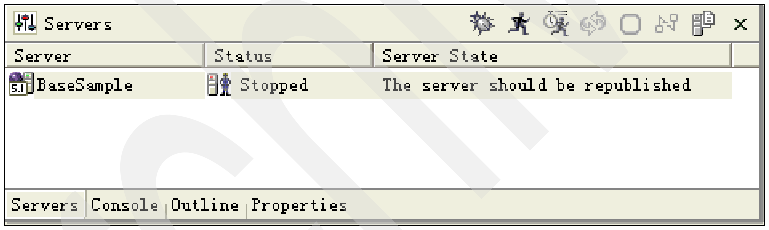
After doing this, you must check to see if errors exist. You must correct any errors so that the server can launch successfully.

7.1.4 Launching the migrated application

Wjem everything we have described so far is completed, launch the server for clients to use it.

In the Server perspective, select the server name **BaseSample** in the **Servers** panel. Click the **running-man** icon to start the server, as shown in Figure 7-1.

The response messages can be found in the console. Use the information to check the server's status.



Server	Status	Server State
BaseSample	Stopped	The server should be republished

Figure 7-1 Launch the test server

7.2 Java client

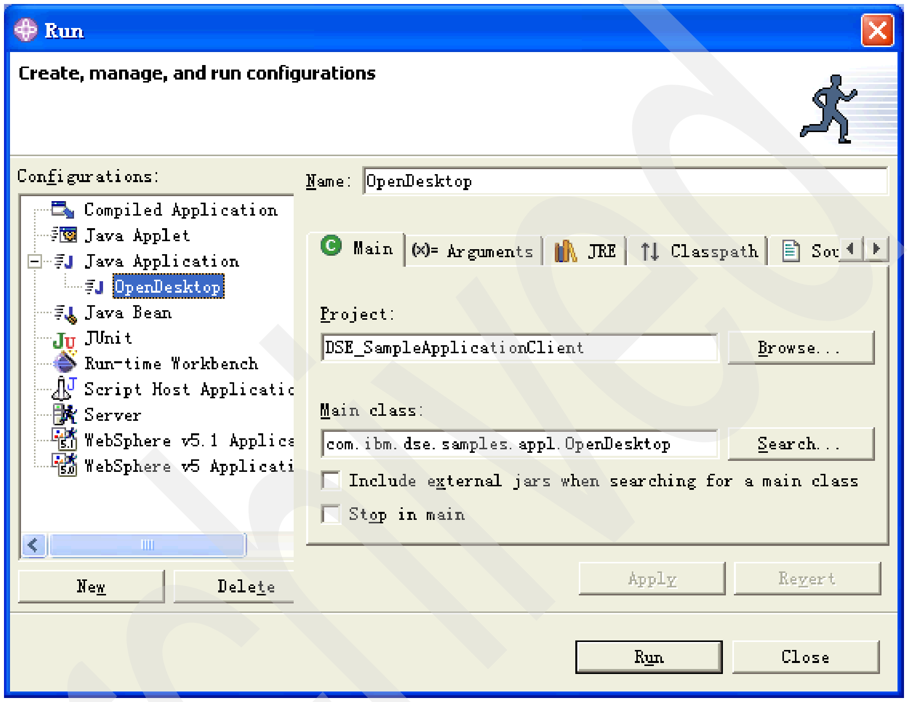
The migrated project supports pure Java access. By creating a Java client application, you can access the applications services and operations and test whether the migrated application works.

Since you have created the client application during the migration process, you only need to locate and start the client by performing the following tasks:

1. From the main menu select **Run** → **Run...**. In the next window, select **Java Application** from the list in the left panel and click **New**.

2. Locate the client program.

   In the right panel, click **Browse** to select **DSE_SampleApplicationClient** as the project, and click **Search** to choose **com.ibm.dse.samples.appl.OpenDesktop** as the main class, as shown in Figure 7-2.



Name: OpenDesktop

Project: DSE_SampleApplicationClient

Main class: com.ibm.dse.samples.appl.OpenDesktop

☐ Include external jars when searching for a main class

☐ Stop in main

Figure 7-2 Run Java application

3. Start the client program by clicking **Apply** and **Run** to bring up the client. Figure 7-2 shows how to start the client.

4. Click **Customer search** and input a value for Customer ID to start testing. If BaseSample works, you will get the correct response. Figure 7-3 shows the welcome screen of the application.

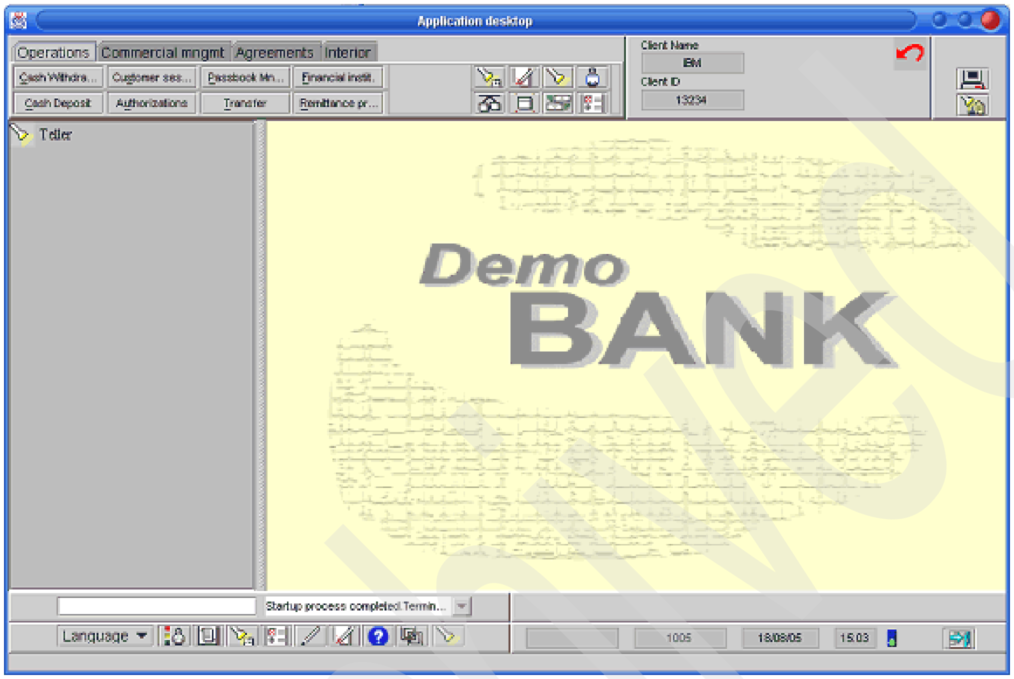


Figure 7-3 Application welcome window

7.3 HTML client

The migrated project also supports Web-based access, so that users can access the application services and operations of server through Web browser.

For this sample, you can use Internet Explorer to test BaseSample after deploying and starting the server.

1. Make sure that you have started the BaseSample properly. If not, do so as described in 7.1, “Configuring the test environment” on page 210.
2. Open an Internet Explorer window and navigate to the following URL:

   `http://127.0.0.1:9080/BaseSampleWeb/jsp/startUp/startUp.do`

3. The main page is displayed.Click **Submit**. Input the value `user01` for both Customer Name and Password to try other transactions.

4. Depending the operation you are testing, you will get a response indicating that the BaseSample server works.

7.4 Application deployment

In order to be able to use the migrated project, deploy it and run it on a real application server platform.

For it’s Web application server, Branch Transformation Toolkit 5.1 supports both the WebSphere Application Server and WebSphere Business Integration Server Foundation platforms. You can select your deployment platform according to the requirements of your application. Figure 7-4 on page 214 shows the features and benefits provided by the different servers.

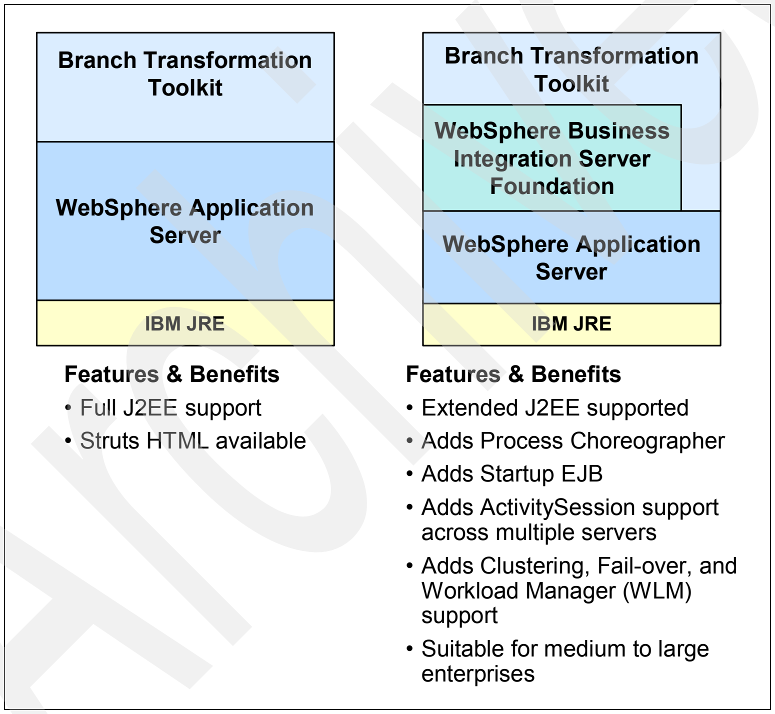

Branch Transformation Toolkit
WebSphere Application Server
IBM JRE
Features & Benefits
• Full J2EE support
• Struts HTML available
Branch Transformation Toolkit
WebSphere Business Integration Server Foundation
WebSphere Application Server
IBM JRE
Features & Benefits
• Extended J2EE supported
• Adds Process Choreographer
• Adds Startup EJB
• Adds ActivitySession support across multiple servers
• Adds Clustering, Fail-over, and Workload Manager (WLM) support
• Suitable for medium to large enterprises

Figure 7-4 Branch Transformation Toolkit runtime environments

Some features such as the Struts BTT Extensions and BPEL, are not supported by the WebSphere Application Server. If the application includes these features, you should use WebSphere Business Integration Server Foundation for your deployment runtime.

You can deploy our sample application on any of the server platforms supported by the Branch Transformation Toolkit. The following procedure describes how to install and run the BaseSample application on the WebSphere Application Server.

Setting up an application on WebSphere Application Server

To set up an application on WebSphere Application Server, follow these steps;

1. Copy external files.

   If you are using the Windows platform, create a directory called `c:\dse` on the server system. Copy all the dse files into this directory, including the configuration files and data files.

   If your operating system is UNIX or Linux, do the following:

   a. Open **BaseSampleWeb**, modify **Web Deployment Descriptor**, set the dseIniPath value to `/dse/dse.ini`.

   b. Open **BTTCHAEJB**, modify **ejb-jar.xml**, and set the dseIniPath value to `/dse/dse.ini`.

2. Create database and tables.

   a. Run the following in the DB2 command window to create a database named `sample`*:*

   ```
   DB2 CREATE DATABASE SAMPLE
   ```

   You can set the user and password for the database SAMPLE, for example, set both user and password as `db2admin`.

   b. Create three tables.

   i. Create a directory called `/temps`.

   ii. Copy **%install root%\dbtools\Windows\DB2\tableDefinition\cha\createCHATables.ddl to directory \temps**.

   Note: If your operating system is UNIX or Linux, copy **%install root%/dbtools/Unix/DB2/tableDefinition/cha/createCHATables.ddl** to the directory /temps.

   iii. Go to the /temps directory and open the DB2 command window.

   iv. In the DB2 command window, run:

   ```
   DB2 CONNECT TO SAMPLE USER db2admin USING db2admin
   ```

   v. In the DB2 Command Window, run:

   ```
   db2 -tvf createCHATables.ddl
   ```

You will see messages indicating that CHAChildren, CHAInstance, and CHAControl tables have been created successfully.

3. Open a Web browser and input the following URL to open the Administrative console:

   `http://serverName:9090/admin`

4. Import BaseSample.ear into WebSphere Application Server.

   a. In the navigation bar, select **Applications** → **Install new Application**.

   b. In the main frame, click **Browse** and locate the BaseSample.ear file. Click **Next**.

   c. In the Step2 frame, specify the DataBase Type.

   d. In the Deploy EJBs Option - Database Type field, use the default value. For DB2, select **DB2UDB_V81**.

   e. In the Step4 frame, input `jdbc/CHADataSource` as JNDI Name. Use default value for other settings

   f. In the setp10 frame, click **Finish**.

   g. Click **Save** to complete the Master Configuration.

   h. Click **Save** to finish importing BaseSample.ear.

5. Set up JDBC Providers.

   a. In the navigation bar, select **Resource** → **JDBC Providers**.

   b. In the main frame, click **New**.

   c. In the JDBC Providers field, select the **(XA) JDBC Provider**. For other settings, use the default value.

   d. Click **OK** to return to the JDBC Providers page.

   e. Click the **JDBC Provider Name**.

   f. In Additional Properties section, click **Data Source**.

   g. Click **New**.

   h. Input the value `jdbc/CHADataSource` in the JNDI Name field and select the option **Use this Data Source in container managed persistence (CMP)**. Click **OK** to return to the Data Sources page.

   i. Click the **Data Source Name**.

   j. In the Related Items section, click **J2C Authentication Data Entries**.

   k. Click **New** to include the following settings:

      - Alias: `CHA`
      - User ID: `db2admin`
      - Password: `db2admin`

l. Click **OK** to save the values.

m. In the navigation bar, select **Resource** → **JDBC Providers** → **JDBC Provider Name** → **Data Source**.

n. Click the **Data Source Name**.

o. Select a value for the Component-managed Authentication Alias and Container-managed Authentication Alias fields. Click **OK** to return to the Data Sources page.

p. Click **Data Source Name**.

q. In the Additional Properties section, click **Custom Properties**.Set the databaseName and serverName. Use the default value for other settings, click **Save** on top of the page.

r. In the navigation bar, select **Resource** → **JDBC Providers** → **JDBC Provider Name** → **Data Source**. Select the **Data Source** and click **Test Connection**. You will see some messages indicating that Test Connection was successful.

6. Start the Applications.

   a. In the navigation bar, select **Applications** → **Enterprise Applications**.

   b. Select **BaseSampleEAR** and click **Start**.

   c. Open a Web browser, and input the following URL to run the BaseSample:

```
http://serverName:9080/BaseSampleWeb/btt/html/sign/prepareSignIn.do
```

Install on WebSphere Business Integration Server Foundation

If you did a typical installation of WebSphere Business Integration Server Foundation and do not have a sample container, you must configure your business process container manually.

Note: To check whether you have a business process container configured on your server, from the administrative console navigation pane, select **Applications** → **Enterprise Applications**. Your process container is available if an application named BPEContainer_<hostName>_<serverName> appears in the list of enterprise applications.

If you carried out a custom installation and choose to configure a sample business process container in the Installation Wizard, performing the following tasks:

1. In your operating system, create a directory called `/dse`. If the OS is Windows, the directory should be `c:\dse\`. Copy all the necessary files to this directory.

   If your OS is UNIX or Linux(R), do the following:

   a. Open the **BaseSampleWeb** project, modify **Web Deployment Descriptor**, set the dseIniPath value to `/dse/dse.ini` from BaseSampleWeb.

   b. Open the **BTTCHAEJB** project, modify **ejb-jar.xml**, set the dseIniPath value to `/dse/dse.ini` from BTTCHAEJB.

2. Create database and tables.

   a. Run the following in the DB2 command window to create a database named `sample`:

   ```
   DB2 CREATE DATABASE SAMPLE
   ```

   You can set user and password for the database sample, for example, set both user and password as `db2admin`.

   b. Create three tables.

      i. Create a directory called `/temps`.

      ii. Copy **%install root%\dbtools\Windows\DB2\tableDefinition\cha\createCHATables.ddl** to the \temps directory.

      Note: If your operating system is UNIX or Linux, copy **%install root%/dbtools/Unix/DB2/tableDefinition/cha/createCHATables.ddl** to the directory /temps.

      iii. Shift to the /temps directory and open the DB2 command window.

      iv. In the DB2 command window, run:

      ```
      DB2 CONNECT TO SAMPLE USER db2admin USING db2admin
      ```

      v. In the DB2 Command Window, run:

      ```
      db2 -tvf createCHATables.ddl
      ```

      You will see messages indicating that CHAChildren, CHAInstance, and CHAControl tables have been created successfully.

3. Open a Web browser, input the following URL to open the Administrative console:

   ```
   http://serverName:9090/admin
   ```

4. Import BaseSample.ear into WebSphere Business Integration Server Foundation.

   a. In the navigation bar, select **Applications** → **Install new Application**.

   b. In the main frame, click **Browse** and locate the BaseSample.ear file. Click **Next**.

   c. In the Step2 frame, specify the DataBase Type.

      In the Deploy EJBs Option - Database Type field, use the default value. For DB2, select **DB2UDB_V81**.

   d. In the Step4 frame, input **jdbc/CHADataSource** for JNDI Name. Use default value for other settings.

   e. In the setp10 frame, click **Finish**.

   f. Click **Save to Master Configuration**.

   g. Click **Save** to finish importing BaseSample.ear.

5. Set up JDBC Providers.

   a. In the navigation bar, select **Resource** → **JDBC Providers**.

   b. In the main frame, click **New**.

   c. In JDBC Providers field, select **JDBC Provider**. It must be the (XA) JDBC Provider. For other settings, use the default value.

   d. Click **OK** to return to the JDBC Providers page.

   e. Click the **JDBC Provider Name**.

   f. In Additional Properties section, click **Data Source**.

   g. Click **New**.

   h. Input **jdbc/CHADataSource** in the **JNDI Name** field and select the option **Use this Data Source in container managed persistence (CMP)**. Click **OK** to return to the Data Sources page.

   i. Click the **Data Source Name**.

   j. In the Related Items section, click **J2C Authentication Data Entries**.

   k. Click **New** and enter the following:

      - Alias: `CHA`
      - User ID: `db2admin`
      - Password: `db2admin`

   l. Click **OK** to save the values.

m. In the navigation bar, select **Resource** → **JDBC Providers** → **JDBC Provider Name** → **Data Source**.

n. Click **Data Source Name**.

o. Select values for Component-managed Authentication Alias and Container-managed Authentication Alias fields. Click **OK** to return to the Data Sources page.

p. Click the **Data Source Name**.

q. In the Additional Properties section, click **Custom Properties**. Set the databaseName and serverName. Use the default value for other settings, and click **save** on top of the page.

r. In the navigation bar, select **Resource** → **JDBC Providers** → **JDBC Provider Name** → **Data Source**, select the Data Source and click **Test Connection**. You will see some messages indicating that Test Connection was successful.

6. Install DummySnaLu0 in the <toolkit root>\jars directory in the WebSphere Business Integration Server Foundation.

   a. In Websphere Administration console, select **Resources\Resource Adapter\Install RAR** and add the following setting:

   ```
   name: dummysnalu0
   ```

   b. Define JAAS Authentication entries:

      i. Select **Security\JAAS Configuration\J2C Authentication Data**.

      ii. Click **New**.

      iii. Add the following parameters:

      - Alias: `sna`
      - User ID: `sna`
      - Password: `sna`

   c. Define the J2C Connection Factories:

      i. Select **Resources\Resource Adapter\dummysnalu0**.

      ii. Click **J2C Connection Factories**.

      iii. Click **New**.

      iv. Add the following parameters

      - Name: `snalu0`
      - JNDI name: `snalu0`
      - Component-managed Authentication alias: `sna`
      - Container-managed Authentication alias: `sna`

   d. Select **Resources\ Resource Adapter\dummysnalu0\J2C Connection Factories\snalu0**, click **Custom Properties**, and change the following parameters:

      - TestFile: `http://127.0.0.1:9080/BTTHTMLSampleBPWeb/response.res`
      - userName: `sna`
      - userPassword: `sna`

7. Start the applications.

   a. In the navigation bar, select **Applications** → **Enterprise Applications**.

   b. Select **BaseSample.ear** and click **Start**.

   c. Open a Web browser, and input the following URL to run BaseSample.ear:

      `http://serverName:9080/BaseSampleWeb/btt/html/sign/prepareSignIn.do`

Part 2

Development with Branch Transformation Toolkit V5.1

(We need to have some content here. According to ITSO style, we cannot have a Part file with only a headline.)

8

Building an application with Branch Transformation Toolkit V5.1

This chapter introduces the development capabilities and tooling that Branch Transformation Toolkit v5.1 provides. In this chapter, we:

- Give background information required to implement a withdrawal operation.
- Introduce WebSphere Studio Application Developer and WebSphere Studio Application Developer Integration Edition.
- Develop a Struts Web application to demonstrate how to use the CHA Editor, the Format Editor, the Struts Tools BTT Extensions, the Business Process BTT Wizard, and the Graphical Builder.
- Describe Java client development.

This chapter is organized into the following sections:

8.1 Before getting started

This chapter discusses the construction of a sample Branch Transformation Toolkit application. Before starting, you should know the application scenario and other background information.

8.1.1 Application scenario

Withdrawal, deposit, and transfer are typical services used by a bank customer. This chapter talks about developing a withdrawal operation, which involves withdrawing money from a customer account. Such an operation requires information such as:

- The account number from which money will be withdrawn.
- The amount to be withdrawn.
- The withdrawal date.
- The branch from where the withdrawal will be performed.

Additional requirements include the following:

- Maintenance of a daily transaction log.

  This includes developing a journal service that consists of a database schema in which one record is added each time a transaction is sent to the host, and is updated with the host reply.

- Format in which data is sent to the host.

  Data is sent to the host using messages that must follow a set of formatting rules. This constraint should guide you when defining new format elements. The customer provides the information required to define these formats.

  The format of the data to be sent for the withdrawal operation is as follows:

  ```
  Tx02AN=10012002377460000018#DT=28092000#AM=2000#BR=1005#
  ```

  In this format:

  – Tx02: Transaction type
  – AN: Account number
  – DT: Date
  – AM: Amount
  – BR: Branch ID

  Note: Each value ends with the # delimiter.

- Validation of the view input data

  When errors occur, messages notifying this situation must be displayed.

A real-world situation involves a branch employee starting a workstation, inputting and validating the customer's data before a withdrawal operation is performed. In our sample, we provide implementation of these actions, in addition to a customer search view to initialize the environment.

After the environment is started, a graphical interface prompts the branch employee to edit information in the entry fields of the view, such as the customer's account number and the withdrawal amount.

After the required information is supplied, the workstation performs initial validation, requests the data from the system, and data is then sent to the server. The server should do the following:

- Create a journal record.
- Send formatted data to the host.
- Receive a host reply.
- Send a reply to the client process and update the last record with the host reply.

The other services that can be developed to improve this application, although not included in this book, are, printing and delivering a receipt to the client, and updating the cash drawers in order to keep control of the cash amount.

8.1.2 Developing the system specification

Developing the system specification is usually the first step to system development. It minimizes the gap between the application scenario and the implementation.

Specification includes architecture, component interactions, message definition, and internal data structure.

Architecture

For the scenario in our sample, we built a Web application using Branch Transformation Toolkit v5.1 to achieve the goal. A typical architecture of this kind of application includes WebSphere Application Server, DB/2 database server, Host, that is, S/390®, and Web Browser, as shown in Figure 8-1.

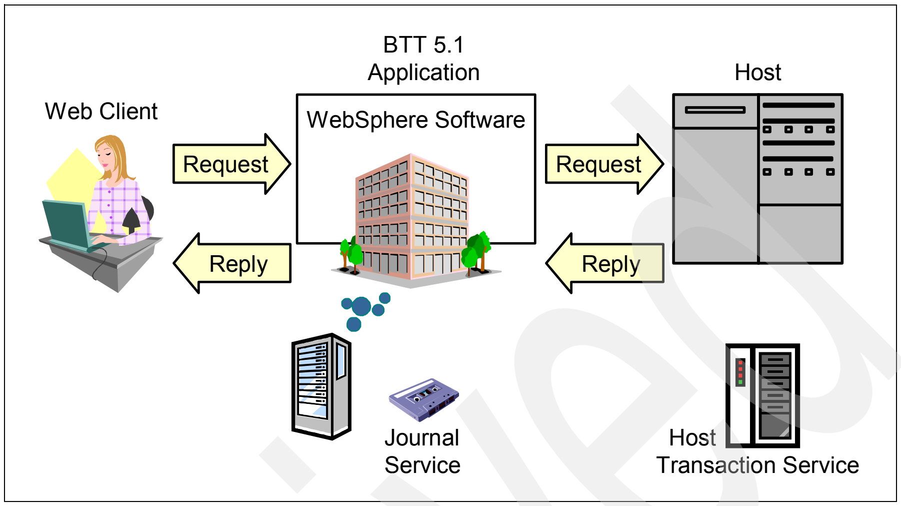


Figure 8-1   The typical architecture of a Branch Transformation Toolkit Web application

In our application, we used two services:

- Journal service

  This service is based on a database and keeps information about the operations performed during a predefined period of time. We simulated this service with a dummy service named DummyJournal. In the second stage, we substituted the DummyJournal with a real journal service that accesses a real DB/2 database.

- Host transaction service

  This service is used to send and receive formatted data to and from a simulated host. We simulated a SNA LU0 protocol with a dummy host communication service. This dummy service was named `DummyLu0SnaSession`.

Component interactions

Here we set up a withdrawal operation. There are three stages to completing a withdrawal operation.

Stage 1

A teller uses a Web browser to connect to a Branch Transformation Toolkit v5.1 application to perform a withdrawal operation. An input form is displayed in the browser, requesting the account number and the amount.

Stage 2

After obtaining the customer's account number and the amount details, the teller inputs these two numbers and submits the request. The application then initializes the transaction, appends Date, Session, and Branch data to withdrawal data (Account Number and Amount). Journal service takes this formatted data and stores it as the log of this transaction in the database.

Stage 3

When the journal service is completed, the application sends the formatted data to the host to execute the real withdrawal operation. The host sends the reply after completing the execution. The result is displayed in the Web browser.

Message definition

When you understood fully the functionality of the application, you should determine the required host transaction messages and the data intended for the electronic journal service.

Host transaction messages

The host transaction messages are:

- Requests sent to the host include the following:
  - A static transaction header code, for example, Tx02
  - Account number
  - Date
  - Amount
  - Branch identifier
- Replies from the host include the following:
  - Reply code
  - Updated account balance
  - Error message

Electronic journal messages

The electronic journal messages include:

- The data that should be stored prior to sending data to the host include the following:
  - User identification
  - Terminal identifier
  - Transaction data being sent to host
- The data that should be stored after receiving the host reply, include the following:
  - Reply code

- Error message
- Updated account balance

Internal data structure

In message definition, you will see that there is a hierarchical structure. In general, a typical message in a real branch application includes branch identifier, terminal identifier, user identification, and so on, as shown in Figure 8-2.

In Figure 8-2, each branch has a branch identifier. Terminal identifier and user identification are associated with the workstation. Other data belongs to the customer and the operation itself.

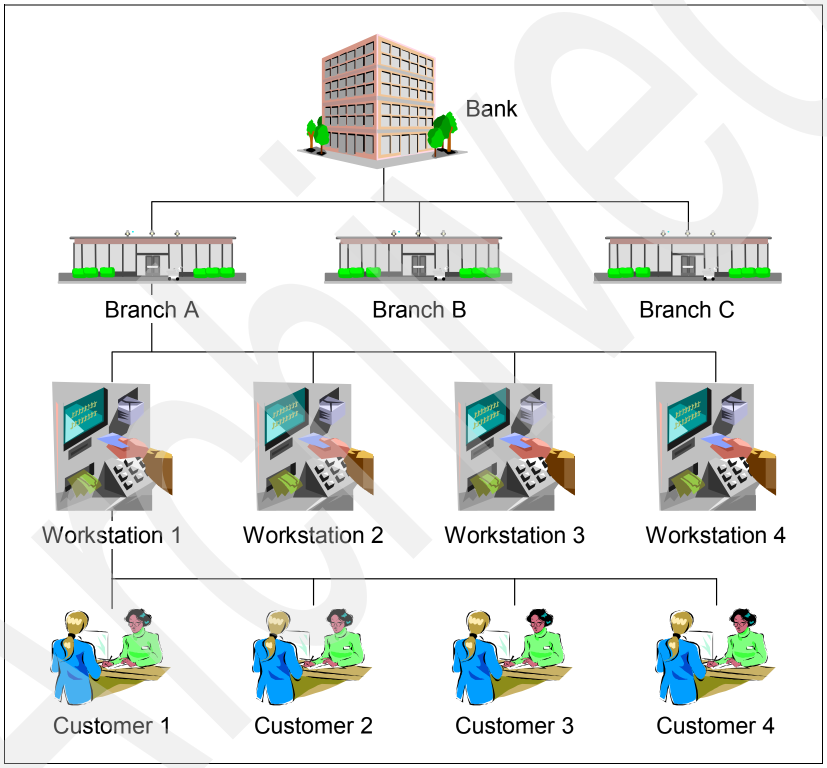


Figure 8-2 Hierarchical nature of a bank

In Branch Transformation Toolkit, the term *Context* is used to represent each element in the hierarchy. Contexts are “objects that provide the ability to structure resources according to the functional and business organization.”

Editor Comment: From where is this quote excerpted? We need to cite the source here. DS

In general, contexts are chained hierarchically. With this concept, a withdrawal operation might involve a Branch context, a Workstation context, a CustomerSession context, and a Withdrawal context. This relationship can be organized into a context chain as shown in Figure 8-3 on page 231.

With this hierarchical data structure, you can generate messages with a specific format with one or two method calls. This simplifies the effort of message formatting and generation for different purposes within an application.

From Branch Transformation Toolkit v5.1, use the Common Hierarchical Area (CHA) to manage the context hierarchy. The goal of CHA is to make context hierarchy a distributed runtime repository. It provides context functionality in a distributed server environment and enables that the information can be shared among Branch Transformation Toolkit applications. See Figure 8-3.

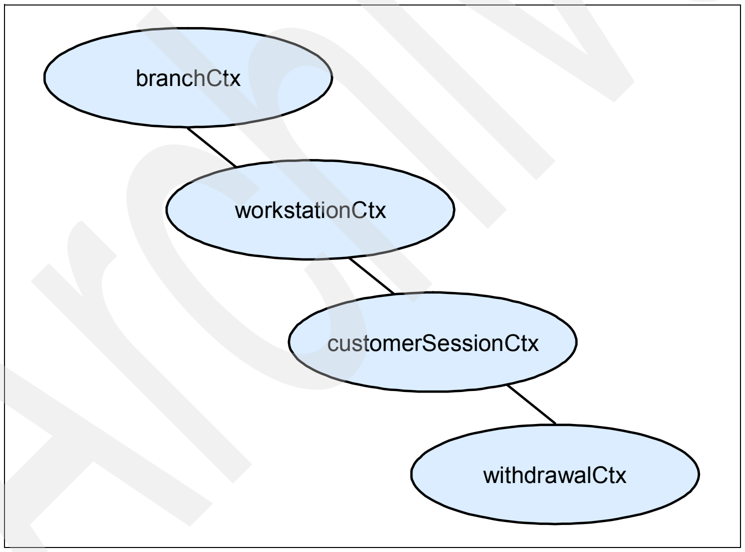


Figure 8-3 Context chain for a withdrawal operation

There are two context types in CHA, local context and remote context:

- Local context
  - This context has temporary runtime storage.
  - It is dynamic in nature.

  - It is an anonymous or predefined context.
  - It's life expectancy is short.
  - It cannot be shared by business processes and services.
- Remote context
  - This context has a server side and a client side.
  - The client side acts as a facade for the server side.
  - The server side consists of an entity EJB and a session EJB.
  - Both client and server sides connect to each other using J2EE EJB technology.
  - It is shared by business processes and services.

According to the arrangement of local and remote context, a typical context tree is shown in Figure 8-4 on page 233. You should decide which context will be remote and which context will be local while developing a Branch Transformation Toolkit application. The rule of the thumb is to put the contexts that are intended to be shared in the CHA server side, and the other temporary contexts in the CHA client side.

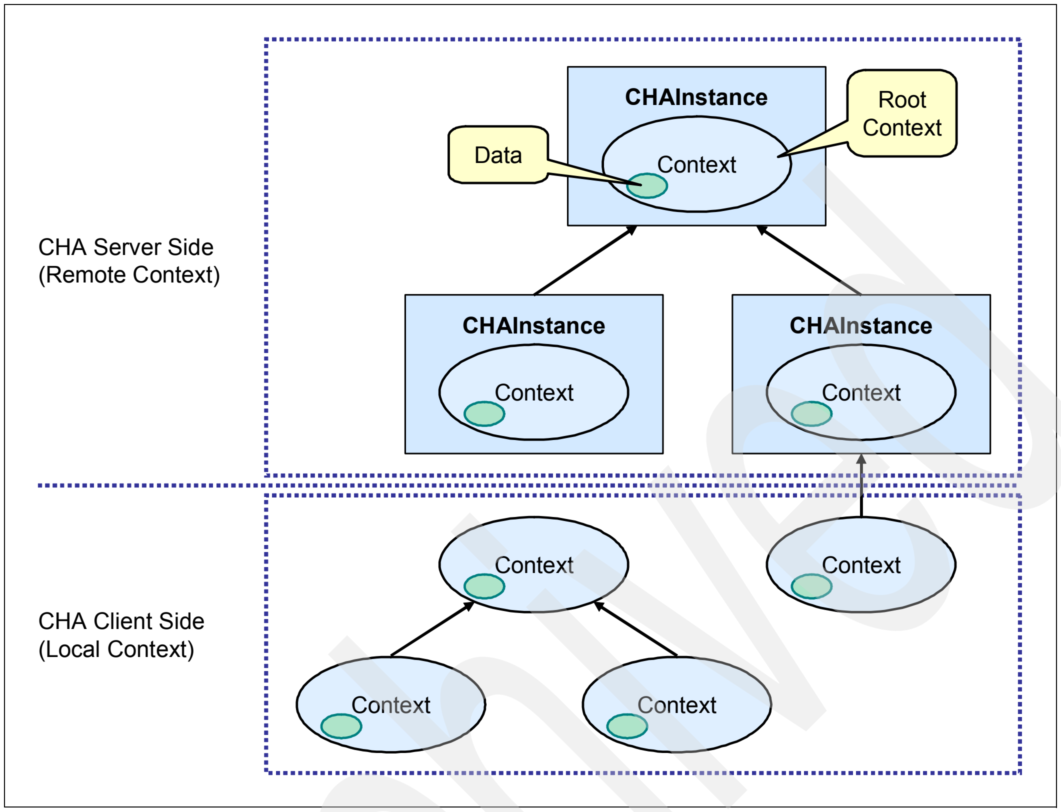


Figure 8-4   Sample context tree in a Branch Transformation Toolkit V5.1 application

Performance improvement

Branch Transformation Toolkit v5.1 provides two key technologies to speed up the access of remote context in a distributed environment.

- Read cache
  - Caches read-only data stored in remote context.
  - Reduces the amount of communication between the client side and the server side.
  - Has the same lifetime as its corresponding CHA context instance.
- Batch update
  - Submits many small sequential updates as a single request.
  - Reduces the amount of communication between the client side and the server side.

8.1.3 Branch Transformation Toolkit

IBM Branch Transformation Toolkit for WebSphere Studio is an application framework and a set of specialized eclipse-based tools that accelerate the build phase for transactional front office applications. It is designed to rapidly deliver multi-channel transactional solutions, as shown in Figure 8-5. New Branch Transformation Toolkit v5.1 framework or tools are based on the following design principles:

- ► Leverages core IBM application development platform.
- ► Enhances tooling capabilities.
- ► Provides a framework for developing standards-based applications.
- ► Stays in sync with current and evolving standards, open source, and platform evolution.
- ► Provides a path forward for WebSphere customers with limited J2EE development experience, who need to quickly develop transactional business applications.

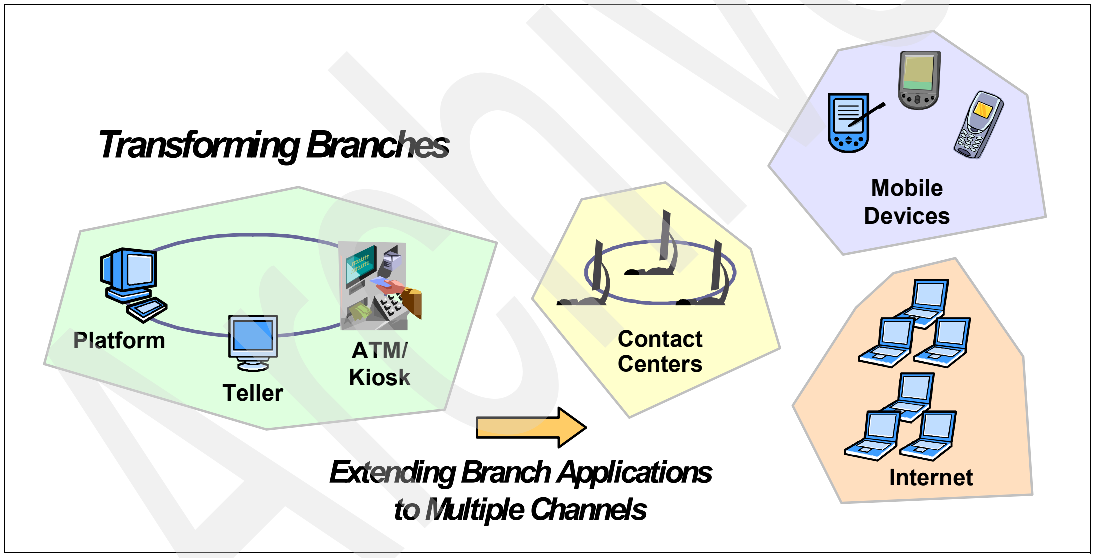


Figure 8-5 Branch Transformation Toolkit rapidly delivers multi-channel transactional solutions

The key features and benefits of IBM Branch Transformation Toolkit for WebSphere Studio include:

- ► Framework for multi-channel infrastructure supports the creation of integrated retail delivery applications, a key factor in reducing the total cost of system ownership.

- Time-saving features and integration build on the WebSphere Studio environment to enable developers maximize their productivity and improve the overall time to value.
- Tooling enhancements such as the new graphical workbench integrate plug-ins for a broad view of the development environment to simplify the creation of multi-tiered business applications.
- A robust banking solution gives retail banks desiring the control and flexibility associated with building a branch teller application.
- A range of sample applications contains sample applications addressing both base and advanced functionalities, such as implementation of Web services.
- Relevant to a variety of industries

  Enables retail initiatives, not only in the banking industry, but any industry with a desire to modernize enterprise applications.

8.2 Leveraging the WebSphere Studio features

Branch Transformation Toolkit v5.1 includes a graphical workbench and visual development tools as add-ons to the WebSphere Studio Application Developer or the WebSphere Studio Application Developer Integration Edition. This provides users a set of integrated development tools for all e-business development roles, including Web developers, Java developers, business analysts, architects, and enterprise programmers.

The toolkit, based on standards and leveraging the WebSphere Application Server and the WebSphere Business Integration Server Foundation, has the following features:

- Application development tooling
- Scalability and availability features
- Administration, deployment, and monitoring tools

Depending on the configuration you use, the toolkit's framework and tools support two different WebSphere product lines, that is, WebSphere Application Server/WebSphere Studio Application Developer and WebSphere Business Integration Server Foundation/WebSphere Studio Application Developer Integration Edition. See Figure 8-6 on page 236.

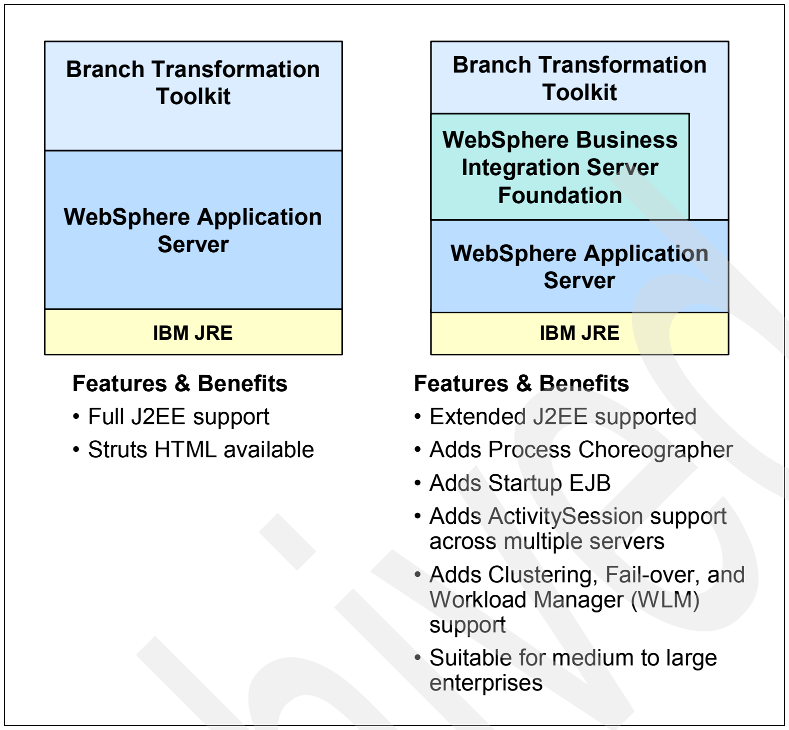

Branch Transformation Toolkit
WebSphere Application Server
IBM JRE
Features & Benefits
• Full J2EE support
• Struts HTML available
Branch Transformation Toolkit
WebSphere Business Integration Server Foundation
WebSphere Application Server
IBM JRE
Features & Benefits
• Extended J2EE supported
• Adds Process Choreographer
• Adds Startup EJB
• Adds ActivitySession support across multiple servers
• Adds Clustering, Fail-over, and Workload Manager (WLM) support
• Suitable for medium to large enterprises

Figure 8-6 Two configurations to use Branch Transformation Toolkit V5.1

As shown in Figure 8-6, the left panel shows the configuration of Branch Transformation Toolkit v5.1/ WebSphere Application Server/ WebSphere Studio Application Developer combination. Using this configuration, you can have full J2EE features with extended Apache Struts support. In the right panel, it combines Branch Transformation Toolkit v5.1/ WebSphere Business Integration Server Foundation / WebSphere Studio Application Developer Integration Edition. With this configuration, you can have full toolkit capabilities that are critical in enterprise environment, including Process Choreographer, Clustering, Fail-over, Workload Management, and so on.

Because Branch Transformation Toolkit v5.1 has a close relationship with WebSphere Studio, a brief introduction to these two products, that is, WebSphere Studio Application Developer and WebSphere Studio Application Developer Integration Edition, is provided.

8.2.1 Development using WebSphere Studio Application Developer

The award-winning IBM WebSphere Studio Application Developer is a comprehensive, integrated development environment for visually designing, constructing, testing, and deploying Web services, portals, and Java 2 Platform, Enterprise Edition(J2EE) applications. WebSphere Studio Application Developer accelerates J2EE development with a complete set of high productivity tools, templates, and wizards.

Built on Eclipse, an open, industry-supported platform for development tools, WebSphere Studio Application Developer enables you to adapt and extend your development environment with best-of-breed plug-in tools from IBM, IBM Business Partners, and the Eclipse community to match your needs and to maximize developer productivity.

WebSphere Studio Application Developer features include:

- Support for J2EE 1.3, including EJB 2.0, Servlet 2.3, and JSP 1.2 levels.
- Concurrent support for WebSphere Application Server V4 (J2EE 1.2) and WebSphere Application Server V5 (J2EE 1.3).
- Full EJB 2.0 support, including EJB Query Language (EJBQL), multiple-mapping support for Container Managed Persistence (CMP) 2.0, and message-driven beans.
- A set of visual portlet development tools and a WebSphere Portal unit test environment. Support for IBM Portlet API and JSR 168, the industry standard specification for portlet aggregation, personalization, presentation, and security.
- Specialized support for Struts, which is a set of Java classes, and JSP tag libraries that provide a conceptual framework for developing Web applications.
- JavaServer™ Faces (JSF) support for drag-and-drop Web application development.
- Lightweight runtime environment supports unit test of applications on local or remote servers, including WebSphere Application Server (V4, V5 and V5.1), WebSphere Application Server - Express (V5 and V5.1), and Apache Tomcat. Includes WebSphere Portal V5.0, WebSphere Application Server V4, V5, and V5.1 for unit test on local server.
- New Visual Editor for Java (Java-based client for building GUI components with Swing or AWT).
- Support for both Rational ClearCase® LT and full Rational ClearCase.

The following sections describe the various application development tools that come with this configuration of WebSphere Studio.

Java development tools

The Java development tools included with WebSphere Studio support the development of any Java application. They add Java perspectives to the workbench as well as a number of views, editors, wizards, builders, and code merging and refactoring tools. The Java development tools offer the following capabilities:

- JDK 1.4.1 support.
- Automatic incremental compilation.
- One debugger for both local and remote debugging.
- Ability to run code with errors in methods.
- Crash protection and auto-recovery.
- Error reporting and correction.
- Java text editor with full syntax highlighting and complete content assist.
- Refactoring tools for reorganizing Java applications.
- Intelligent search, compare, and merge tools for Java source files.
- Scrapbook for evaluating code snippets.
- Pluggable runtime support for JRE™ switching and targeting multiple runtime environments from IBM and other vendors.

Web development tools

The Web development environment in WebSphere Studio provides the tools necessary to develop Web applications as defined in the Java Servlet 2.3 Specification and the JavaServer Pages 1.2 Specification. Web applications include static Web pages, JavaServer Pages (JSPs), Java Servlets, deployment descriptors (web.xml files), and other Web resources.

This environment brings all aspects of Web application development into a common interface. Everyone in your Web site team, including content authors, graphic artists, programmers, and Webmasters, can work on the same projects and access the files they need. Within the integrated Web development environment, it is easy to collaborate on creating, assembling, publishing, deploying, and maintaining dynamic and interactive Web applications.

The Web development environment includes the following high-level capabilities:

- Web project creation, using either the J2EE-defined hierarchy or a static version that reduces project overhead when dynamic elements are not required.
- Creating and editing a Web deployment descriptor (web.xml) file.
- Creating, validating, editing, and debugging JSP and HTML files.
- Editing and validating JavaScript™.
- Supporting custom JSP tags (taglib) based on the JSP 1.2 specification.

- Cataloging and organizing reusable programming objects such as HTML, JavaScript, and JSP code, along with files and tag libraries, with the help of an extensible view called Library view.
- Editing images and animation.
- Supporting Cascading Style Sheet (CSS) editing.
- Importing HTTP/FTP.
- Exporting FTP (simple resource copying) to a server.
- Web archiving (WAR) file import, export, and validation.
- Link viewing, parsing, validation, and management, including converting links, flagging broken links, and fixing links as resources are moved or renamed.
- Creating servlets with a wizard and adding servlet mappings to the deployment descriptor (web.xml) file.
- Generating Web applications using wizards that create Web resources from database (SQL) queries and beans.
- Integrating with the WebSphere test environment.
- Publishing support for multiple Web server types.

Web services development tools

WebSphere Studio provides wizards and other tools to enable rapid development of Web services. Web services are modular, standards-based e-business applications that businesses can dynamically mix and match to perform complex transactions with minimal programming. Web services allow buyers and sellers all over the world to discover each other, connect dynamically, and execute transactions in real time with minimal human interaction.

Some examples of Web services are theatre review articles, weather reports, credit checks, stock quotations, travel advisories, airline travel reservation processes, and so on. Each of these self-contained business services is an application that can easily integrate with other services from the same or different companies, to create a complete business process. This interoperability allows businesses to publish dynamically, discover, and bind a range of Web services through the Internet.

The Web services development tools provided in WebSphere Studio are based on open, cross-platform standards:

- Universal Description Discovery and Integration (UDDI)

  UDDI enables businesses to describe themselves, publish technical specifications on how they want to conduct e-business with other companies, and search for other businesses that provide the goods and services they need, all through online UDDI registries.

- Simple Object Access Protocol (SOAP)

  SOAP is a standard for reliably transporting electronic business messages from one business application to another over the Internet.
- Web Services Description Language (WSDL)

  WSDL describes programs accessible through the Internet or other networks, and the message formats and protocols used to communicate with them.

Enterprise JavaBeans (EJB) development tools

The EJB development environment features full EJB 1.1 and 2.0 support, an EJB test client, a unit test environment for J2EE, and deployment support for Web archive (WAR) files and enterprise archive (EAR) files. Entity beans can be mapped to databases, and EJB components generated to tie into transaction processing systems. XML provides an extended format for deployment descriptors within EJB.

The following EJB development tools are included:

- Tools for import and export, creation and code generation, and editing, as well as support for standard deployment descriptors and extensions and bindings specific to WebSphere Application Server.
- EJB-to-RDB mapping tools that provide the model runtime environment and interface for editing the mapping between EJB beans and relational database tables with top-down and bottom-up capability. The mappers support associations, inheritance, and converters and composers as helpers on column maps.
- A query engine that supports deployed code by generating SQL strings into persistent classes.
- Tools that provide the ability to create, edit, and validate EAR files.
- Editors for deployment descriptors.
- Graphical RDB schema viewing and editing tools.
- The deployment tools for enterprise beans provides a command-line environment for you to run overnight build processes and automatically generate your deployment code in batch mode.

XML and XSL tools

The comprehensive XML toolset includes components for building DTDs, XML schemas, XML, and XSL files. It also supports integration of relational data and XML.

The XML Editor simplifies development tasks in the following ways:

- Provides a wizard that makes it quick and easy to create XML files from scratch, or from existing DTD or XML schema files.
- Provides a Design view and a Source view. The Design view represents the XML file simultaneously as a table and a tree, which helps make navigation and editing easier. You can use the Source view to view and work with a file's source code directly.
- Enables you to add a DTD declaration or XML schema information to an XML editor.

Relational database tools

WebSphere Studio provides you with relational database tools that you need to work with relational databases in your application development. The relational database tools include views, wizards, editors, and other features that make it easy for you to develop and test the database elements of your application. Unless explicitly stated, the features in the relational database tools support all database vendors.

You can manage the database definitions and connections that you need for your application development, connect to databases, import database definitions, and define new databases, schemas, tables, and views.

The SQL query builder provides a visual interface for creating and executing SQL statements. You can create a simple statement or add complex expressions and grouping. When you are satisfied with your statement, use the SQL to XML wizard to generate an XML document as well as XSL, DTD, XSD, and HTML files, plus other related artifacts, and then use the files to implement your query in other applications, for example, a servlet or JSP.

Struts application development tools

Struts is a set of Java classes and JSP tag libraries that provide a conceptual framework for developing Web applications. The Struts technology is open source and was developed as part of the Apache Software Foundation's Jakarta project.

For an overview of Struts, including information about how WebSphere Studio supports the technology, refer to the following Web site:

http://struts.apache.org/index.html

Component test tools

The component test tools provide a framework for defining and executing test cases. The basic framework supports three sorts of test cases, that is, manual, Java, and HTTP. You can also create report generators to work with the data returned by an executed test case.

Use the component test tools to perform the following tasks:

- Define manual test cases that automate a tester's to-do list.
- Define HTTP test cases that automate requests against a Web site.
- Define Java test cases that implement the JUnit framework to automate Java method calls.
- Run test cases locally or remotely, using the Agent Controller.
- Track execution results as the test case executes.
- Generate reports on test case information.
- Define new report generators.

The tools can be used by developers to test their code or by testers to coordinate project-wide testing efforts.

Testing and publishing tools

The testing and publishing tools, referred to in this section as *Server tools*, provide a unit test environment, where you can test JSP files, servlets, and HTML files. Server tools also provide the capability to configure other local or remote servers for integrated testing and debugging of Web and EJB applications.

To configure remote servers, you should have RAC, which is included with the product, installed on the remote system. Server tools support the following projects:

- Web projects that might contain JSP files, HTML files, servlets, and beans.
- EJB projects that contain enterprise beans.
- Enterprise Application projects that may contain Java archive (JAR) files or Web archive (WAR) files, or both, and pointers to other Web or EJB projects.

Profiling tools

WebSphere Studio provides tools that enable you to test your application's performance early in the development cycle. This allows enough time to make architectural changes and the resulting implementation changes. This reduces risk early in the cycle, and avoids problems in the final performance tests. The Profiling tools bring together a range of techniques that let you explore many aspects of your program.

Debuggers

All the products based on Eclipse include a debugger that enables you to detect and diagnose errors in your programs that are running either locally or remotely. The debugger lets you control the execution of your program by setting breakpoints, suspending execution, stepping through your code, and examining the contents of variables.

You can debug live server-side code as well as programs running locally on your workstation. The debugger includes a Debug view that shows threads and stack frames, a Processes view that shows all currently running and recently terminated processes, and a Console view that lets you interact with running processes. There are also views that display breakpoints and let you inspect variables.

For more information, refer to the IBM Redbook *WebSphere Studio Application Developer Version 5 Programming Guide*, SG24-6957.

8.2.2 Using Integration Edition

WebSphere Studio Application Developer Integration Edition v5.1, which is optimized for developing applications that deploy to IBM WebSphere Business Integration Server Foundation v5.1, delivers a next generation integration platform optimized for building and deploying composite applications that extend and integrate existing IT assets.

The advantages of WebSphere Studio Application Developer Integration Edition v5.1 include:

- Extend and integrate existing IT assets using a next generation integration development environment optimized for building composite applications that deploy to IBM WebSphere Business Integration Server Foundation.
- Maximize the return on your IT investments by creating easily reusable services out of your Web services, Java assets, back-end systems, packaged applications, people, and processes.
- Improve your IT responsiveness by leveraging a service-oriented architecture to build modular applications that are designed to adapt quickly to change.
- Expand the reach of your existing systems using a broad portfolio of rich application and technology adapters.
- Maximize your developer productivity by quickly constructing new process-based applications using drag-and-drop development tools to visually coordinate the interactions between your software assets.
- Anticipate change by using business rules to embed adaptable business logic into your applications and business processes.

- Minimize your development, deployment, and administration costs by building on the industry-leading, industry-tested, industry-supported WebSphere platform.
- Protect your infrastructure investments and minimize training costs by developing applications using industry supported open standards

Following are the distinct features that WebSphere Studio Application Developer Integration Edition provides:

BPEL4WS support

Business Process Execution Language for Web Services (BPEL4WS) defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. Support for BPEL4WS includes:

- Application assembly, deployment, and runtime support for BPEL4WS-based business processes.
- Intuitive drag-and-drop tools to visually define the sequence and flow of BPEL4WS business processes.
- A visual business process debugger to step through and debug BPEL4WS business processes.
- Compensation support to provide transaction rollback such as support for loosely coupled business processes that cannot be undone automatically by the application server.
- Flexibility to develop processes using a top-down, bottom-up, or meet-in-the-middle approach.
- A standards-based XML Path Language (XPATH) / Extensible Stylesheet Transformation (XSLT) wizard to map data between nodes in a process
- Integrated fault handling to provide an easy and integrated means of performing in-flow exception handling.
- A visual condition builder allowing you to easily direct the execution of BPEL4WS processes.
- Support for including Java snippets and artifacts as part of a business process.

Human workflow support

Human workflow support expands the reach of BPEL4WS to include activities that require human interaction as steps in an automated business process. Business processes involving human interaction are interruptible and persistent, for example, a person may take a long time to complete a task, and resume when the person completes the task. Human workflow support includes:

- Staff activity nodes to represent a step in a business process that is performed manually.
- Ability to assign people, for example those who report to you directly, to specific instances of a process through staff queries that are resolved at runtime, using an existing enterprise directory.
- Graphical browser-based interface for querying, claiming, working with, completing, and transferring work items to another user.
- Work item management support for managing the creation, transfer, and deletion of work items.
- Dynamic setting of duration and calendar attributes for staff activities.
- Dynamic setting of staff assignment through custom attributes.

Back-end system connectivity

Back-end system connectivity supports building Web applications and BPEL4WS business processes that integrate with back-end systems, including the following:

- Integrated tool support for using J2EE Connector Architecture 1.0 (JCA) 1.0 resource adapters to access back-end systems.
- Enhanced tool integration for JCA adapters with tool plug-in extensions that are available from IBM and business partners.
- Easy to use tools for creating services out of JCA resource adapters and including those services as part of a BPEL4WS business process.
- Enhanced JCA 1.0 resource adapters included for CICS®, IBM Host On-Demand, and IBM IMS™, which is for development use only.
- Sophisticated wizards to manage the low-level data handling requirements for JCA resource adapters.
- Wizards to quickly and simply expose CICS or IMS programs as enterprise services, including the ability to import definitions from COBOL, C structures, CICS basic mapping support (BMS), and IMS Message Format Service (MFS) definitions.
- Support for the entire suite of WebSphere Business Integration Adapters.

Business rule beans

Business rule beans offer a powerful, real-time framework for defining, executing, and managing business rules that encapsulate business policies that vary based on changes in the business environment. For example, a simple business rule might be, “If a customer's shopping cart is greater than $X, then offer a Y% discount." Business rule support includes the following:

- Easy to use tools for defining, executing, and managing business rules.
- Cheat sheet for defining business rules.
- Update business rules at runtime using a straightforward user interface, without the need to bring the application or server down.
- Organize business rules into logical categories using the business rules beans framework.
- Define a start and end date until when a rule is effective.

Programming model extensions

This accelerates large-scale application development by taking advantage of the latest innovations that build on today's J2EE standards. Programming model extensions include:

- Asynchronous beans

  This enables J2EE applications to decompose operations into parallel tasks in order to speed performance.
- Startup beans

  This enables J2EE applications to execute business logic automatically, whenever an application starts or stops normally.
- Last participant support

  This provides automated coordination for transactions, including two-phase commit resources and a single, one-phase commit resource.
- Internationalization service

  This allows customers to build applications that can automatically adjust to handle global audiences.
- Work areas

  This provides the ability to efficiently share information across a distributed application.
- Scheduler service

  This enables tasks to be executed at a requested time. When used in conjunction with asynchronous beans, it enables batch processing applications within J2EE.
- Activity session services

  This provides the ability to extend the scope of and group multiple local transactions.

- Dynamic query service

  This provides the ability to pass in and execute SQL query statements at runtime.
- Gateway filters

  This allows customers to write filters for the Web Services Gateway such as filters that select a target service and port, capture Web service invocation information, and handle exceptions.
- Object pools

  This enables an application to avoid creating new Java objects repeatedly.
- Container-managed messaging

  This offers automated support for outbound as well as inbound messaging.
- Distributed map

  This offers an interface to enable J2EE applications and system components to cache and share Java objects by storing a reference to the object in the cache in order to improve performance.
- Container-Managed Persistence over anything

  This extends the existing J2EE Container-Managed Persistence (CMP) framework to support any back-end system or service that supports create, retrieve, update, and delete (CRUD) methods.

Quality of service

- Application profiling

  This allows customers to carefully optimize the performance of their EJB CMP 2.0 applications without impacting application source code by delivering a mechanism for instructing the same component to interact with the runtime infrastructure, such as a database, differently depending on the application that calls it.
- Back-up cluster support

  This enables customers to automatically configure their system to set up a back-up cluster of servers if the primary cluster fails, without having to write any code.

  For more information, refer to the IBM Redbook *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200.

8.3 Developing an application using Branch Transformation Toolkit

Branch Transformation Toolkit v5.1 includes a graphical workbench and visual development tools as add-ons to the WebSphere Studio Application Developer or the WebSphere Studio Application Developer Integration Edition. It provides tools to create multi-channel application in bottom-up and top-down approaches,

8.3.1 Development paths

Branch Transformation Toolkit v5.1 is a versatile toolset that allows the development of a multi-channel application along a variety of paths. The development path used depends on whether the development begins from a new project, or from existing components. Two typical paths in application development are:

- Top-down development, which starts with a high-level architecture.
- Bottom-up development, which starts with existing components.

Top-down development

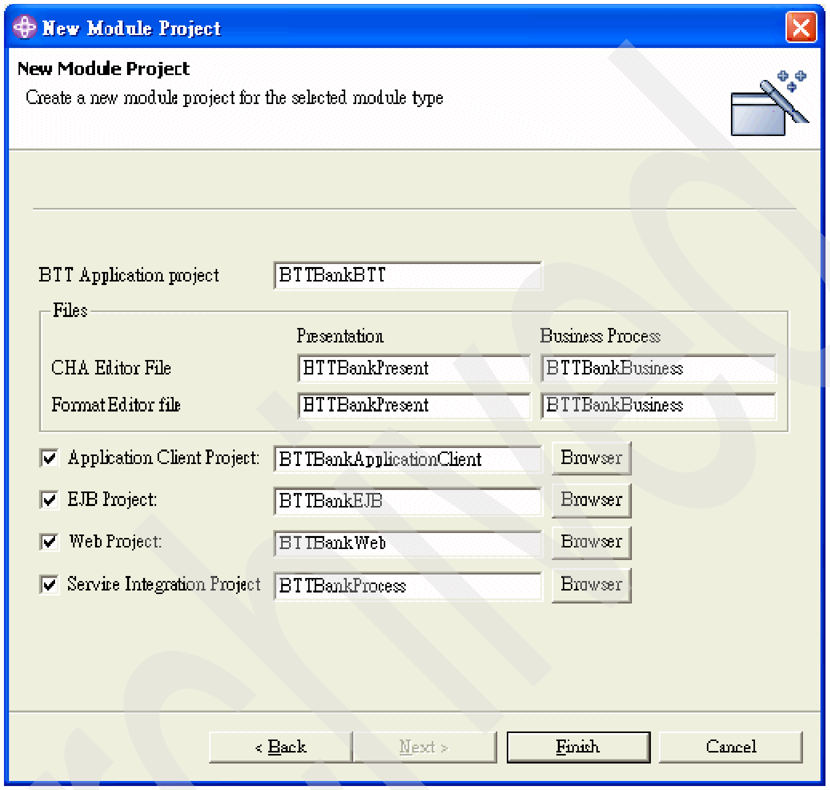
The first step in the development path is the topology of the target system defined by the technical architect. From there, you can use the toolkit's Graphical Builder to generate the appropriate artifacts, such as presentation model, business model, and so on. You can then divide the tasks of realizing artifacts assigning them to different team members. This shows the advantage of top-down development path, which is, *parallel development with an integrated architecture.*

After the artifacts are realized, it is time to generate the real working code. With Branch Transformation Toolkit tools, you can control every generation step and make adjustments to the generation options as well as to the results created by the Graphical Builder and other ancillary tools.

The top-down path is also typical for prototypes, rapid application development (RAD), and initial versions of an application. This development path guarantees the option for clean, object-oriented implementation of the business model, as well as performing state of the art J2EE implementations.

Following is a typical development path of a Branch Transformation Toolkit application using the top-down approach:

1. Create a Branch Transformation Toolkit project, as shown in Figure 8-7.



New Module Project

Create a new module project for the selected module type

BTT Application project: BTTBankBTT

Files

	Presentation	Business Process
CHA Editor File	BTTBankPresent	BTTBankBusiness
Format Editor file	BTTBankPresent	BTTBankBusiness

- [✓] Application Client Project: BTTBankApplicationClient
- [✓] EJB Project: BTTBankEJB
- [✓] Web Project: BTTBankWeb
- [✓] Service Integration Project: BTTBankProcess

Figure 8-7 The Branch Transformation Toolkit project wizard

2. Elaborate the presentation layer, as shown in Figure 8-8.

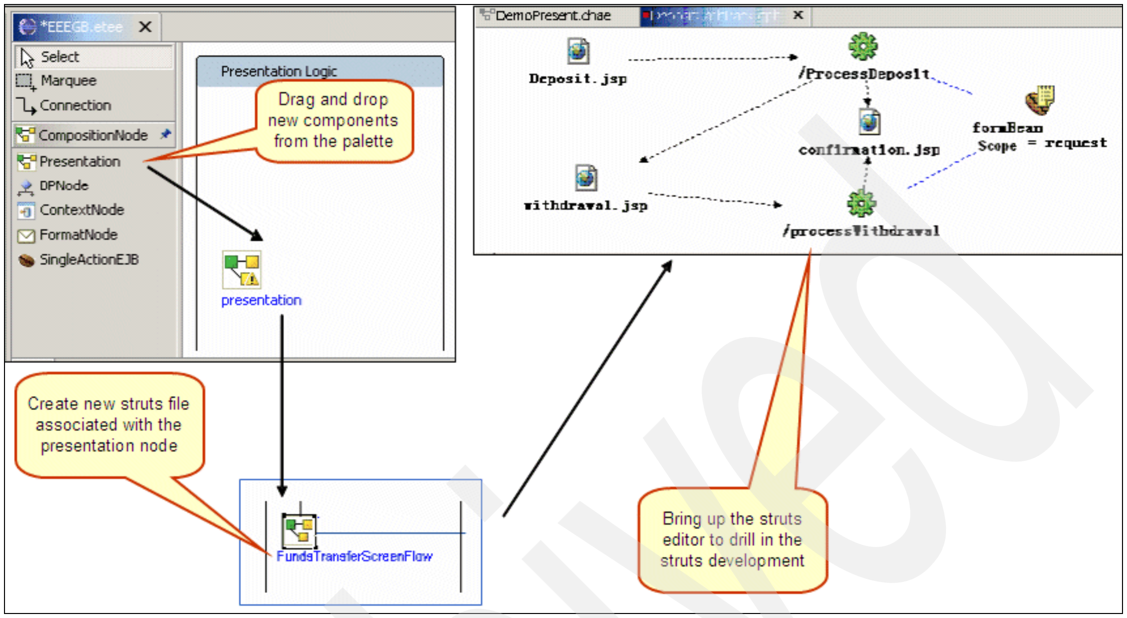
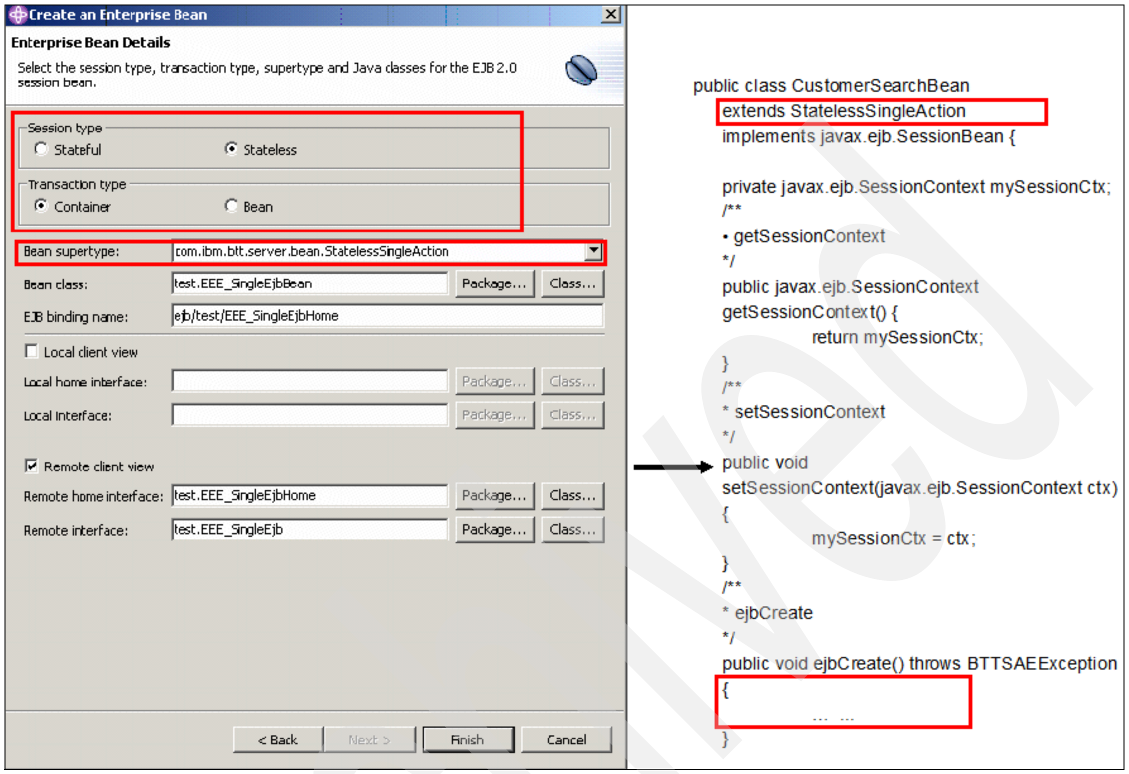


Figure 8-8 Create models for the presentation layer

3. Construct the business layer as shown in Figure 8-9.



Enterprise Bean Details

Select the session type, transaction type, supertype and Java classes for the EJB 2.0 session bean.

Session type: Stateful [✓ Stateless]

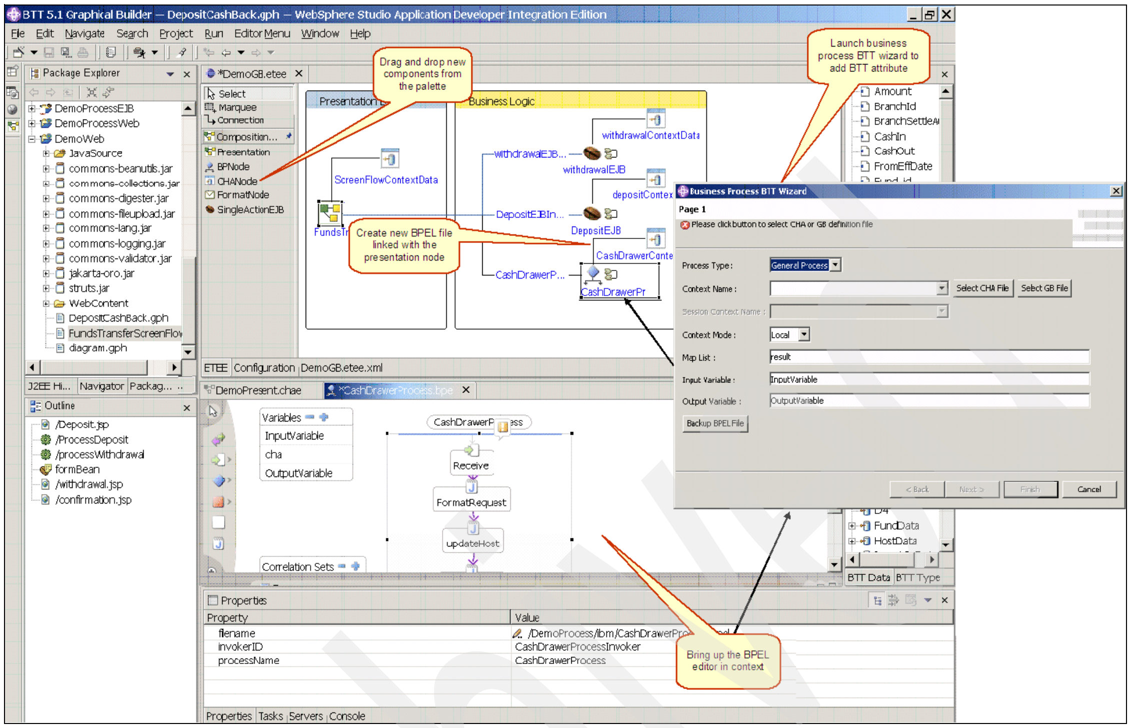
Transaction type: [✓ Container] Bean

Bean supertype:	com.ibm.btt.server.bean.StatelessSingleAction
Bean class:	test.EEE_SingleEjbBean
EJB binding name:	ejb/test/EEE_SingleEjbHome
☐ Local client view	
Local home interface:	
Local interface:	
☑ Remote client view	
Remote home interface:	test.EEE_SingleEjbHome
Remote interface:	test.EEE_SingleEjb

```
public class CustomerSearchBean
    extends StatelessSingleAction
    implements javax.ejb.SessionBean {

    private javax.ejb.SessionContext mySessionCtx;
    /**
    • getSessionContext
    */
    public javax.ejb.SessionContext
    getSessionContext() {
                return mySessionCtx;
    }
    /**
    * setSessionContext
    */
    public void
    setSessionContext(javax.ejb.SessionContext ctx)
    {
                mySessionCtx = ctx;
    }
    /**
    * ejbCreate
    */
    public void ejbCreate() throws BTTSAEException
    {
                ... ...
    }
```

Figure 8-9   Create Single Action EJB as the business operation

4. Create any required business processes. See Figure 8-10 on page 252



Business Process BTT Wizard

Page 1

Please click button to select CHA or GB definition file

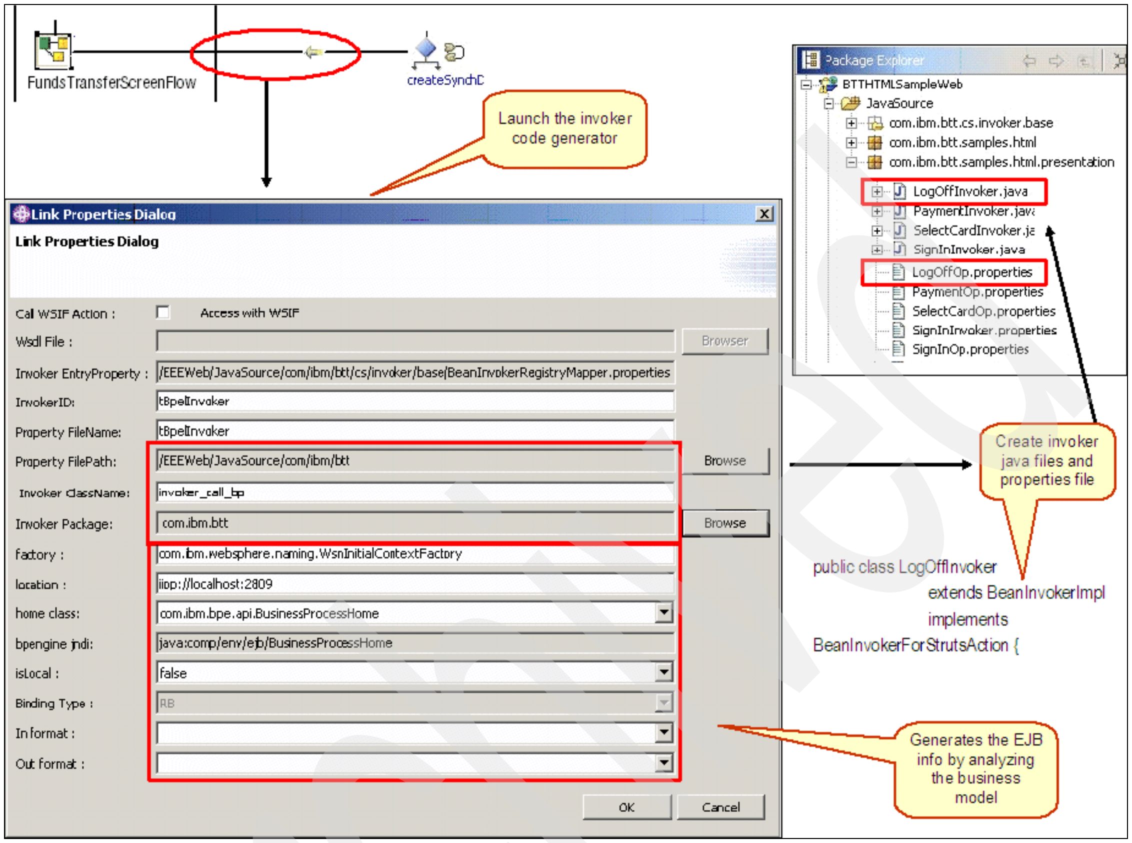
Process Type :	General Process
Context Name :	
Session Context Name :	
Context Mode :	Local
Map List :	result
Input Variable :	InputVariable
Output Variable :	OutputVariable

Select CHA File | Select GB File | Backup BPEL File | < Back | Next > | Finish | Cancel

Property	Value
filename	/DemoProcess/ibm/CashDrawerPro...
invokerID	CashDrawerProcessInvoker
processName	CashDrawerProcess

Figure 8-10   Use BPEL as the business operation

5. Associate presentation and business logic as shown in Figure 8-11.



public class LogOffInvoker
extends BeanInvokerImpl
implements
BeanInvokerForStrutsAction {

Link Properties Dialog

Call WSIF Action :	☐ Access with WSIF
Wsdl File :	
Invoker EntryProperty :	/EEEWeb/JavaSource/com/ibm/btt/cs/invoker/base/BeanInvokerRegistryMapper.properties
InvokerID:	tBpelInvoker
Property FileName:	tBpelInvoker
Property FilePath:	/EEEWeb/JavaSource/com/ibm/btt
Invoker ClassName:	invoker_call_bp
Invoker Package:	com.ibm.btt
factory :	com.ibm.websphere.naming.WsnInitialContextFactory
location :	iiop://localhost:2809
home class:	com.ibm.bpe.api.BusinessProcessHome
bpengine jndi:	java:comp/env/ejb/BusinessProcessHome
isLocal :	false
Binding Type :	RB
In format :	
Out format :	

Figure 8-11 Link the presentation layer and business layer by invokers

Bottom-up development

The bottom-up approach is typical for a new application or application extension, based on the existing components. Depending on the action you are carrying out, Graphical Builder is the key tool that keeps the architecture of an application clear and extensible.

The first step in this path is capturing the existing origianl code as a presentation or business model in the Graphical Builder. From there, you can generate the components, that is, the hierarchical context, business processes, and UI flows. You can control every capture or generation step and make adjustments to the options as well as to the results produced by the Graphical Builder. The generated results are a good base to start with the changes and adjustments. See Figure 8-12.

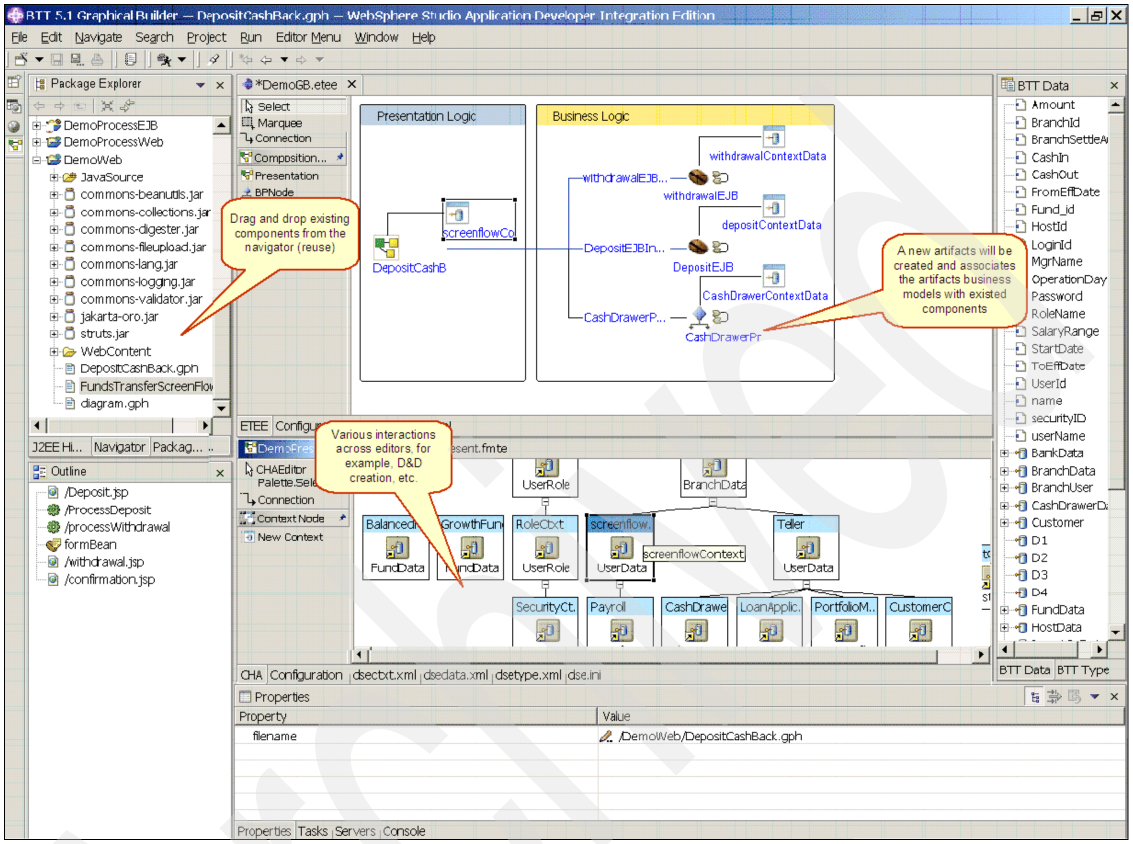

BTT 5.1 Graphical Builder — DepositCashBack.gph — WebSphere Studio Application Developer Integration Edition
File Edit Navigate Search Project Run Editor Menu Window Help
Package Explorer
*DemoGB.etee
Presentation Logic
Business Logic
BTT Data
Drag and drop existing components from the navigator (reuse)
A new artifacts will be created and associates the artifacts business models with existed components
Various interactions across editors, for example, D&D creation, etc.
Properties
Property
Value
filename
/DemoWeb/DepositCashBack.gph
Outline
Properties Tasks Servers Console

Figure 8-12   Bottom-up development with existing components

For developing a new application using the bottom-up approach, use the CHA Editor, the Format Editor, the Business Process BTT Wizards, and the Struts tools BTT Extensions to construct each component of an application. Although you can complete a Branch Transformation Toolkit application this way, it is better to use the Graphical Builder to weave all these components into a big picture. Figure 8-12 helps you spot potential performance bottlenecks and design flaws.

8.3.2 Preparing for sample application

In our sample, we developed a Branch Transformation Toolkit Web application based on the application scenario described in 8.1.1, “Application scenario” on page 226. Through the construction of the withdrawal operation, you can explore the CHA Editor, the Format Editor, the Business Process BTT Wizard, the Struts Tools BTT Extensions, and the Graphical Builder.

Creating a Branch Transformation Toolkit project

Create a Branch Transformation Toolkit project by performing the following tasks:

1. From the WebSphere Studio Application Developer menu, select **File** → **New** → **Other**.
2. In the dialog box, select **IBM Branch Transformation Toolkit** in the left navigation panel, and **BTT Project** in the right panel, as shown in Figure 8-13, and Click **Next.**

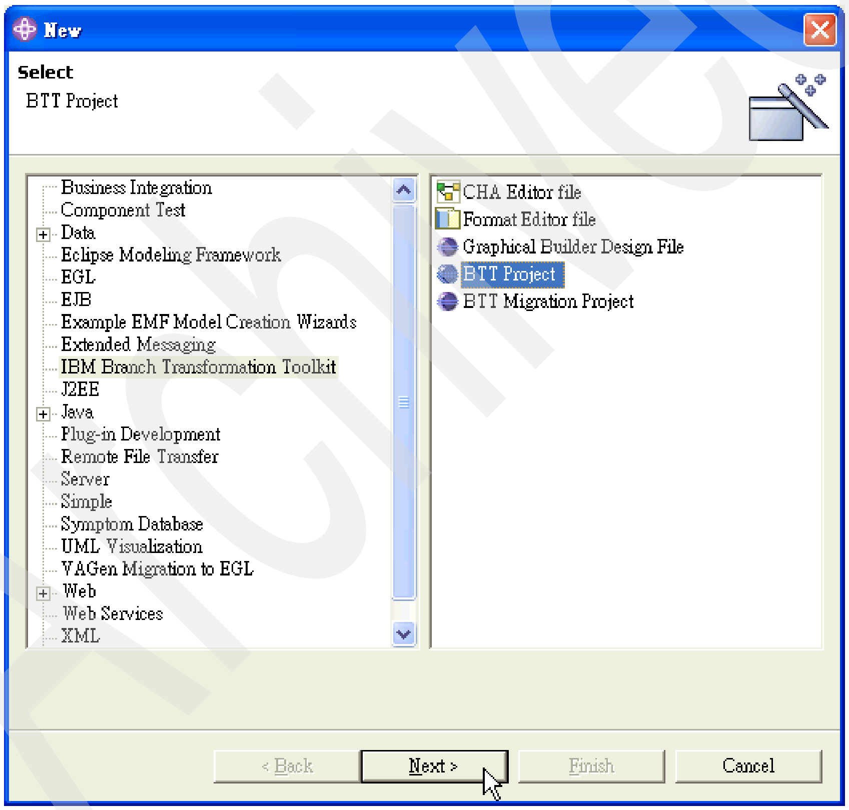
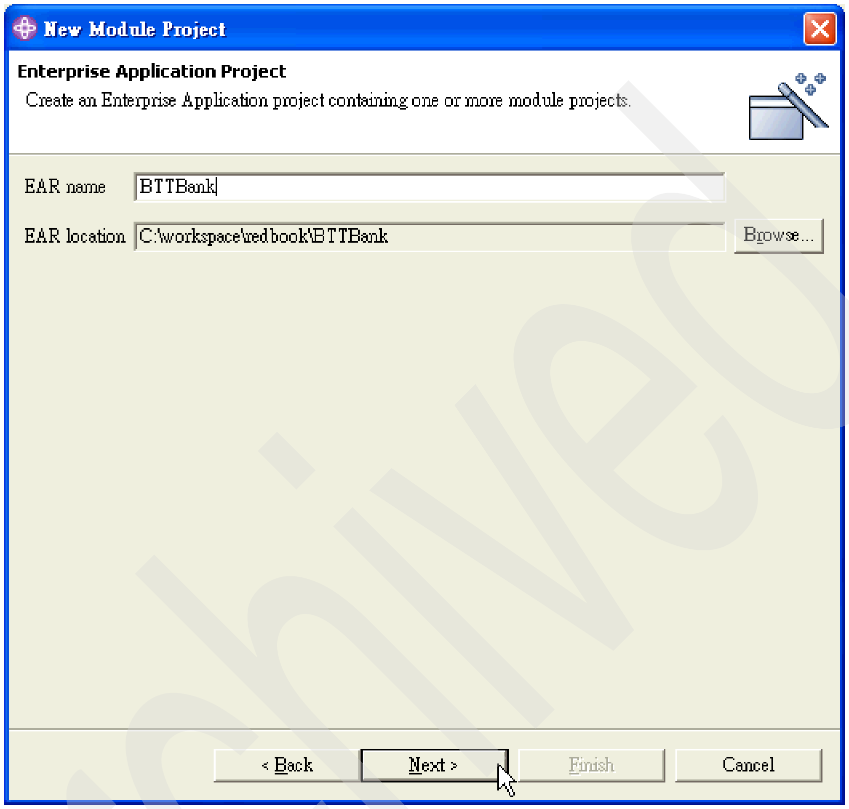


Figure 8-13 Creating a Branch Transformation Toolkit project

3. In the new Enterprise Application Project page, enter `BTTBank` as the EAR name as shown in Figure 8-14, and click **Next**.


New Module Project

Enterprise Application Project

Create an Enterprise Application project containing one or more module projects.

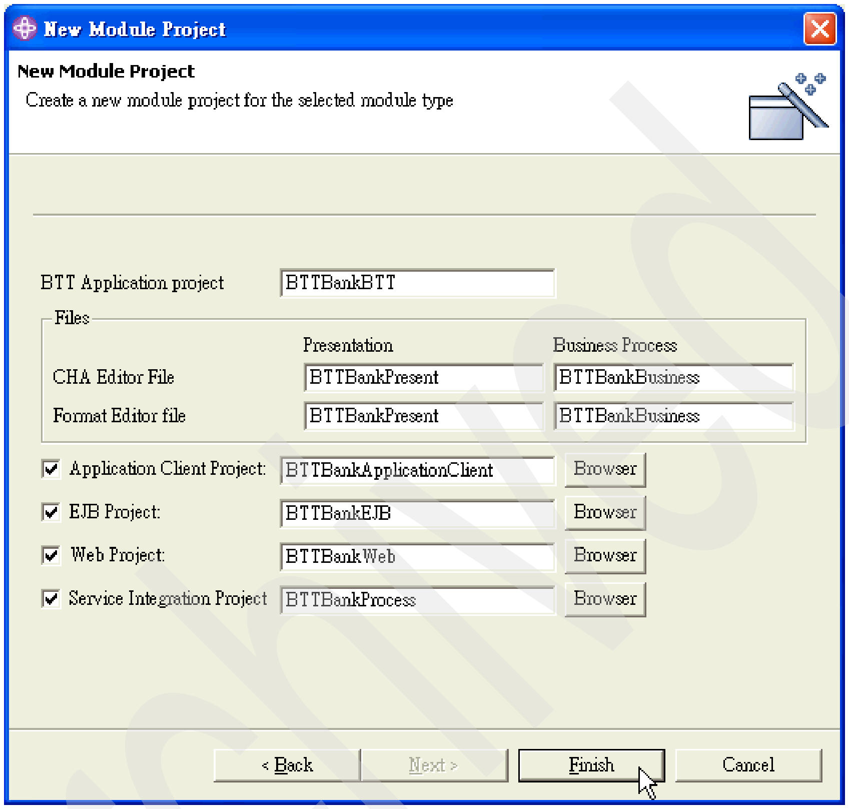
EAR name: BTTBank

EAR location: C:\workspace\redbook\BTTBank

Figure 8-14 Specify the EAR name of the Enterprise Application Project

4. In the next dialog box shown in Figure 8-15 on page 257, you are prompted to confirm the BTT Application project name, the CHA Editor, and the Format

Editor file names and optional supporting modules. Accept the defaults and click **Finish**.


New Module Project

New Module Project

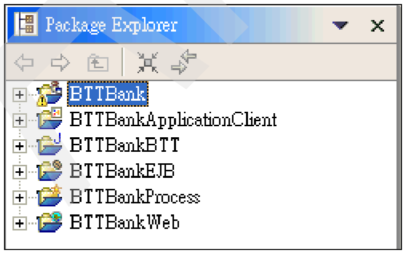
Create a new module project for the selected module type

BTT Application project	BTTBankBTT	
Files	Presentation	Business Process
CHA Editor File	BTTBankPresent	BTTBankBusiness
Format Editor file	BTTBankPresent	BTTBankBusiness

- [✓] Application Client Project: BTTBankApplicationClient
- [✓] EJB Project: BTTBankEJB
- [✓] Web Project: BTTBankWeb
- [✓] Service Integration Project BTTBankProcess

Figure 8-15 The New Module Project page of the toolkit project wizard

5. After a couple of moments, the selected projects are created, as shown in Figure 8-16.


Package Explorer

- BTTBank
- BTTBankApplicationClient
- BTTBankBTT
- BTTBankEJB
- BTTBankProcess
- BTTBankWeb

Figure 8-16 Skeleton projects created by toolkit project wizard

Setting up the sample database

This section provides instructions for deploying the BTTBank sample database. In our sample, we used DB2 v8.1 as the database management system.

To create the database, create the connection, and then create the tables for the BTTBank sample, as described:

1. Create a database, `BTTBank`, by entering the following command in the DB2 command window:

   ```
   DB2 CREATE DATABASE BTTBank
   ```

2. Create CHA tables.

   If the userID and password for the database BTTBank is db2admin/db2admin:

   a. Open the DB2 command window and change to the directory <BTT_install_dir>\dbtools\Windows\DB2\tableDefinition\cha\

   b. In the DB2 command window, run:

      ```
      DB2 CONNECT TO BTTBank USER db2admin USING db2admin
      ```

   c. Type the following command:

      ```
      db2 -tvf createCHATables.ddl
      ```

      You will see messages indicating that CHA tables have been created successfully.

Setting up the WebSphere testing environment

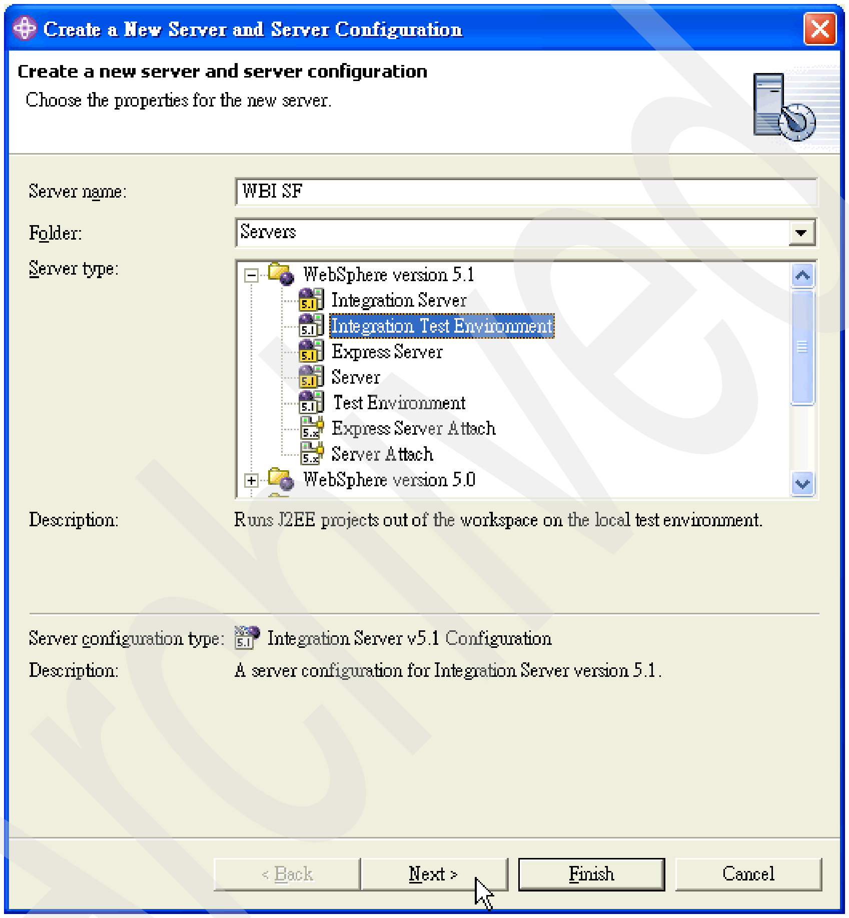
In this section, we provide instructions for setting up a WebSphere testing environment for our sample application in WebSphere Studio Application Developer Integration Edition v5.1.1.

Creating a server configuration

Follow these steps to create a server configuration.

1. Ensure that the WebSphere Studio Application Developer Integration Edition v5.1.1 is started.
2. Open the **Server** perspective.
3. In the Server Configuration view, right-click the empty area and select **New** → **Server and Server Configuration**.

4. Create a new server configuration with the following properties, as shown in Figure 8-17, and click **Finish**:
   – Server name: `WBI SF`
   – Folder: `Servers`
   – Server Type: `WebSphere version 5.1 Integration Test Environment`


Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: WBI SF

Folder: Servers

Server type:
- WebSphere version 5.1
  - Integration Server
  - Integration Test Environment
  - Express Server
  - Server
  - Test Environment
  - Express Server Attach
  - Server Attach
- WebSphere version 5.0

Description: Runs J2EE projects out of the workspace on the local test environment.

Server configuration type: Integration Server v5.1 Configuration

Description: A server configuration for Integration Server version 5.1.

Figure 8-17 Create a server configuration

Configuring the DataSource

To configure a new DataSource, perform the following tasks:

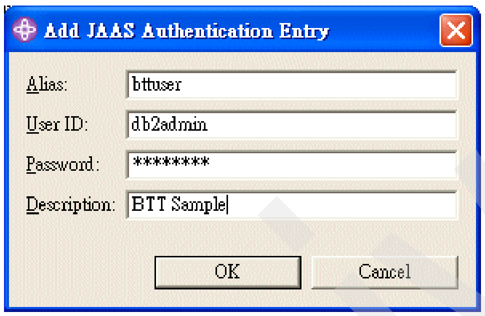
1. Open the server configuration.

   Double-click the server configuration **WBI SF** in the Server Configuration or Server view. You will see the file named WBI SF opened in the Editor panel.

2. Add a JAAS Authentication Entry.

   Select the **Security** tab. You will see the Cell Settings section in this page. Click **Add** to the right of the JAAS Authentication Entries. Enter the following values to create a JAAS Authentication Entry, as shown in Figure 8-18, and click **OK**:

   – Alias: `bttuser`
   – User ID: `db2admin`
   – Password: `db2admin`
   – Description: `BTT Sample`



Alias: bttuser
User ID: db2admin
Password: ********
Description: BTT Sample

Figure 8-18 Add a JAAS authentication entry

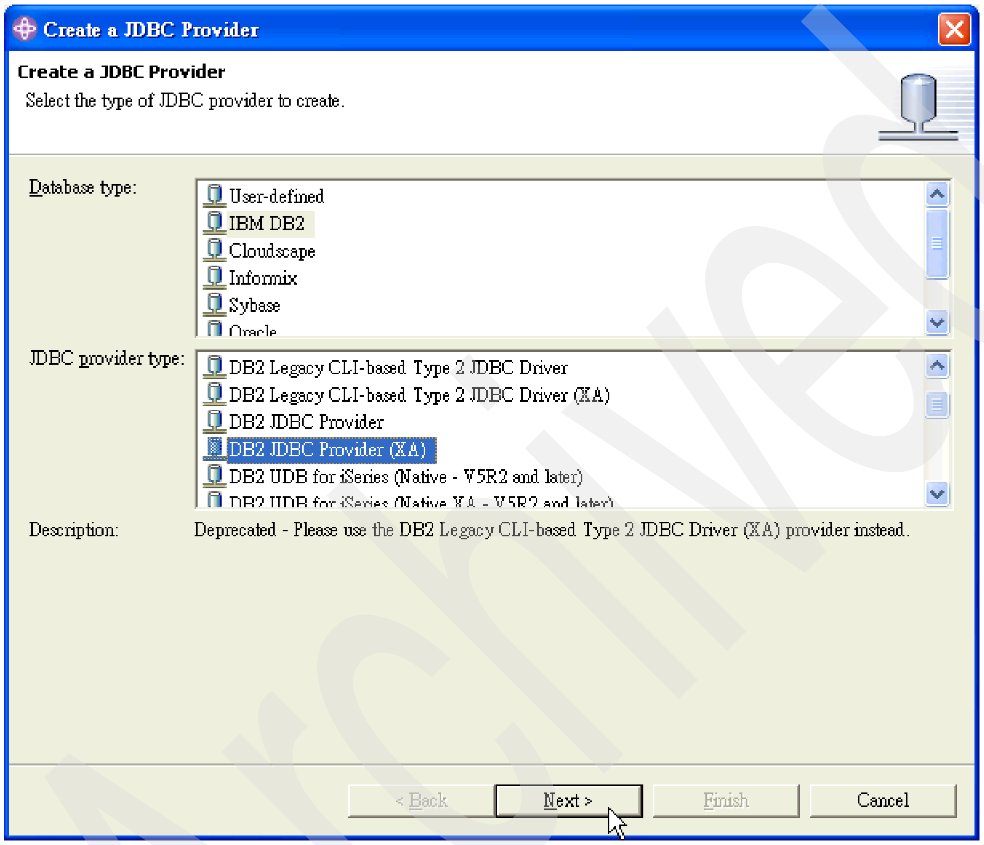
3. Add a JDBC provider.

   a. Select the **Data Source** tab. You will see the Server Settings section in this page.

   b. Click **Add** to the right of the JDBC provider list.

c. In the Create a JDBC Provider dialog box, select **IBM DB2** as the Database type, and **DB2 JDBC Provider (XA)** in the JDBC provider type list box, as shown in Figure 8-19.

d. Click **Next**.



Create a JDBC Provider

Select the type of JDBC provider to create.

Database type:

- User-defined
- IBM DB2
- Cloudscape
- Informix
- Sybase
- Oracle

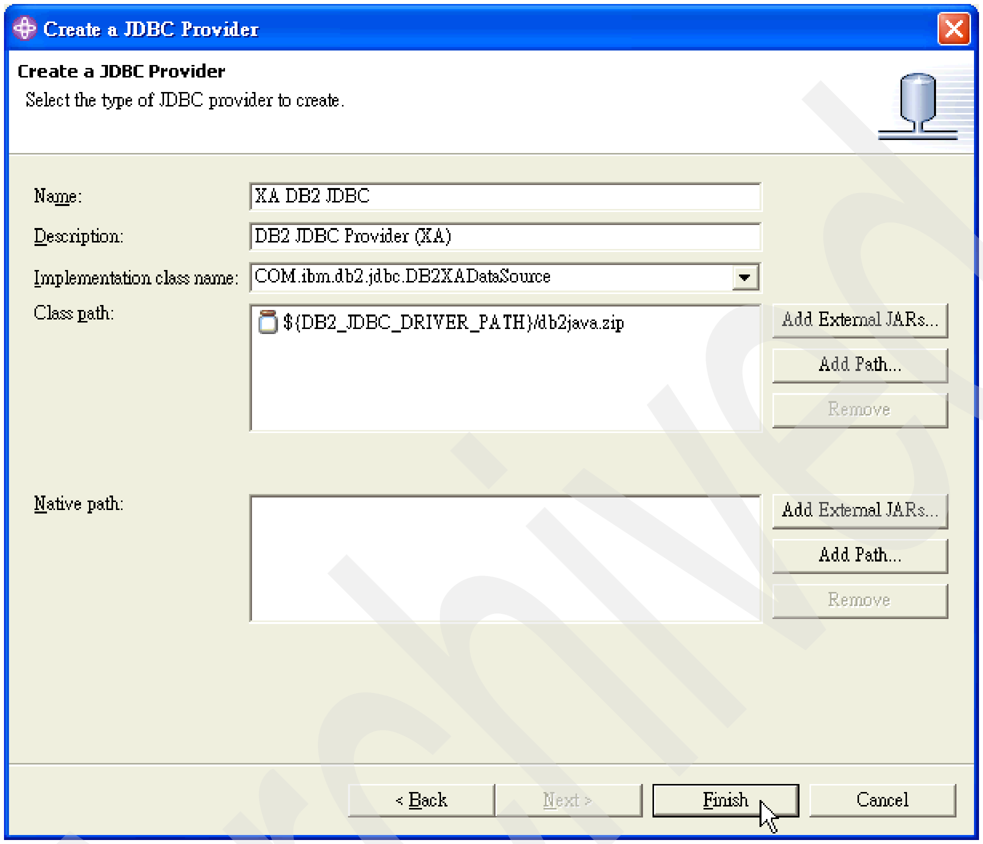
JDBC provider type:

- DB2 Legacy CLI-based Type 2 JDBC Driver
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
- DB2 JDBC Provider
- DB2 JDBC Provider (XA)
- DB2 UDB for iSeries (Native - V5R2 and later)
- DB2 UDB for iSeries (Native XA - V5R2 and later)

Description: Deprecated - Please use the DB2 Legacy CLI-based Type 2 JDBC Driver (XA) provider instead.

Figure 8-19 Create a JDBC provider

e. Name this provider `XA DB2 JDBC` in the next page of the wizard as shown in Figure 8-20, and click **Finish** to create the provider.

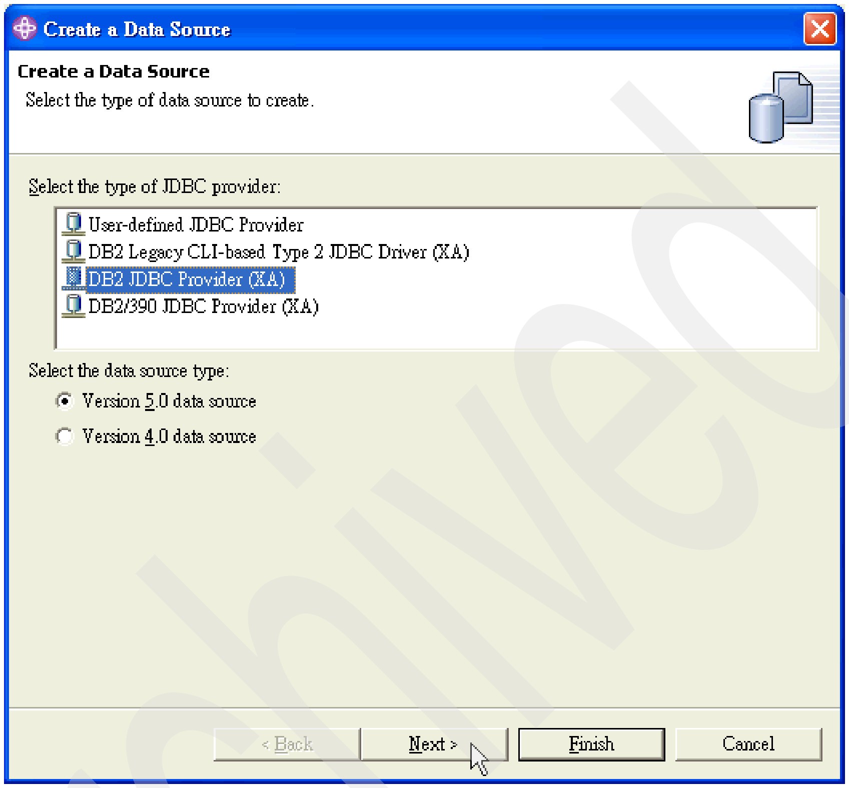


Name: XA DB2 JDBC
Description: DB2 JDBC Provider (XA)
Implementation class name: COM.ibm.db2.jdbc.DB2XADataSource
Class path: ${DB2_JDBC_DRIVER_PATH}/db2java.zip
Native path:

Figure 8-20 Name this provider XA DB2 JDBC

4. Add a data source definition.
   a. Ensure that the **XA DB2 JDBC** provider is selected.
   a. Click the second **Add** in this page.

b. Select **DB2 JDBC Provider (XA)**, and then click **Next**, as shown in Figure 8-21.


Create a Data Source

Create a Data Source

Select the type of data source to create.

Select the type of JDBC provider:

- User-defined JDBC Provider
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
- DB2 JDBC Provider (XA)
- DB2/390 JDBC Provider (XA)

Select the data source type:

[✓ Version 5.0 data source]

Version 4.0 data source

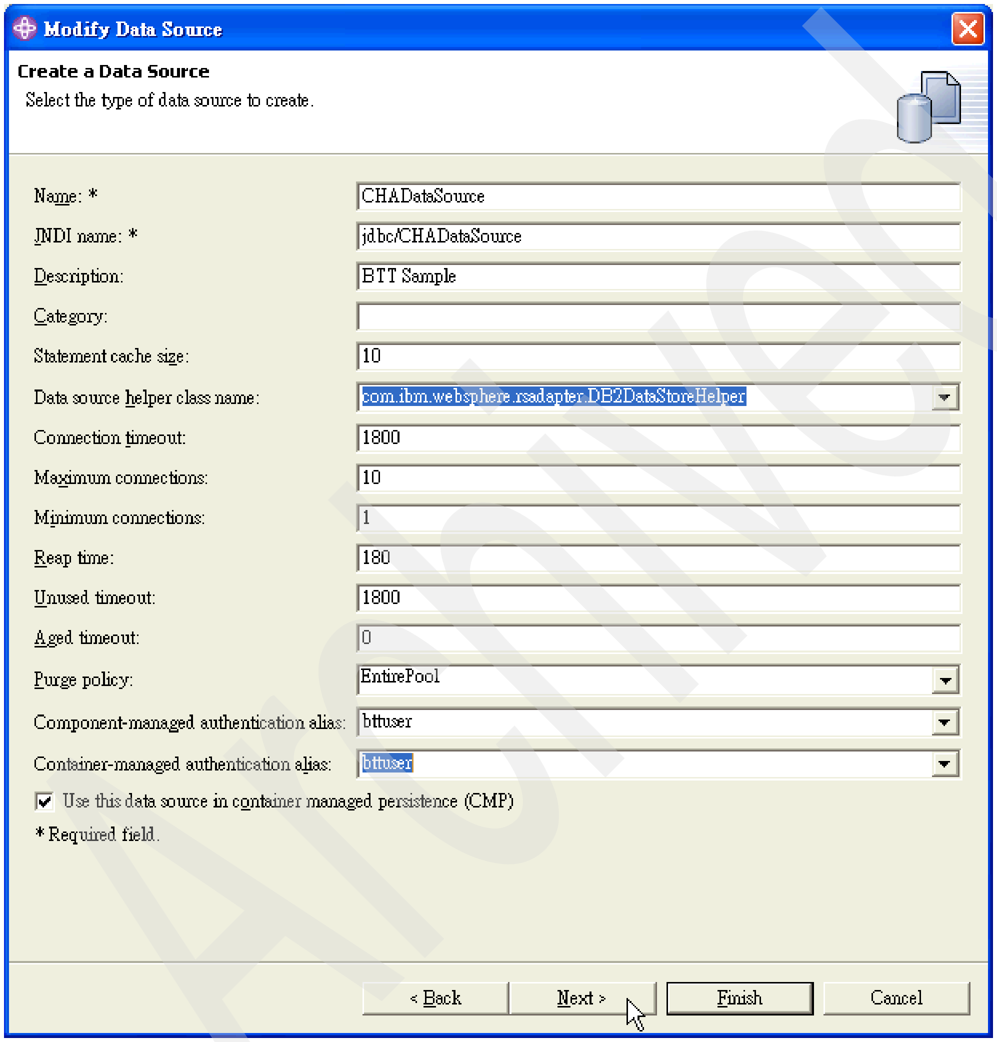
Figure 8-21 Select DB2 JDBC Provider (XA) provider for the data source

c. In the next dialog box, enter the following properties to configure the data source:

- Name: `CHADataSource`
- JNDI name: `jdbc/CHADataSource`
- Description: `BTT Sample`
- Component-managed authentication alias: `bttuser`
- Container-managed authentication alias: `bttuser`

Make sure you have selected the check box against **Use this data source in container managed persistence (CMP)**. For other fields, retain the default values, as shown in Figure 8-22.

d. Click **Next**.



Modify Data Source

Create a Data Source

Select the type of data source to create.

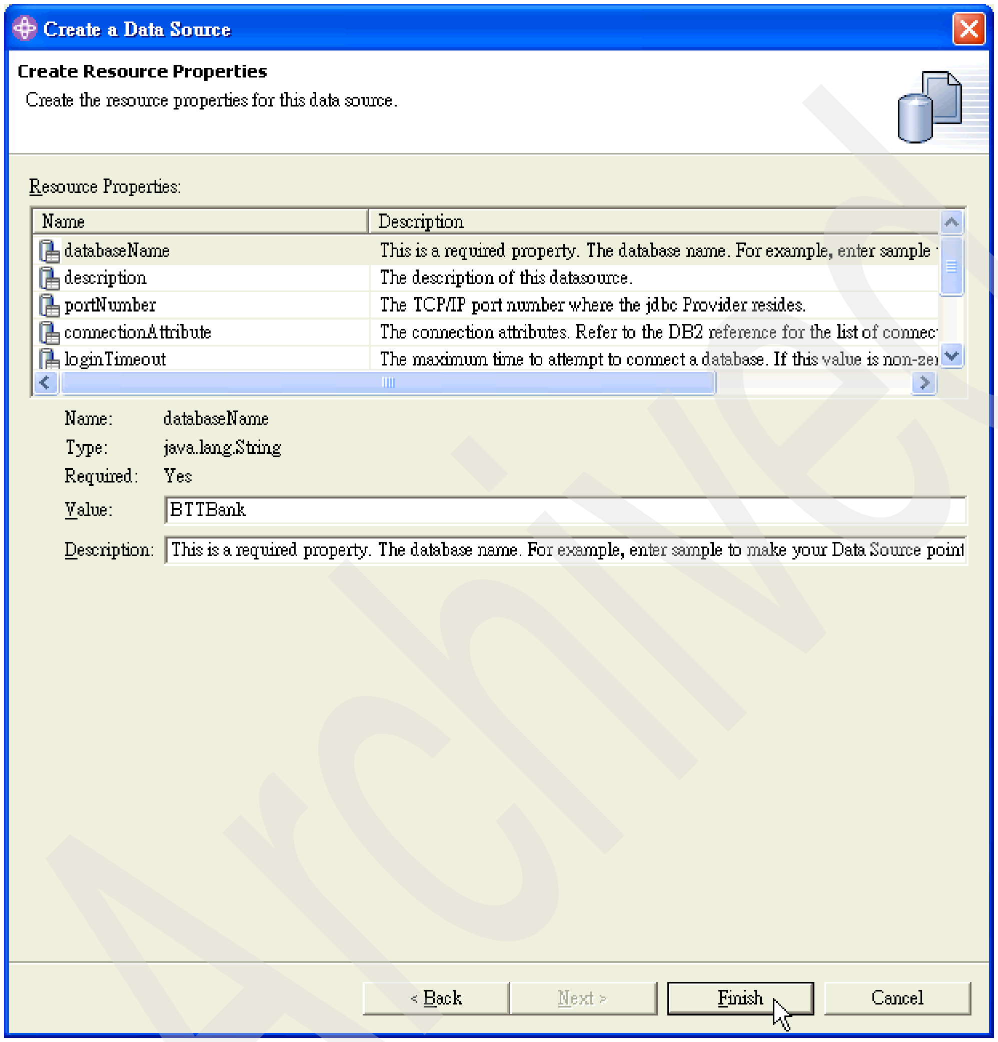
Name: *	CHADataSource
JNDI name: *	jdbc/CHADataSource
Description:	BTT Sample
Category:	
Statement cache size:	10
Data source helper class name:	com.ibm.websphere.rsadapter.DB2DataStoreHelper
Connection timeout:	1800
Maximum connections:	10
Minimum connections:	1
Reap time:	180
Unused timeout:	1800
Aged timeout:	0
Purge policy:	EntirePool
Component-managed authentication alias:	bttuser
Container-managed authentication alias:	bttuser

[✓] Use this data source in container managed persistence (CMP)

* Required field.

Figure 8-22 The settings for CHADataSource

e. In the next dialog box, change the databaseName from `sample` to `BTTBank`, and then click **Finish**, as shown in Figure 8-23.



Create Resource Properties

Create the resource properties for this data source.

Resource Properties:

Name	Description
databaseName	This is a required property. The database name. For example, enter sample
description	The description of this datasource.
portNumber	The TCP/IP port number where the jdbc Provider resides.
connectionAttribute	The connection attributes. Refer to the DB2 reference for the list of connec
loginTimeout	The maximum time to attempt to connect a database. If this value is non-ze

Name:	databaseName
Type:	java.lang.String
Required:	Yes
Value:	BTTBank
Description:	This is a required property. The database name. For example, enter sample to make your Data Source point

Figure 8-23 Specify database name

f. Save the server configuration by pressing the Ctrl+S keys.

8.3.3 Creating the context hierarchy with CHA Editor

The CHA Editor provides a graphical and simple way for users to create and edit the dse file set. The dse file set stores the configuration information used by the

Branch Transformation Toolkit's runtime framework. This configuration information includes runtime settings, contexts, data elements, types, and so on. Combined with the formatting service, both provide the key value for Branch Transformation Toolkit developers. See Figure 8-24.

I

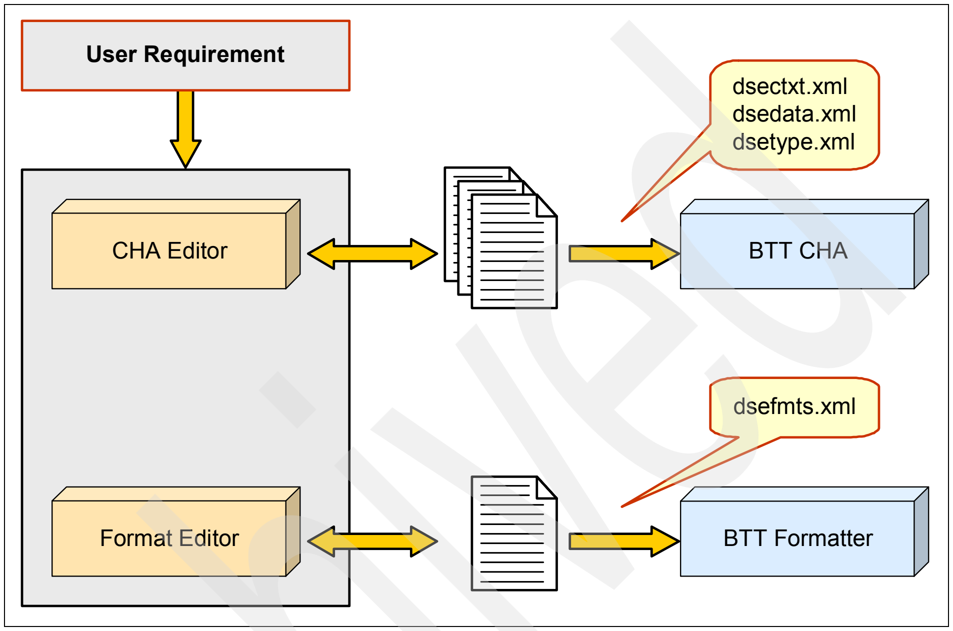


Figure 8-24 CHA Editor and Format Editor are tools used to manipulate dse file set

Due to the close relationship between the CHA Editor and the Format Editor, this section first describes using the CHA Editor, followed by the Format Editor.

Introduction

The CHA Editor provides a graphical and easier way to work with CHA contexts and their data elements and types. Since they are defined within the XML files, it is possible to define, modify, and delete CHA contexts, data elements, and types, using any text editor. However, as the number of definitions in the files increase and the CHA structure increases in size and complexity, maintaining a mental picture of the entire CHA structure becomes increasingly difficult. The CHA Editor provides a visual representation of the structure and relieves you of the need to deal with XML tags directly. This allows you to concentrate on business requirements and issues.

The working of the CHA Editor

When the CHA Editor initializes, it reads the CHA configuration file in the project. This file identifies the context, data, and type definition XML files. By default, these files are called dsectxt.xml, dsedata.xml, and dsetype.xml. The CHA Editor then loads and parses the definitions to build the CHA data structures. The CHA Editor displays the CHA contexts, data elements, types, and their definitions in a set of views.

When you make an addition, modification, or deletion to the CHA contexts and data definitions within one view in the CHA Editor, the Editor updates the other views to reflect the changes. For example, if you create a CHA context within the CHA page of the Editor view, the Editor adds the CHA context to the Outline view and adds the definition for the new CHA context to the CHA definition file.

Note: Branch Transformation Toolkit does not support forward references, that is, the definition being referred to must appear before the definition making the reference. The order in which definitions appear in the definition files might not match the order used to display the contexts, data elements, and types in the various views of the CHA Editor.

Views

The CHA Editor consists of the following views:

- Editor view: This is the primary view of the CHA Editor. It consists of a set of tabbed pages in which you can edit values used by the CHA:
  - CHA page: This page displays the CHA hierarchy trees that describe the relationships between various CHA contexts. Each node in the hierarchy tree is a rectangular container containing a single keyed collection. Additionally, the Editor panel has a palette that provides following tools:
    - Select: This is used to select CHA contexts.
    - Connection: This is used to chain contexts into hierarchies.
    - Context Node drawer
    - New Context. This is used to create a CHA context in the Editor.
  - Configuration page: This page displays the contents of the CHA Editor's configuration file, that is, the ones with the *.chae extension.
  - Context file page: This page displays the contents of the CHA context definition file. It displays the CHA contexts sorted alphabetically. The tab displays the name of the file. For example, if the default file is used, the name in the tab is dsectxt.xml.
  - Data file page: This page displays the contents of the data definition file. It displays the data elements sorted by category, that is, field, data, keyed collection, indexed collection, and then alphabetically. The tab displays the name of the file. For example, if the default file is used, the name in the tab is dsedata.xml.

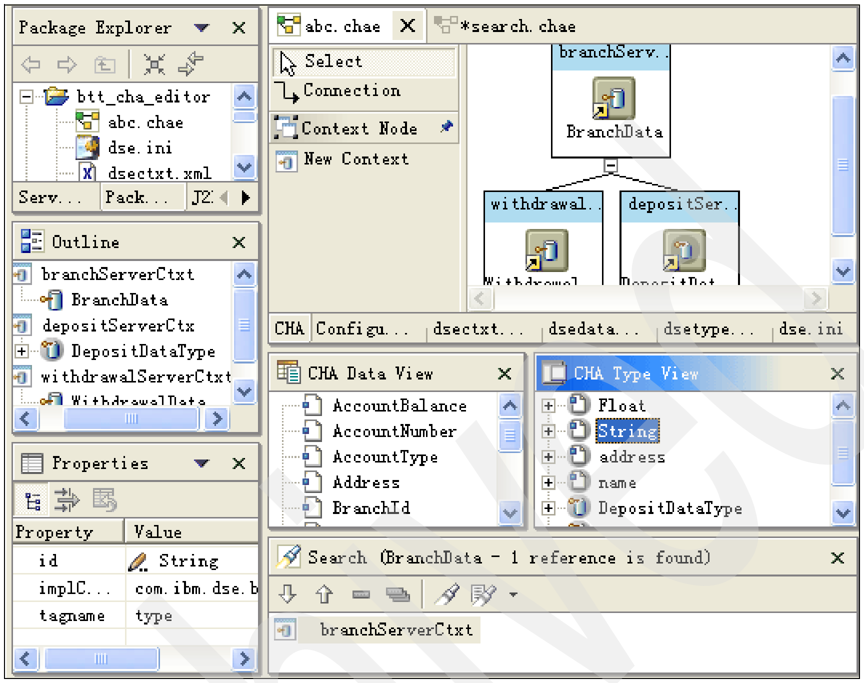
  – Type file page: This page displays the contents of the type definition file. It displays the types sorted alphabetically. The tab displays the name of the file. For example, if the default file is used, the name in the tab is dsetype.xml.

- CHA Data View: This lists all the data definitions. If this view is not visible, you can make it visible by selecting **Window** → **Show View** → **Other** from the main menu. In the IBM Branch Transformation Toolkit folder, select **CHA Data View** when the Show View dialog box opens, and click **OK**.
- CHA Type View: This lists all the type definitions. If this view is not visible, select **Window** → **Show View** → **Other** from the main menu. In the IBM Branch Transformation Toolkit folder, select **CHA Type View** when the Show View dialog box opens, and click **OK**.

In addition, the CHA Editor uses the following standard Application Developer views:

- Outline view lists all the CHA contexts in alphabetical order. You can expand each CHA context to display its data structure.
- Properties view displays the properties of the selected context, data definition, type definition, and so on. These properties map to XML tag or tag attributes such as ID, type, and parent.
- Package Explorer view displays the files created and used by the CHA Editor.
- Search view lists the results when you search the CHA contexts and compound data elements referencing a specific data element.

Figure 8-25 shows the views relating to the CHA Editor.



Property	Value
id	String
implC...	com.ibm.dse.b
tagname	type

Figure 8-25 Various views related to CHA Editor

Creating data elements

To create data elements and context hierarchy for the withdrawal operation, perform the following tasks:

1. Expand **BTTBankBTT** project and double-click the **BTTBankBusiness.chae** file to open it, as shown in Figure 8-26.

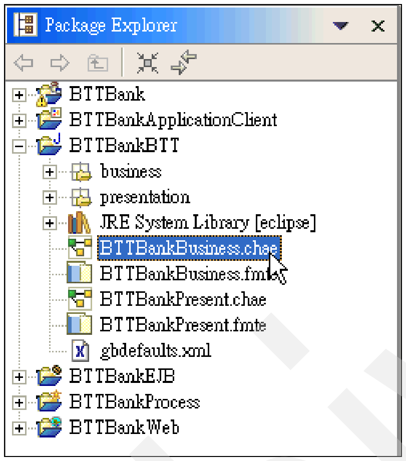


Figure 8-26 Bring up the CHA Editor

2. Right-click **CHA Data View** in the right-hand side of the development workbench, select **Add New Data Element** → **New field** from the context menu, as shown in Figure 8-27.

   A new data element named NewData1 is added in the CHA Data View. Select **NewData1**. It's properties will be shown in the Properties view.

   Note: If you cannot see the Properties view in the workbench, select **Window** → **Show View** → **Properties** to bring it up. By default, the Properties view overlaps the CHA Data View. You can drag the Properties view and drop it in the Tasks view so that you can see both the CHA Data View and Properties view at the same time.

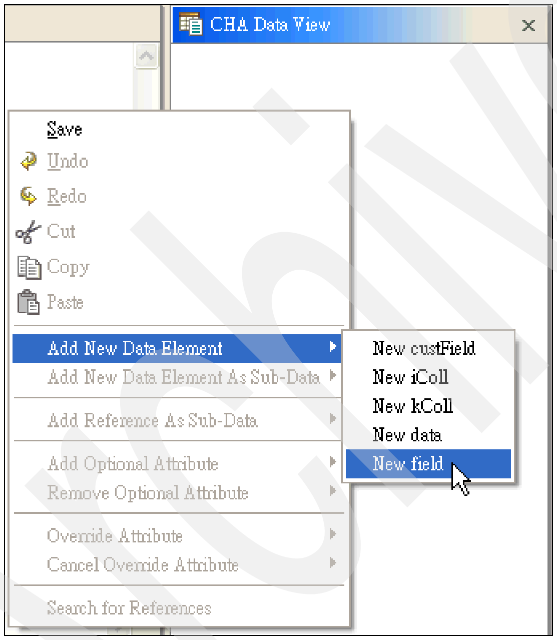
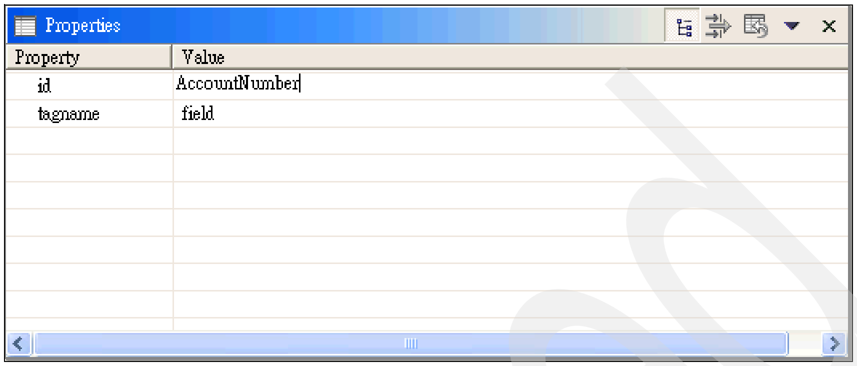


Figure 8-27 Add a new data element

3. Change NewData1's ID to `AccountNumber` in the Properties view and press **Enter**, as shown in Figure 8-28.


Properties

Property	Value
id	AccountNumber
tagname	field

Figure 8-28 Change ID to AccountNumber for the data element

4. Repeat the steps to add the following data elements:
   – AccountBalance
   – Amount
   – Date
   – TrxErrorMessage
   – TrxReplyCode

5. To create a keyed collection, which is often used to manage the structure of data bundles, right-click the **CHA Data View**, and select **Add New Data Element** → **New kColl** from the context menu, as shown in Figure 8-29.

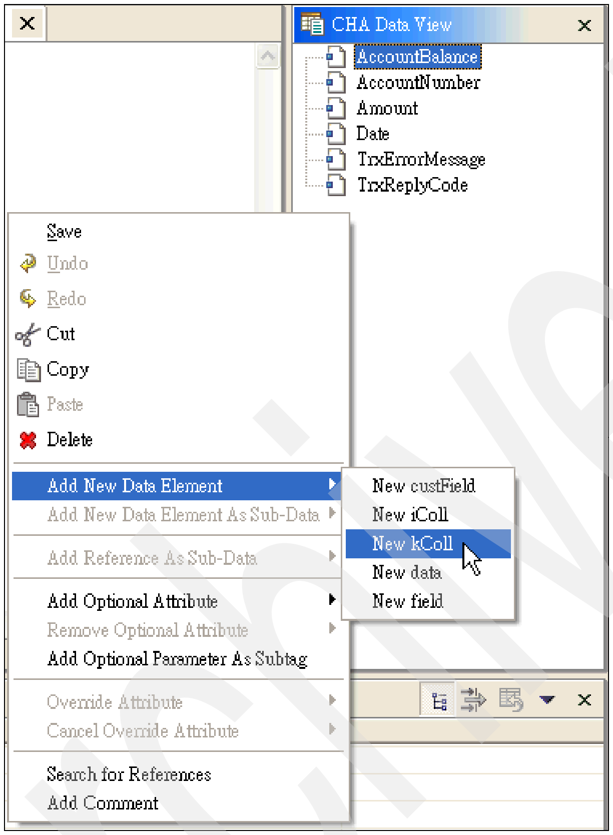
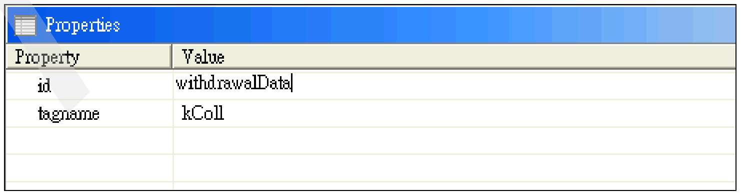


Figure 8-29 Add a keyed collection to organize data

A new keyed collection named NewKeyedColl1 is added in the CHA Data View. Change its ID to `WithdrawalData` in the Properties view and press **Enter**, as shown in Figure 8-30.



Property	Value
id	withdrawalData
tagname	kColl

Figure 8-30 Name the new keyed collection as withdrawalData

6. Add references to other data elements in the keyed collection, as its content. In the CHA Data View, right-click **withdrawalData** element, and select **Add Reference as Sub-Data** from the context menu. You will see the data elements defined earlier. Select the following data elements to add them into withdrawalData collection one by one:

   – **AccountBalance**
   – **AccountNumber**
   – **Amount**
   – **Date**
   – **TrxErrorMessage**
   – **TrxReplyCode**

   See Figure 8-31

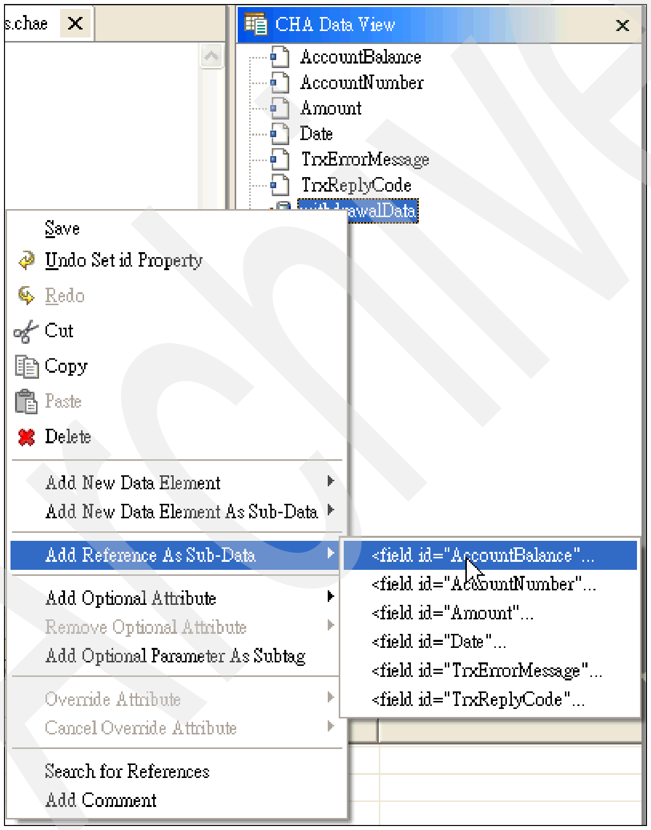


Figure 8-31   Add subdata elements into withdrawalData collection

Note: You can also complete this task by dragging and dropping the corresponding data elements into the withdrawalData in the CHA Data View. If the number of data elements is small, this is a better way to add data elements into collections.

You can see all the subdata elements if you expand **withdrawalData** element in the CHA Data View, as shown in Figure 8-32. Press Ctrl+S to save your work.

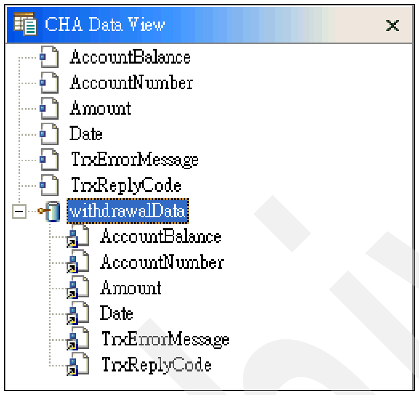


Figure 8-32 Expanded withdrawalData collection

Importing more data elements

You now know how to create fields and keyed collections. However, our sample Branch Transformation Toolkit v5.1 application needs more data elements to run correctly. Perform the following tasks to add the required data elements:

1. Open the file **BTTBankBusiness.chae**.
2. Click **dsedata.xml** tab in CHA Editor.
3. Copy the XML snippet in Example 8-1 and paste it just before the </dsedata.xml> tag in the dsedata.xml file.

Example 8-1 More data elements required to run the sample application

```
<field id="BranchId"/>
<field id="CustomerId"/>
<field id="CustomerName"/>
<field id="dse_sessionId"/>
<field id="dse_pageId"/>
<field id="dse_replyPage"/>
<field id="HostBuff"/>
<field id="sessionID"/>
```

```
<field id="TID"/>
<field id="pw"/>
<field id="userId"/>
<field id="UserId"/>
<field id="WKSContext"/>
<field id="WKSParentContext"/>

<kColl id="branchData">
   <refData refId="BranchId"/>
</kColl>

<kColl id="javaSessionData">
   <refData refId="TID"/>
   <refData refId="CustomerId"/>
   <refData refId="CustomerName"/>
   <refData refId="HostBuff"/>
   <refData refId="sessionID"/>
   <refData refId="UserId"/>
</kColl>

<kColl id="htmlSessionData">
   <refData refId="TID"/>
   <refData refId="CustomerId"/>
   <refData refId="CustomerName"/>
   <refData refId="HostBuff"/>
   <refData refId="dse_sessionId"/>
   <refData refId="dse_pageId"/>
   <refData refId="dse_replyPage"/>
   <refData refId="UserId"/>
</kColl>

<kColl id="signInData">
   <refData refId="userId"/>
   <refData refId="pw"/>
</kColl>

<kColl id="startupServerData">
   <refData refId="TID"/>
   <refData refId="WKSContext"/>
   <refData refId="WKSParentContext"/>
   <refData refId="sessionID"/>
   <refData refId="UserId"/>
</kColl>
```

4. Press Ctrl+S to save the change.

Creating the CHA context

You have to update the empty dsectxt.xml file with CHA Editor. To create a CHA context, perform the following tasks:

1. Open the file **BTTBankBusiness.chae**. Switch to the **CHA** page as needed.
2. Select the **New Context** icon (New Context) in the Context Node drawer. Drop it into the CAH Editor. A new CHA context, a rectangular container with the name New Context1, appears as shown in Figure 8-33.

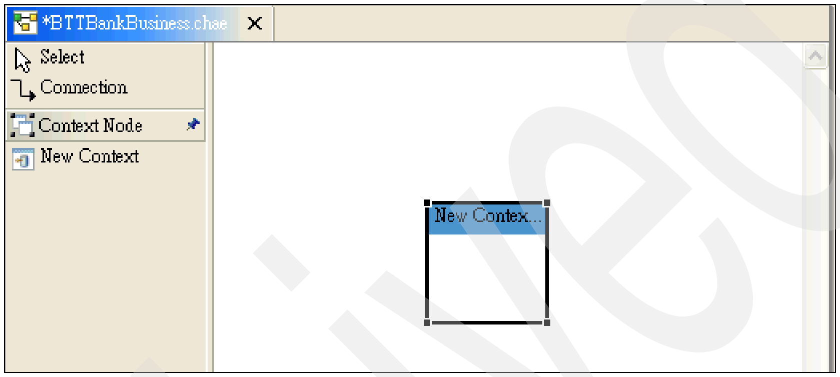


Figure 8-33 New context appears on the canvas of the CHA Editor

> **Note:** You can also do this by right-clicking the blank area in the Editor view and selecting **Create New Context** from the context menu.

3. Select the newly created rectangular container. Change its ID to `withdrawalServerCtx` in the Properties view. Press **Enter**.
4. The new context is complete. To assign a keyed collection data element to it to hold the real data, right-click **withdrawalServerCtx** and select **Add Keyed**

Collection → **Data** → **<kColl id="withdrawalData"** to complete the assignment, as shown in Figure 8-34.

Note: All the valid keyed collections will be listed when you select **Add keyed Collection** → **Data**. In our sample, we defined only one keyed collection. Therefore, Figure 8-34 displays only one option.

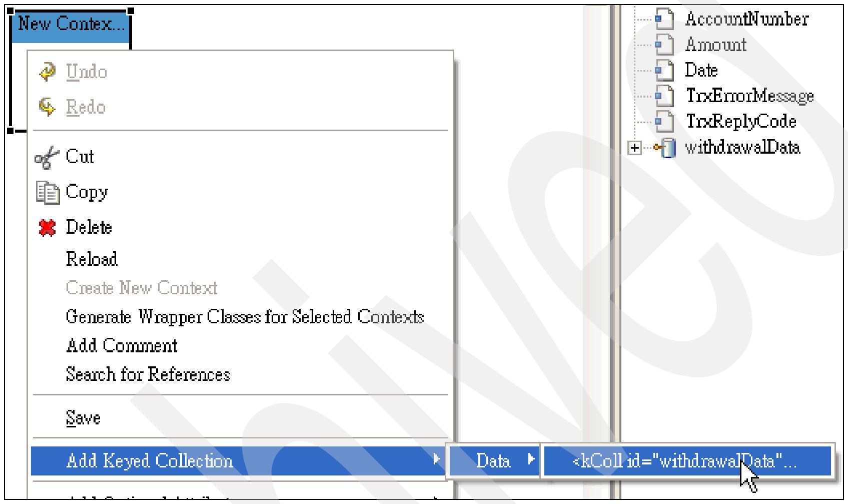


Figure 8-34 Assign a keyed collection data element to withdrawalServerCtx

Note: You can also do this by dragging and dropping the corresponding keyed collection from the CHA Data View to the CHA context directly.

5. Press Ctrl+S to save your work.

The withdrawalServerCtx in the CHA Editor will look as displayed in Figure 8-35:

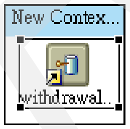


Figure 8-35 withdrawalServerCtx

You can also select the **dsedata.xml** and **dsectxt.xml** tabs to check if the definition XML files are updated properly.

Importing more contexts

To add more contexts for our sample application to run correctly, perform the following tasks:

1. Open the file **BTTBankBusiness.chae**. Click the **dsectxt.xml** tab in the CHA Editor.
2. Copy the XML snippet in Example 8-2 and paste it just before the </dsectxt.xml> tag in the dsectxt.xml file.

Example 8-2 More contexts required to run the sample applications

```
<context id="javaSessionCtx" type="op">
   <refKColl refId="javaSessionData"/>
</context>

<context id="htmlSessionCtx" type="op">
   <refKColl refId="htmlSessionData"/>
</context>

<context id="signInCtx" type="op">
   <refKColl refId="signInData"/>
</context>

<context id="branchServer" type="branch">
   <refKColl refId="branchData"/>
</context>

<context id="startupServerCtx" parent="branchServer" type="op">
   <refKColl refId="startupServerData"/>
</context>
```

3. Press Ctrl+S to save the change.

Configuring the CHA settings in dse.ini

In our sample, because we did not use multiple CHA servers, this option should be turned off. Perform the following tasks to do this:

1. Open the **BTTBankBusiness.chae** file.
2. Click the **dse.ini** tab.
3. Locate the field with the ID supportMultipleCHAServers, in the cha-server keyed collection. Change the value from `true` to `false`.
4. Press Ctrl+S to save your change.

8.3.4 Creating message formats with Format Editor

Now that contexts and data elements are defined, you should create message formats with the Format Editor.

Introduction

The Format Editor provides a graphical and easier way to work with the definition file of formatters. It visualizes formatters in a Graphical Editor. This relieves you from the chores of handling XML tags directly. Common editing features such as cut, copy, paste, delete, undo, redo, load, save, drag-and-drop, reorder, and sort are included in the Format Editor. With these features, you can work with message formats quickly and easily.

The way the Format Editor works

When you double-click a Format Editor file, that is files with *.fmte extension, the Format Editor starts. By default, the definition file dsefmts.xml is loaded. The editor's configuration file also identifies the name of the CHA Editor file with which a Format Editor file works.

Working with the CHA Editor file, the Format Editor displays the CHA contexts, data elements, format elements, and their definitions, in its views. When you make an addition, modification, or deletion to format elements, CHA contexts, or data definitions, the editor updates the other views to reflect the changes. For example, if you create a format definition within the FMTE page of the Editor view, the Editor adds the format definition to the Outline view and adds the definition to the format definition file. This means that you can use either the Editor view or the Outline view to create, modify, and delete format elements.

Note: Branch Transformation Toolkit does not support forward references, that is, the definition being referred to must appear before the definition making the reference. The order in which format definitions appear in definition files may not match the order used to display the format elements, data elements, and CHA contexts in various views of the Format Editor.

Views

The Format Editor consists of the following views:

- Editor view is the primary view of the Format Editor, consisting of a set of tabbed pages in which you can edit values:
  – FMTE page: This page displays format hierarchy trees that describe the relationships between various format definitions. Each node in the

hierarchy tree is a rectangular container that contains a format element. In addition, there is a set of palettes on the left with the following features:

- Select: This is used to select format definitions.
- Format drawer: This drawer contains tools to create a format definition, format elements for CHA contexts, or format elements for data.

  > **Note:** You can use this palette only to create format elements for new CHA contexts or data elements. You cannot use it to create format elements for existing CHA contexts and data elements. To create format elements for existing CHA contexts or data elements, use the context menu in the Outline view or in the main editing area of the FMTE page.

- Add format decorator drawer: This drawer contains tools to add format decorators.

– Configuration page: This page displays the contents of the Format Editor's configuration file, that is, the one with the *.fmte extension.

– Format file page: This page displays the contents of the format definition file. It displays the format definitions in the order in which they are created. The tab displays the name of the file. For example, if the default file is used, the name in the tab is dsefmts.xml.

► CHA Editor View provides a graphical user interface for you to edit CHA contexts. This is the primary view of the CHA Editor.

► CHA Data View lists all the data elements. It first lists all the simple data elements, and then the compound data elements. Both the simple data elements and the compound data elements are in alphabetical order. You can expand each compound data element to display its structure.

► CHA View lists all the CHA contexts in alphabetical order. You can expand each CHA context to display its structure.

► Format Hierarchy View displays the hierarchical structure of the format you selected in the Outline view. The Format Hierarchy View consists of two subviews: the Data/CHA subview, which displays the structure of a data element on the left, and the Format subview, which displays the structure of its format on the right.

  This view does not display the decorators. If this view is not visible, you can make it visible by selecting **Window** → **Show View** → **Other**. Select **Format Hierarchy View** in the IBM Branch Transformation Toolkit folder when the Show View dialog box opens, and click **OK**.

► Format View lists all the format definitions in alphabetical order. You can expand each format definition to display its structure. If this view is not visible, select **Window** → **Show View** → **Other**. Then select **Format View** in the IBM Branch Transformation Toolkit folder, when the Show View dialog pops up and click **OK**.

In addition, the Format Editor uses the following standard Application Developer views:

- Outline view lists all the format definitions in alphabetical order. You can expand each format definition to display its structure.
- Properties view displays the properties of the selected format definition, CHA contexts, data definition, and so on. These properties map to XML tag or tag attributes such as ID, type, and parent.
- Package Explorer view displays the files created and used by the Format Editor.
- Search view lists the search results of format definitions referencing a specific format definition.

Figure 8-36 shows the views related to Format Editor.

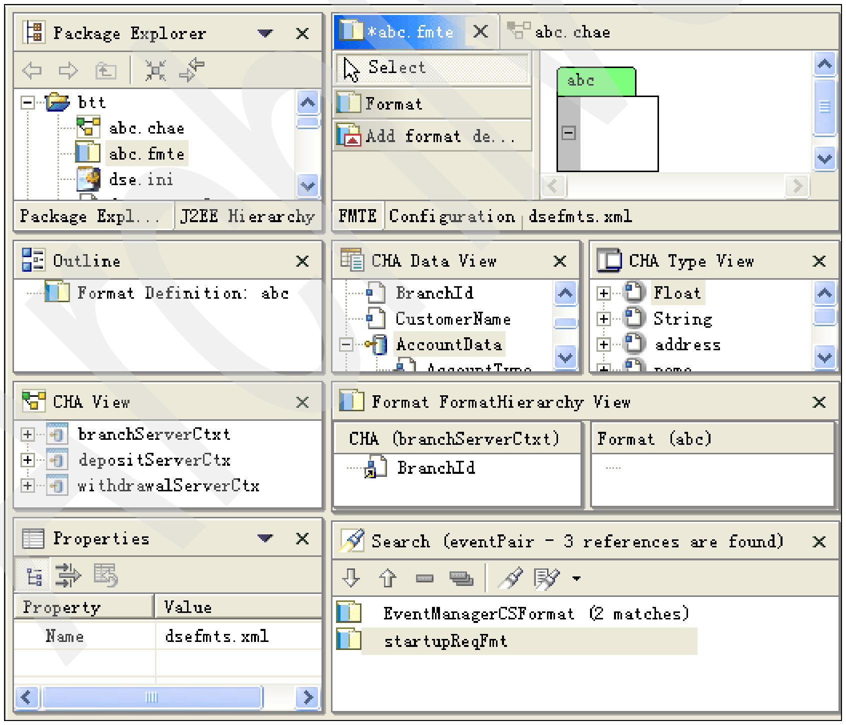


Figure 8-36 Various views related to Format Editor

Creating format definitions and format elements

As seen in 8.1.1, “Application scenario” on page 226, you construct a specific formatted message and send the message to the host. After the host completes the transaction, it will send a reply back to indicate whether the operation was successful or not.

In order to complete the withdraw operation, the sample application should handle the following formatted messages:

Withdrawal request message

A sample of the withdrawal request message can be as follows:

```
Tx02AN=10012002377460000018#DT=28092000#AM=2000#BR=1005#
```

In this message:

- Tx02: Transaction type
- AN: Account number
- DT: Date
- AM: Amount
- BR: Branch ID

Note: Each value ends with the # delimiter.

Withdrawal reply message

A sample of the withdrawal reply message can be as follows:

```
RC=00#BL=1,000#MSG=withdrawalOK#
```

In this message:

- RC: Reply code
- BL: Account balance
- MSG: Error message

Note: Each value ends with the # delimiter.

Two format definitions, withdrawalCSRequestFmt and withdrawalCSReplyFmt, should be created for the withdraw operation.

1. Drag and drop the **Format Definition** icon (Format Definition) from Format drawer to the Editor. The new format definition, that is, the one with a rectangular shape, appears as shown in Figure 8-37.

   Note: You can also do this by right-clicking the blank area in the Editor view, and selecting **Add new format definition** from the context menu.

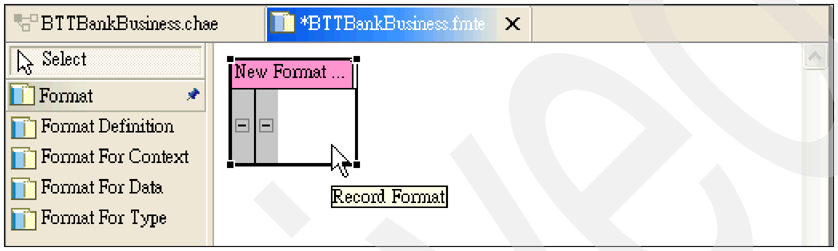
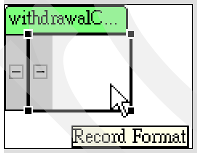


Figure 8-37 New format definition

2. Select the newly created format definition. Change its ID to `withdrawalCSRequestFmt` in the Properties view and press **Enter**.

   Note: By default, a Record format element is also created and contained by the newly created format definition. This is represented as an embedded rectangular within the format definition as shown here.



3. To add the transaction type (Tx02) to the format definition, right-click the **Record Format** of the withdrawalCSRequestFmt and select **Add new format element** → **Constant Format** from the context menu.

   Select **Constant format** and change its Constant Value to `Tx02` in the Properties view and press **Enter**.

4. Add the account number tag, that is, `AN=`, to this format definition, using the Constant Format to complete this task. Right-click the **Record Format**

element and select **Add new format element** → **Constant Format** from the context menu.

Select the new **Constant format** element and change its Constant Value to `AN=` in the Properties view and press **Enter**.

See Figure 8-38 on page 285

Note: Selecting the Record Format element is a bit tricky. Select, click, or right-click it at the right edge of the entire format definition. You can then perform the remaining operations easily.

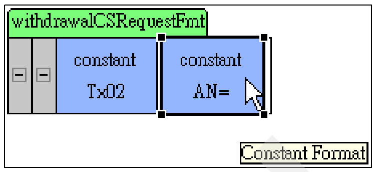


Figure 8-38 Two constant format elements added

5. Right-click the **Record Format** element and select **Add Data element** → **<field id = AccountNumber**. A window pops up, asking you to select the format for the new data. Select **String Format** from the drop-down list and click **Finish**, as shown in Figure 8-39.

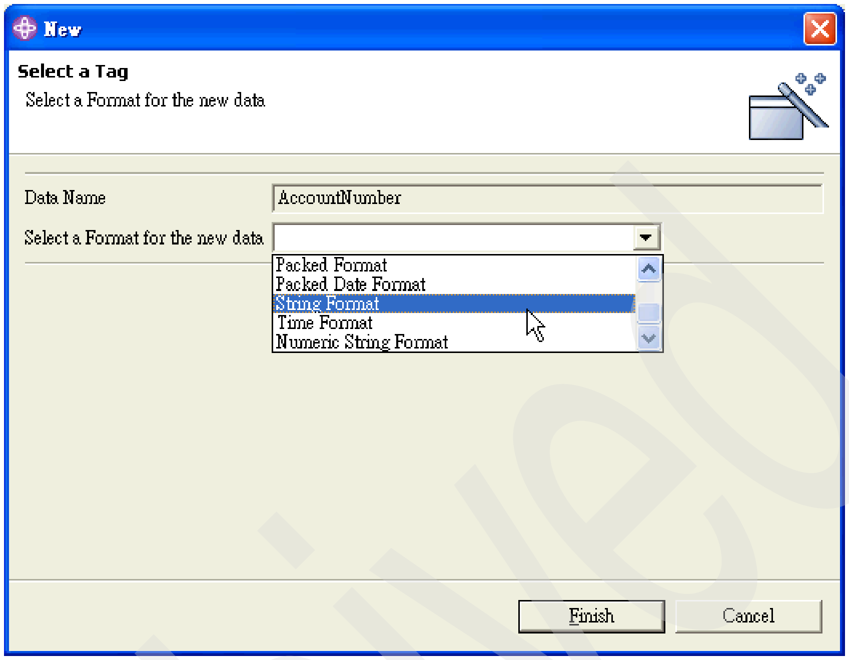


Figure 8-39 Select string format for AccountNumber data element

6. From the Add format decorator drawer in the palette, select the **Null Check Decorator** icon (Null Check Decorator), and click the right edge of the **Record Format** element, as shown in Figure 8-40. Lookfor a null check decorator at the right-most side of the Record Format element.

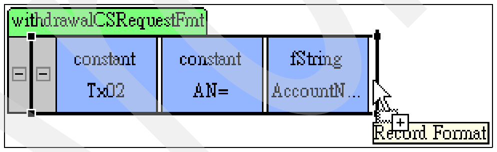


Figure 8-40 Click the right edge of the Record Format element

7. Select the **Delimiter** icon (Delimiter) from the sample drawer and click the right edge of the Record Format element. The delimiter decorator opens. Set its Delimiter Char to # in the Properties view and press **Enter**.

   See Figure 8-41.

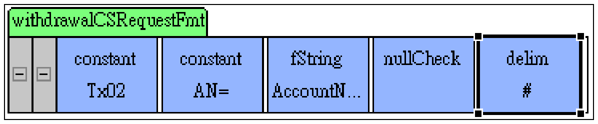


Figure 8-41 withdrawalCSRequestFmt after delimiter decorator is added

8. Similarly, add the remaining format elements at the end of the record in order, according to Table 8-1.

Table 8-1 Other format elements to be added

Formatter	**Data name**	**Constant format**	**Decorator**
Date Format	Date	DT=	Null Check, Delimiter #
Numeric String Format	Amount	AM=	Null Check, Delimiter #
String Format	BranchId	BR=	Null Check, Delimiter #

Figure 8-42 shows the final withdrawalCSRequestFmt.

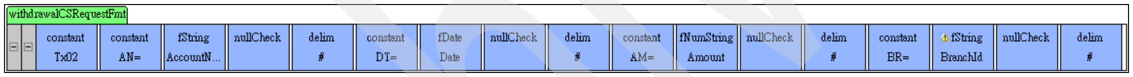


Figure 8-42 Final withdrawalCSRequestFmt

9. Create another format definition with the ID `withdrawalCSReplyFmt`.

10. Add the following format elements to the definition `withdrawalCSReplyFmt`.

Table 8-2 Format elements for withdrawalCSReplyFmt

Formatter	**Data name**	**Constant format**	**Decorator**
String Format	TrxReplyCode	RC=	Null Check, Delimiter #
Numeric String Format	AccountBalance	BL=	Null Check, Delimiter #
String Format	TrxErrorMessage	MSG=	Null Check, Delimiter #

Figure 8-43 shows the final withdrawalCSReplyFmt.

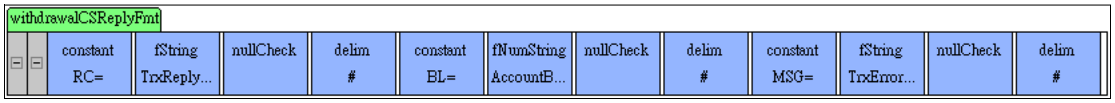


Figure 8-43 Final withdrawalCSReplyFmt

11. Save your work by pressing Ctrl+S.

Importing more formats

In our sample, since we need more formats for it to run correctly, perform the following tasks to add the required formats:

1. Open the file **BTTBankBusiness.fmte**.
2. Click **dsefmts.xml** tab in Format Editor.
3. Copy the XML snippet in Example 8-3 and paste it just before the </dsefmts.xml> tag in the dsefmts.xml file.

Example 8-3 Extra formats required by the sample application

```
<fmtDef id="preSendJournalFmt">
   <hashtable>
      <fObject dataName="UserId"/>
      <fObject dataName="TID"/>
      <fObject dataName="HostBuff"/>
   </hashtable>
</fmtDef>
<fmtDef id="afterRecJournalFmt">
   <hashtable>
      <fObject dataName="TrxReplyCode"/>
      <fObject dataName="AccountBalance"/>
      <fObject dataName="TrxErrorMessage"/>
   </hashtable>
</fmtDef>
```

4. Press Ctrl+S to save the change.

Adding format settings into dse.ini

By default, some settings are not found in dse.ini. If you do not add these configurations, a Branch Transformation Toolkit application might not work as expected. Perform the following tasks to add them:

1. Open the **BTTBankBusiness.chae** file.
2. Click the **dse.ini** tab, locate the keyed collection definition with the ID Files. Add following field definition in it:

```
<field id="format" value="dsefmts.xml"/>
```

3. Locate the keyed collection definition with the ID Formats, and replace the entire keyed collection definition with the one listed in Example 8-4.

Example 8-4 The correct formats keyed collection definition

```
<kColl id="formats" dynamic="false" >
   <field id="fTypedData" value="com.ibm.btt.base.TypedDataElementFormat" description="" />
   <field id="codeSetTrans" value="com.ibm.btt.base.CodeSetTranslator" description="" />
   <field id="compress" value="com.ibm.btt.base.Compressor" description="" />
   <field id="constant" value="com.ibm.btt.base.ConstantFormat" description="" />
   <field id="delim" value="com.ibm.btt.base.Delimiter" description="" />
   <field id="fDate" value="com.ibm.btt.base.DateFormat" description="" />
   <field id="fFloat" value="com.ibm.btt.base.FloatFormat" description="" />
   <field id="fInteger" value="com.ibm.btt.base.IntegerFormat" description="" />
   <field id="fixedLength" value="com.ibm.btt.base.FixedLength" description="" />
   <field id="fNumString" value="com.ibm.btt.base.NumericStringFormat" description="" />
   <field id="fObject" value="com.ibm.btt.base.ObjectFormat" description="" />
   <field id="fString" value="com.ibm.btt.base.StringFormat" description="" />
   <field id="fTime" value="com.ibm.btt.base.TimeFormat" description="" />
   <field id="fXML" value="com.ibm.btt.base.XMLFormat" description="compound" />
   <field id="hashtable" value="com.ibm.btt.base.HashtableFormat" description="compound" />
   <field id="hashtableIColl" value="com.ibm.btt.base.HashtableIndexedCollectionFormat" 
description="compound" />
   <field id="iCollF" value="com.ibm.btt.base.IndexedCollectionFormat" description="compound" 
/>
   <field id="id" value="com.ibm.btt.base.Identifier" description="" />
   <field id="map" value="com.ibm.dse.services.ldap.MapFormat" description="compound" />
   <field id="mapper" value="com.ibm.btt.base.DataMapperFormat" description="compound" />
   <field id="mapperConverter" value="com.ibm.btt.base.DataMapperConverterFormat" 
description="" />
   <field id="mapping" value="com.ibm.dse.services.ldap.Mapping" description="" />
   <field id="maxLength" value="com.ibm.btt.base.MaximumLength" description="" />
   <field id="nullCheck" value="com.ibm.btt.base.NullCheckDecorator" description="" />
   <field id="record" value="com.ibm.btt.base.RecordFormat" description="compound" />
   <field id="refFmt" value="java.lang.Object" description="" />
</kColl >
```

4. Press Ctrl+S to save the change.

8.3.5 The Branch Transformation Toolkit architecture

The IBM IBM Branch Transformation Toolkit for WebSphere Studio is a component-based toolkit for developing enterprise e-business applications. The Branch Transformation Toolkit enables the development of interfaces to the services of a financial institution's information system so that they become ubiquitous through all delivery channels such as the traditional branch, call center, banking kiosk, Internet banking, and mobile access. This minimizes the necessity to develop new code and reduces the time required to make new financial services available to all delivery channels.

The architecture and technological approach of the Branch Transformation Toolkit creates retail delivery solutions that preserve investment in existing enterprise systems while accounting for the inherent instability of any infrastructure due to the innovations that appear frequently in the high-tech industry. While providing a way to preserve the existing systems, the Branch Transformation Toolkit is not tied to one particular platform since it is built on Java^{TM}, the programming language of choice for handling platform change.

The toolkit also takes advantage of existing platforms and technologies such as Eclipse, Web services, J2EE, and so on. The toolkit runtime architecture is based on the J2EE architecture with extensions, and many development tools the toolkit provides are Eclipse plug-ins.

Architecture

The architecture of the toolkit's application is based on a logical three-tier model, that is, back-end enterprise tier, application server tier, and client tier.

Within the application server tier, the toolkit has two separate layers. The application presentation layer is responsible for receiving requests from the client and passing that request to the application logic layer. It also passes the response back to the client. The application logic layer is responsible for performing the request as a process and passing the response back to the application presentation layer.

In general, the application presentation layer resides in a Web container in WebSphere Application Server, while the application logic layer resides in a EJB container. The services are the exception because they can reside anywhere.

The design and portability of the toolkit allow the middle-tier servers to exist at either the branch level, that is, one server per branch, the regional level, that is, one server per group of branches, or even a centralized level, that is, a single server for the entire financial institution. The design provides flexibility to achieve the right balance between the number of servers and the network bandwidth, without affecting any application logic. Besides the application server, there might be a *technical server* responsible for providing common services such as disks or printers to the client workstations. If the application presentation layer and the application logic layer are on the same architectural level, they can physically be on the same machine.

See Figure 8-44 on page 291.

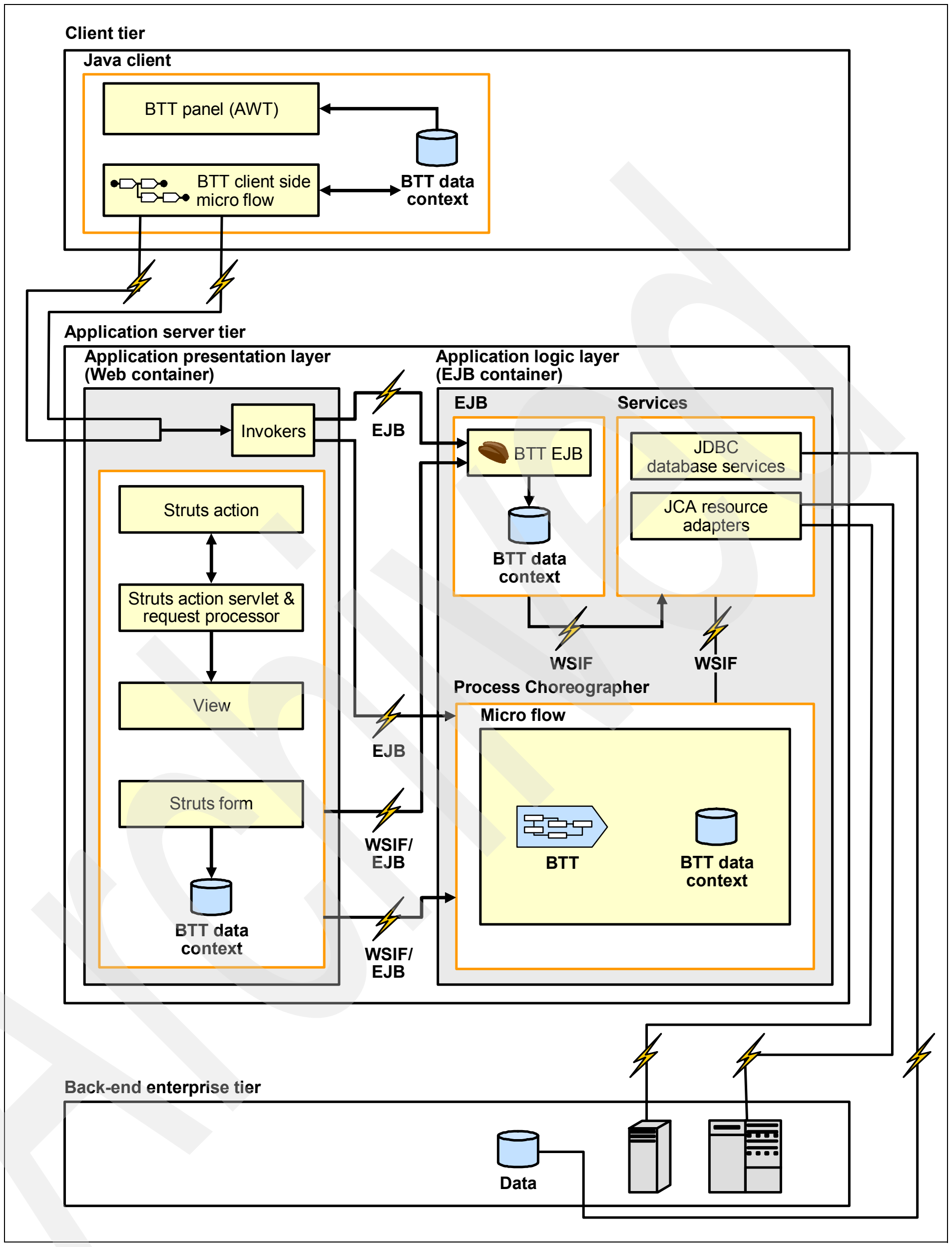


Figure 8-44 Three-tier architecture

Client

A client in the three-tier architecture contains little logic. The logic it does have is usually presentation logic or logic required locally to do such things as access financial devices or validate entered data. The code to execute the client logic is downloaded on an on-demand basis, and therefore does not reside on the client, but on a Web server. The Branch Transformation Toolkit supports any kind of physical client device that uses the following technologies:

- Java^{TM} applets in a browser environment
- Java applications
- HTML clients

The toolkit provides implementations for current client technologies, but these concrete implementations anticipate that significant differences might be found when realizing solutions. The toolkit is not limited to these technologies because its design is generic and can be extended to support other technologies.

A clear separation exists between Java clients and HTML clients. For a Java client, the application, which can also be executed inside a browser, can be built from toolkit-provided visual components that are implemented as Java beans, using visual composition. The visual components of the toolkit and the interaction with toolkit services facilitate the implementation of the required application tasks such as interacting with financial devices, database access, and other services. For an HTML client, the flow of the navigation is delegated to the server.

Application presentation layer

The application presentation layer works in conjunction with a system application server such as IBM WebSphere Application Server to provide a layered multiple channel architecture. The application presentation layer works as a bridge that connects the clients with the application logic layer, which performs business transactions. Java^{TM} clients and HTML clients use different application presentation components to connect to the application logic layer.

To get connected with the application logic layer, the presentation layer defines the following entities:

- Java RequestHandler processes a Java client request for a particular type of requester. The toolkit registers these handlers to determine which specific handler it needs for a specific request. For example, there are different RequestHandlers for requests coming from a Java client in a home banking environment, from a Java client in a branch teller environment, and from a Java client in a call center environment. The RequestHandler is responsible for interacting with the client side operations that controls the dialog navigation for a specific client type and for interacting with invokers that call application logic layer transactions.
- Java PresentationHandler processes the reply for a particular type of requester.

- Struts Extensions processes requests from HTML clients, calls application logic layer components for business transactions, and renders presentation to HTML clients based on the business transaction results.

To pass business process requests to the application logic layer, the application presentation layer has the Bean Invoker Factory, which creates invokers, so that the requester can invoke the EJBs that perform the business processes in the application logic layer. The requester can be a request handler from the Java client or an EJB Action from the toolkit Struts Extensions component.

Application logic layer

The application logic layer provides the core business logic using Enterprise JavaBeans™. It does this in a channel-neutral manner, that is, it handles a transfer funds request, whether the request came from a Web client or a kiosk.

The mechanism for performing the business logic is a business process running in the Process Choreographer in WebSphere Business Integration Server Foundation or a business process running as a Single Action EJB. The business process can involve interacting with Web services, host applications using the JCA Connectors, and enterprise data sources to fulfill the request. The toolkit provides a set of services that support the application logic layer by providing connectivity to enterprise data stores or to existing systems.

If the application presentation layer and the application logic layer are both running on WebSphere Business Integration Server Foundation, the presentation layer can use a work area to pass the session IDs to the application logic layer. Otherwise the application presentation layer includes the session ID with the data required to process the business request in the request message.

Tiers and components

Branch Transformation Toolkit architecture consists of components, tiers or logical subsystems, subsystems, and Java™ packages, which can be used to define, view, and package solution technologies. The solution development process identifies the different parts of the architecture. For example, the requirements phase identifies components, the analysis phase identifies tiers and logical subsystems, and the design and coding phases identify physical subsystems and Java packages. Identifying these items during development ensures that the system structure is consistent and has integrity.

- Components: These are the building blocks of toolkit-based solutions. They are relatively independent and discrete parts that satisfy specific business or technical functions. Components have public interfaces that allow their functionality and implementation to evolve over time, and independent of the rest of the solution. To a certain degree, a toolkit-based solution is a group of reusable components, and the current deployed toolkit solutions are reference configurations about how to combine components to solve a technology problem for a customer channel.

- Tiers (Logical subsystems): These are the analytical building blocks of toolkit-based solutions. They represent a partitioning of the system that is independent of the technology and physical implementation. The Branch Transformation Toolkit tiers map to the J2EE solution architecture.
- Subsystems: These represent its physical partitioning for deployment and execution. This is different from tiers, which represents the logical partitions of a toolkit solution. A subsystem is a physically independent part of the system, and therefore, can be run on any computer within the system. A subsystem can be seen as a subset of components inside a domain.
- Java packages: Also called just *packages*, these elements are the way a toolkit-based solution delivers code. Packages provide a way of grouping functionality for a set of related classes. They define the namespace of a class and follow a naming convention that commonly maps to domains and subsystems. The naming convention used by the Branch Transformation Toolkit is as follows:
  - com.ibm.dse.domainName.subsystemName.furtherPackages.Class
  - com.ibm.btt.domainName.subsystemName.furtherPackages.Class

  The "tierName" and "subsystemName" portions of the naming convention are determined by the system architect. The "furtherPackages" portion is for grouping classes into more discrete functional groups within a subsystem, and is determined by the subsystem designer.

Java client components in the client tier

The Java^{TM} client components in the client tier of the J2EE architecture provides the entities for developing the Java client side of an application. These components control the user interface of the Java client, gather data from the user, send requests to the application presentation tier, and receive response from the application presentation tier.

The toolkit architecture allows the externalization of most of these entities, thereby separating data for a specific client operation from the Java code. This reduces coding effort, which facilitates application development, enhancement, and maintenance.

See Figure 8-45 on page 295.

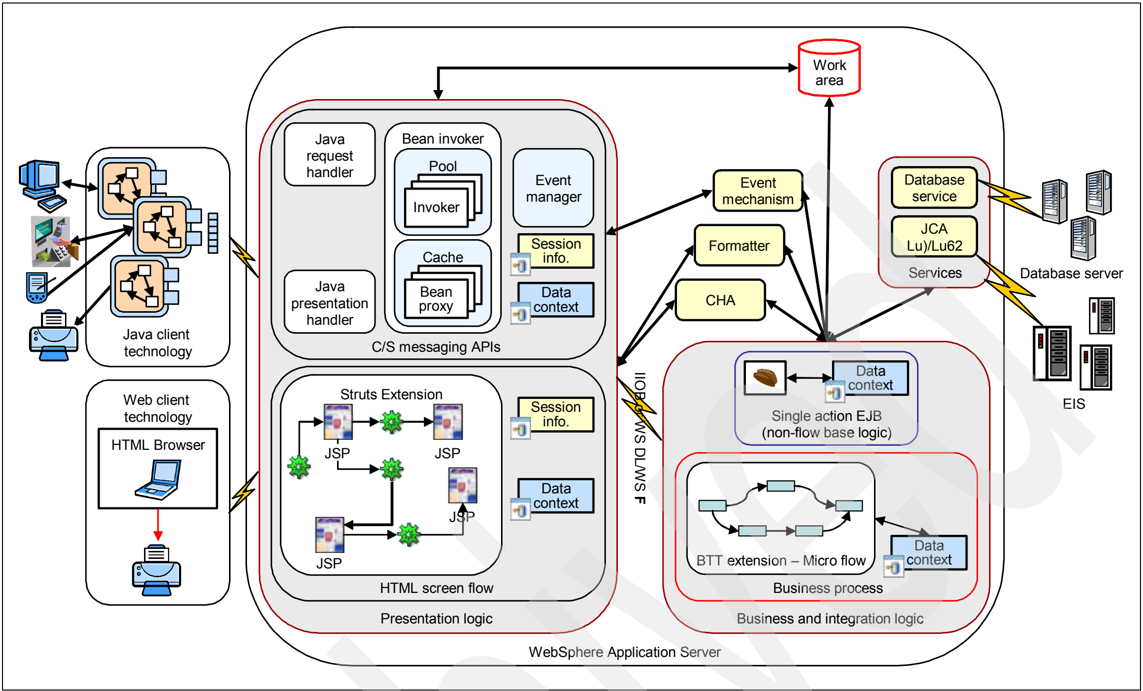


Figure 8-45 The component view of a Branch Transformation Toolkit application

Components in the application server tier

The Branch Transformation Toolkit components that run on application servers can be divided into two categories, that is, components running in the Web container of the application server and components running in the EJB container of the application server. The Web container and the EJB container share some toolkit components. These components are needed both by the application presentation components for handling requests from clients and by the application logic components for doing business transactions. The shared components consist of the following:

- Data elements and typed data

  These elements are similar to those running on the Java^{TM} client. Data elements and typed data elements provide a mechanism for managing data used throughout a transaction.

- CHA and CHA Formatter Service

  The CHA and CHA Formatter Service supports other components in the Web container and EJB container by providing a distributed data structure and mechanism to import and export data into the structure. The CHA uses the contexts, data elements, and typed data elements.

- Events, externalizers, and exceptions

  These components work in the same way as their counterparts running on the Java client.
- Trace facility

  The trace facility provides a class that keeps information in memory and records information on disk.

Application presentation components in the Web container

The application presentation components in the Web container provide the main entities for developing the application presentation layer part of an application. These components control the user interface of HTML clients, gather data from HTML clients, and launch the business processes performed in the application logic layer. This group includes the client/server connectivity components that provide multichannel support connectivity between various client devices and the application presentation layer.

- Invokers

  Instantiated by the Bean Invoker Factory, invokers enable Struts actions or Java^{TM} request handlers to access the business processes and Single Action EJBs through an EJB call.

  When a request comes from a requester (a request handler or a Struts action), the request brings a request ID and a session ID to the Bean Invoker Factory. The request ID indicates what kind of transaction the client is requesting, and the session ID identifies the session of this transaction request. The Bean Invoker Factory generates or allocates an invoker with the request ID and session ID. The Bean Invoker Factory then returns the invoker to the requester so that the requester can send the request to the application logic layer.
- Java Client/Server Messaging APIs

  The Java^{TM} Client/Server Messaging APIs component is an implementation of multichannel support that allows the creation of distributed Java applications - not just distributed data but also distributed logic - using Internet technologies. The Java Client/Server Messaging APIs is based on the HTTP protocol, but adds the concept of a session between the client and the server. The Java Connector supports session clustering, load balancing, and a network dispatcher so that an application can be distributed among several servers.

  Furthermore, the Java Client/Server Messaging APIs component allows a kind of dynamic application topology reconfiguration, which means that at any point in time, the server executing the logic can be changed. A request from a toolkit client implies the execution of a business process or activity in the application logic layer. The request contains the name of the process or activity to execute, along with relevant data for unformatting into the process context. However, it does not specify where to execute the process or activity.

The Java Client/Server Messaging APIs component can use the SSL protocol capabilities, allowing secure information interchange between the client side and the server side of a toolkit-based system. The capability is particularly useful when information is flowing through non-trustworthy networks. The toolkit can use up to 1024-bit RSA for key exchange and 128-bit symmetric encryption of data.

- Struts Extensions

  The toolkit's Struts Extensions enables HTML clients to send requests to and receive responses from a toolkit application using the HTTP protocol. The Struts Extensions provide additional functionalities such as handling the back button in the browser, duplicating requests, NLS, validation, and error handling.

- JavaServer Pages

  JavaServer Pages (JSPs) are used to dynamically construct HTML pages requested by a client Web browser. Any HTML page can contain URL links that cause a JSP to be rendered and returned to the HTML browser. JSP tags render HTML contents based on the attributes of the tag and dynamic information obtained from the context.

  Based on the Apache Struts Framework, the Branch Transformation Toolkit provides custom JSP tags and utility beans to enable applications to retrieve information from the context hierarchy, get resources, and handle errors. If your application requires additional behavior, you can build new tags using the StrutsJspContextServices interface.

Application logic components in the EJB container

The application logic components residing in the EJB container support the execution of business logic.

- Business Process Component

  The Business Process Component provides supporting entities so that an application can run a business process within the Process Choreographer of WebSphere(R) Business Integration Server Foundation.

- Single Action EJB

  As an alternative to the Process Choreographer, an application can use a Single Action EJB to perform the business process. A Single Action EJB may or may not use the CHA and CHA Formatter Service.

- Startup beans

  The application logic domain uses startup beans to initialize the application logic layer entities.

- Communication services

  Communication services provide connectivity to the existing data and applications in the enterprise systems. These services isolate the client application from the communications complexity by providing a clear and easy public interface. The SNA JCA LU0 Connector and the SNA JCA LU62 Connector are resource adapters that enable business processes.

- JDBC Database services

  The JDBC Database services of the Branch Transformation Toolkit interact with a database through the JDBC protocol to provide access to database tables. The services also map context data to database records and database records to context data using a formatter. The Database Table Mapping service enables toolkit applications to access databases using a common interface. The Electric Journal service can record the services and processes used or performed by an entity such as a branch, user, or terminal. The design of each application determines what information the service records, as well as when it writes that information.

- Generic Pool

  The Generic Pool service in the application logic layer performs the same function as it does in the Java™ clients and Web Container.

8.3.6 Setting up CHA and formatter services

The Branch Transformation Toolkit application consists of numerous components. A toolkit application is developed by composing all the required components. The CHA and Formatter services are the core components in a toolkit application. This section describes the process involved in adding these two components into the sample application.

Adding the CHA service

Now that the definitions of the CHA data and the CHA context is complete, you can move further. The Branch Transformation Toolkit provides the BTTCHAEJB EJB module, which converts these definitions into live objects in the runtime. You should now import the CHA EJB module.

Removing invalid EJB modules

Before importing the CHA EJB module, remove the three invalid EJB modules created by BTT Project wizard by performing the following tasks:

1. Expand the **BTTBank** project and open the **deployment descriptor (application.xml)** stored in the META-INF directory.

2. Click the **Module** tab, select **EJB BTTFormatterEJB.jar, EJB bttsvcinfra.jar,** and **EJB BTTCHAEJB.jar**, and click **Remove**, as shown in Figure 8-46.

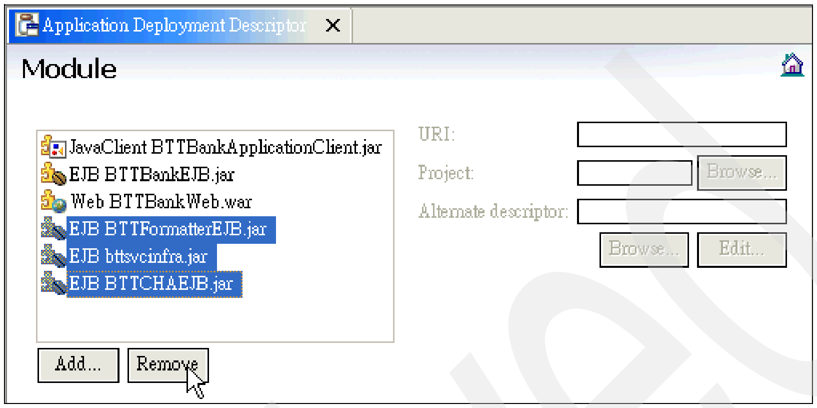


Figure 8-46 Remove unused EJB modules

3. Press Ctrl+S to save the change.

Import BTTCHAEJB.jar

To import the CHA EJB JAR file, perform the following steps:

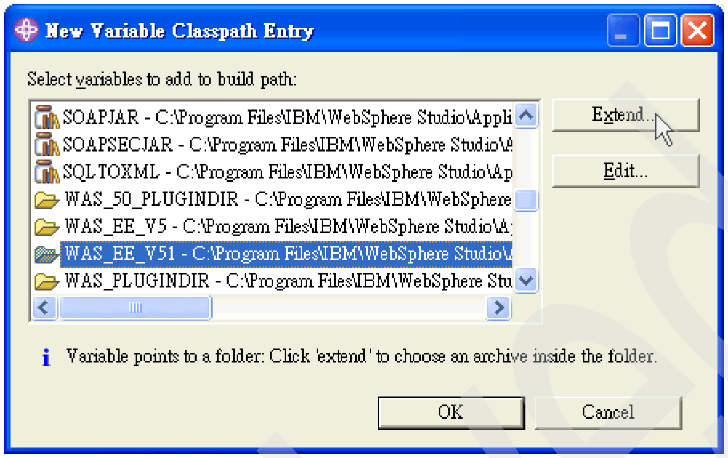
1. Open the **J2EE** perspective, and select **File** → **Import** from the menu bar.
2. In the Import dialog box, select **EJB JAR file** and click **Next.**
3. Enter the following values for the EJB import:
   - EJB JAR File: `<BTT_install_dir>\jars\BTTCHAEJB.jar`
   - EJB project: `BTTCHAEJB`
   - EAR project: `BTTBank`
4. Click **Finish**.

Set up project properties

To set up the project properties, follow these steps:

1. Switch to the **Project Navigator** view. The BTTCHAEJB EJB Project appears. You will see that it has some errors.
2. Right-click the **BTTCHAEJB** project, and select **Properties** from the context menu.
3. In the pop-up window, select **Java Build Path**, and then click the **Libraries** tab.

4. Click **Add Variable**. In the New Variable Classpath Entry pop-up window, select **WAS_EE_V51** and click **Extend**, as shown in Figure 8-47.


New Variable Classpath Entry
Select variables to add to build path:
SOAPJAR - C:\Program Files\IBM\WebSphere Studio\Appli
SOAPSECJAR - C:\Program Files\IBM\WebSphere Studio\A
SQLTOXML - C:\Program Files\IBM\WebSphere Studio\Ap
WAS_50_PLUGINDIR - C:\Program Files\IBM\WebSphere
WAS_EE_V5 - C:\Program Files\IBM\WebSphere Studio\A
WAS_EE_V51 - C:\Program Files\IBM\WebSphere Studio\
WAS_PLUGINDIR - C:\Program Files\IBM\WebSphere Stu
Extend...
Edit...
OK
Cancel

Variable points to a folder: Click 'extend' to choose an archive inside the folder.

Figure 8-47 Extend the variable WAS_EE_V51 to add a jar file

5. In the Variable Extension pop-up window, select **lib/startupbean.jar,** and click **OK**, as shown in Figure 8-48.

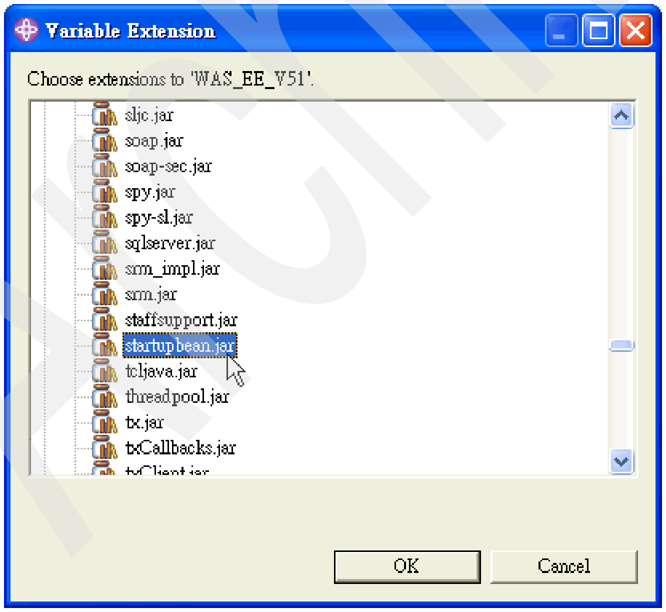


Figure 8-48 Add lib/startupbean.jar to the build path

6. Again, in the **Libraries** page, click **Add JARs**. Select **bttbase.jar** from the BTTBank project, then click **OK**, as shown in Figure 8-49.

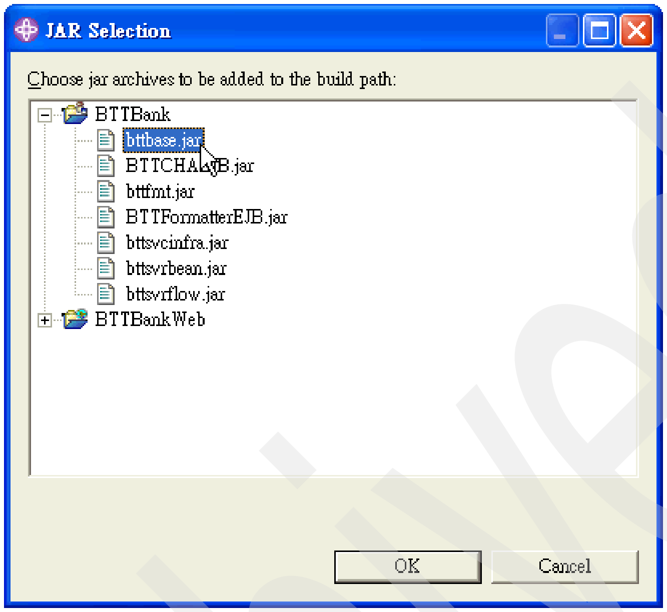


Figure 8-49 Add bttbase.jar into the build path

7. Click **OK** again in the BTTCHAEJB Properties window. The WebSphere Studio Application Developer Integration Edition will rebuild the project automatically. In addition, all the errors will be fixed.

> **Note:** If WebSphere Studio Application Developer Integration Edition 5.1.1 does not rebuild the project automatically, select the project, and then select **Project** → **Rebuild Project** from the menu bar.

Generating EJB to RDB mapping

To generate the EJB to RDB mapping, follow these steps:

1. Right-click **BTTCHAEJB** project, select **Generate** → **EJB to RDB Mapping** from the context menu, as shown in Figure 8-50.

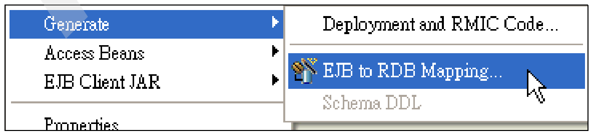
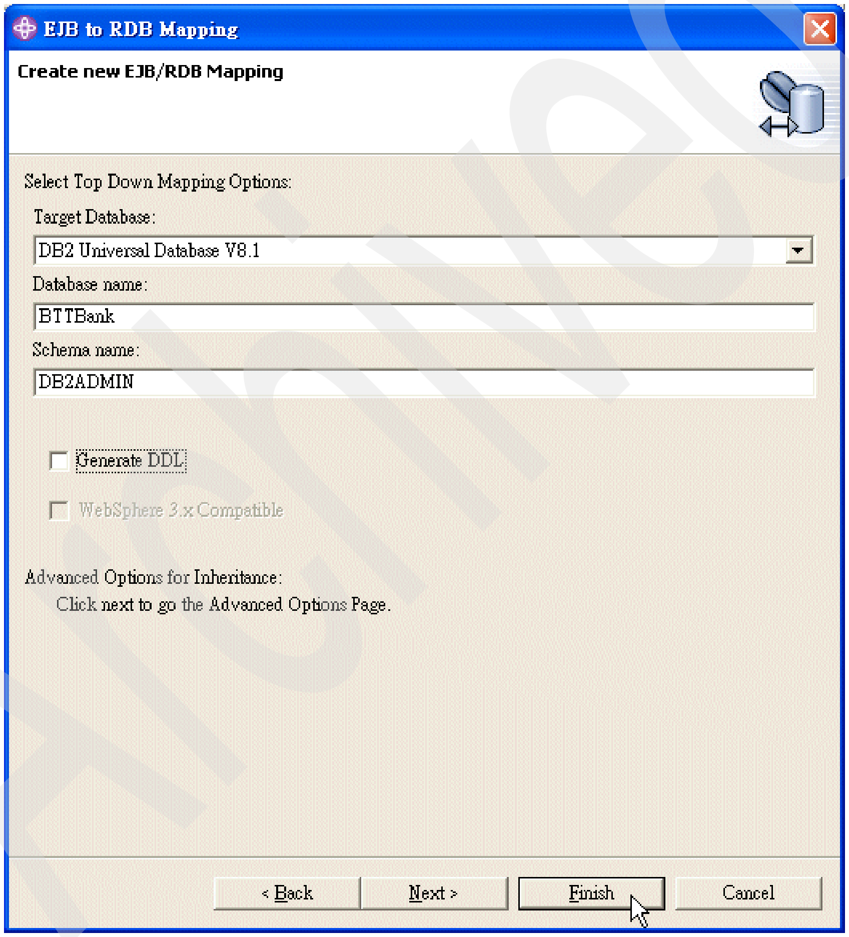


Figure 8-50 Select Generate → EJB to RDB Mapping

2. Select **Create a new back-end folder**, and click **Next**.
3. Select **Top Down**, and click **Next**.
4. Enter the following values for the Top Down Mapping Options, as shown in Figure 8-51:
   - Target Database: `DB2 Universal Database V8.1`
   - Database name: `BTTBank`
   - Schema name: `DB2ADMIN`.

   Uncheck the check box against **Generate DDL**, and click **Finish**.



Select Top Down Mapping Options:

Target Database: DB2 Universal Database V8.1

Database name: BTTBank

Schema name: DB2ADMIN

[ ] Generate DDL

[ ] WebSphere 3.x Compatible

Advanced Options for Inheritance:
Click next to go the Advanced Options Page.

Figure 8-51 Specify top-down mapping options

Generating Deployment and RMIC Code

To generate the deplyment and RMIC code, follow these steps:

1. Right-click **BTTCHAEJB** project, and select **Generate** → **Deployment and RMIC Code** from the context menu, as shown in Figure 8-52.

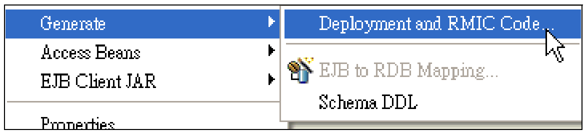
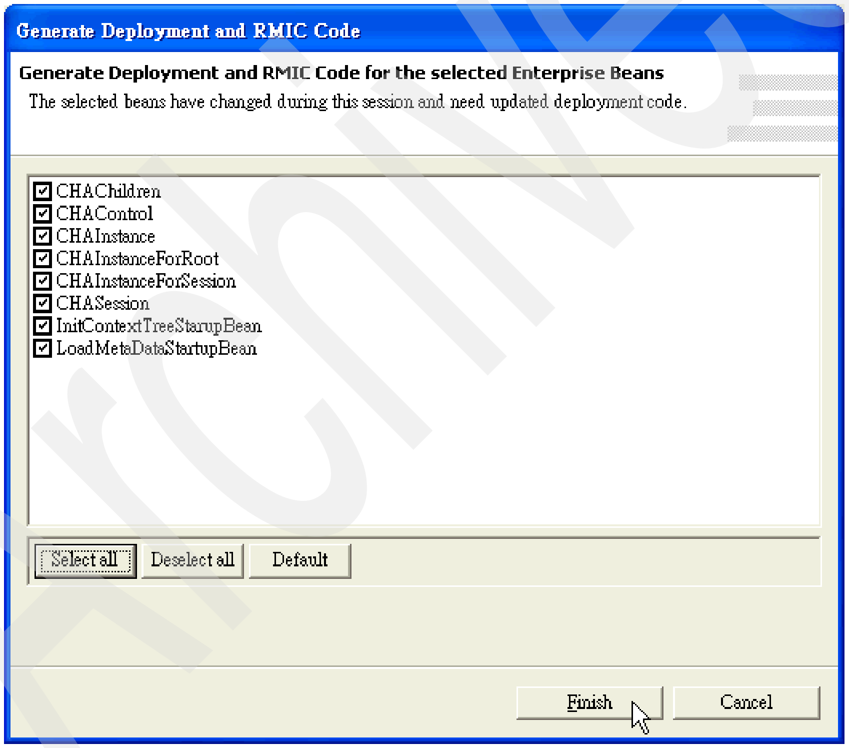


Figure 8-52 Select Generate → Deployment and RMIC Code

2. Click **Select all** in the pop-up window, and click **Finish**, as shown in Figure 8-53.



Generate Deployment and RMIC Code for the selected Enterprise Beans

The selected beans have changed during this session and need updated deployment code.

- ☑ CHAChildren
- ☑ CHAControl
- ☑ CHAInstance
- ☑ CHAInstanceForRoot
- ☑ CHAInstanceForSession
- ☑ CHASession
- ☑ InitContextTreeStarupBean
- ☑ LoadMetaDataStartupBean

Figure 8-53 Generate Deployment and RMIC Code for all enterprise beans

Adding the CHA Formatter service

The Branch Transformation Toolkit v5.1 also provides the BTTFormatterEJB EJB module, which converts the defined formats into live objects in the runtime.

Before deploying CHA Formatter Service, ensure that the CHA environment is ready in the workspace.

Importing BTTFormatterEJB.jar

To import the BTTFormatterEJB.jar, follow these steps:

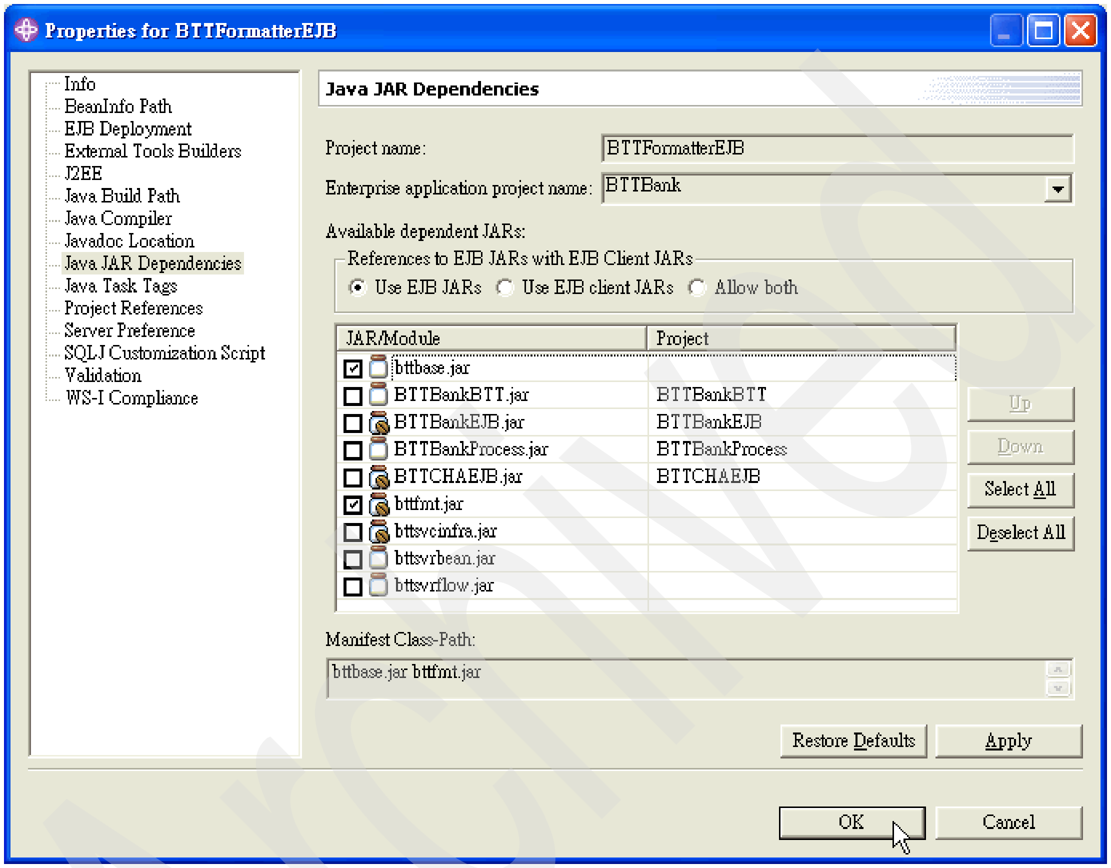
1. Open the **J2EE** perspective and get into the J2EE Hierarchy view.
2. Right-click **BTTBank** in the Enterprise Applications folder. Select **Import** → **Import EJB Jar**.
3. Enter the following values for the EJB import:
   - EJB JAR File: `<BTT_install_dir>\jars\BTTFormatterEJB.jar`
   - EJB project: `BTTFormatterEJB`
   - EAR project: `BTTBank`
4. Click **Finish**.

Setting up project properties

To set up the project properties, follow these steps:

1. Switch to the **Project Navigator** view. The BTTFormatterEJB EJB Project appears.
2. Right-click the **BTTFormatterEJB** project, and select **Properties** from the context menu.

3. In the pop-up window, select **Java JAR Dependencies**. Select the **Use EJB JARs** option, and then check **bttbase.jar** and **bttfmt.jar** in the JAR/Module list. Click **OK**, as shown in Figure 8-54.



Java JAR Dependencies

Project name: BTTFormatterEJB

Enterprise application project name: BTTBank

Available dependent JARs:

References to EJB JARs with EJB Client JARs

[✓ Use EJB JARs] Use EJB client JARs Allow both

	JAR/Module	Project
✓	bttbase.jar	
☐	BTTBankBTT.jar	BTTBankBTT
☐	BTTBankEJB.jar	BTTBankEJB
☐	BTTBankProcess.jar	BTTBankProcess
☐	BTTCHAEJB.jar	BTTCHAEJB
✓	bttfmt.jar	
☐	bttsvcinfra.jar	
☐	bttsvrbean.jar	
☐	bttsvrflow.jar	

Manifest Class-Path:

bttbase.jar bttfmt.jar

Figure 8-54 Set up Java JAR dependencies for BTTFormatterEJB

Generating Deployment and RMIC code

To generate the deployment and RMIC code, follow these steps:

1. Right-click the **BTTFormatterEJB** project and select **Generate → Deployment and RMIC Code.**

2. In the pop-up dialog box, click **Select all** and **Finish**, as shown in Figure 8-55.

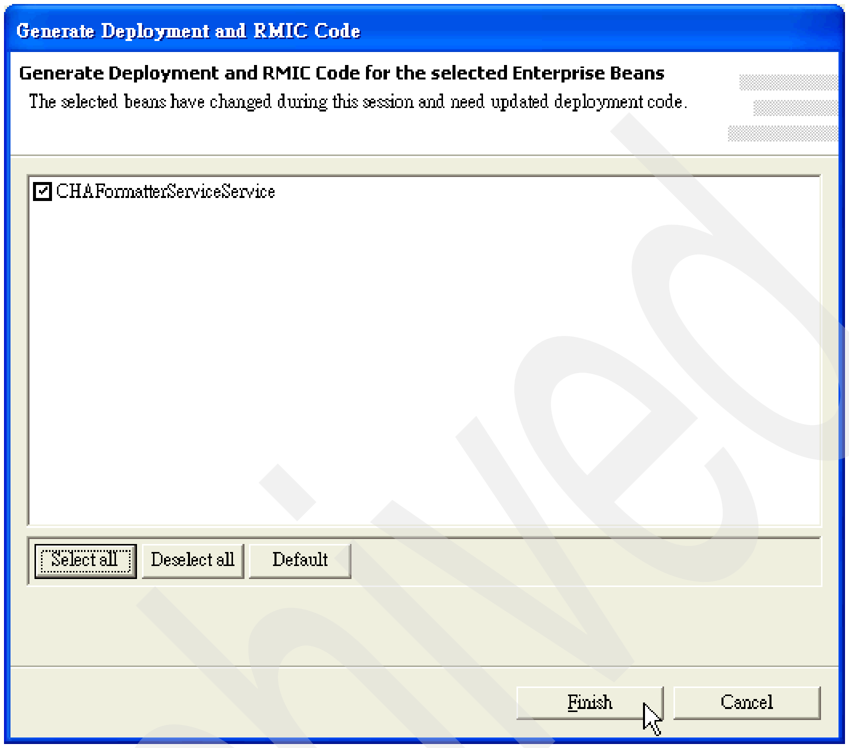

Generate Deployment and RMIC Code
Generate Deployment and RMIC Code for the selected Enterprise Beans
The selected beans have changed during this session and need updated deployment code.
CHAFormatterServiceService
Select all
Deselect all
Default
Finish
Cancel

Figure 8-55 Generate Deployment and RMIC Code for BTTFormatterEJB

Defining proxy in client side dse.ini

This section describes how to define CHA formatter service proxy in client side dse.ini. The CHA formatter service has four proxies. In our sample, we used PureEJBProxy. To define CHA formatter service proxy in client side dse.ini, perform the following steps:

1. Open **BTTBankBusiness.chae** by double-clicking the file in BTTBankBTT project.
2. Click the **dse.ini** tab, locate the keyed collection definition with the ID cha-server. Add the following keyed collection definition shown in Example 8-5 after it.

Example 8-5 Configuring PureEJBProxy for CHA formatter service

```
<kColl id="CHAFormatterServiceClient">
   <field id="CHAFormatterServiceProxyClass"
          value="com.ibm.btt.formatter.client.CHAFormatterServicePureEJBProxy"/>
   <field id="PureEJBProxy_initialContextFactory"
```

```
          value="com.ibm.websphere.naming.WsnInitialContextFactory"/>
   <field id="PureEJBProxy_jndiName"
          value="com/ibm/btt/formatter/server/CHAFormatterServiceServiceHome"/>
   <field id="PureEJBProxy_providerURL" value="iiop://localhost:2809/"/>
</kColl>
```

3. Press Ctrl+S to save it.

8.3.7 Creating a Single Action EJB

This section describes the creation of Single Action EJB as the withdrawal business operation. In the toolkit's architecture, the Single Action EJB is a little different from pure EJB in J2EE. The Single Action EJB's main function is like an Decorator or an Adapter. Single Action EJB is used to wrap the operation provided by the back-end host. It can include journaling or message transformation services as well. Thus, Single Action EJB *should not* handle data persistence as per the toolkit design. The table and journal services are better candidates for this purpose.

Client operation, invoker, and server SAE strategy

This section looks at how invokers and server Single Action EJB work with client operation.

In a client/server environment, a business operation is composed of a flow being executed in three phases, that is, a client operation, an Invoker, and a Single Action EJB, as shown in Figure 8-56.

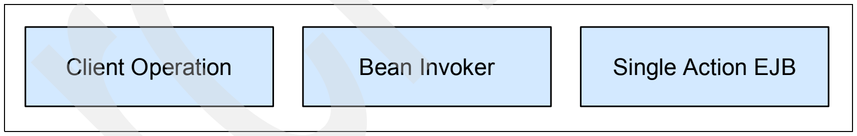


Figure 8-56 Components joined in the client/server communication

The client operation must know the name of the corresponding Invoker. This should be stored as an attribute of client operation, as shown in Figure 8-57.

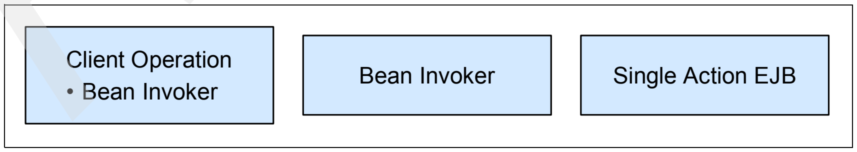


Figure 8-57 Client 0peration should know information about the Bean Invoker

Each operation needs a set of data. Data elements are stored as collections inside the context hierarchy. By simply passing the operation and the name of a context, the operation can access the data of the entire context hierarchy. See Figure 8-58.

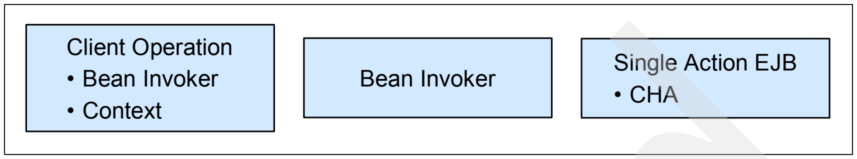


Figure 8-58 Client operation and SAE have knowledge of underlying contexts

You should now let both ends communicate with each other. The Branch Transformation Toolkit provides the Client/Server Communication Service for this purpose. The Client/Server Communication Service requires two specific message formats to work correctly. To send a request message from Client Operation, the csRequestFormat will be used. Likewise, it uses csReplyFormat to reply back to the client.

Thus, the scenario of the overall operation can be as follows:

Using the Client/Server Communication service, the client will prepare and send a request message to the server for processing. The server will process the operation, produce a reply, and send that reply back to the client. See Figure 8-59.

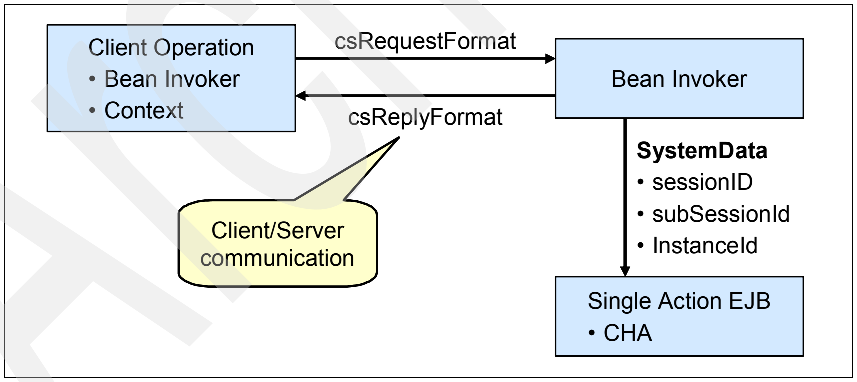


Figure 8-59 The scenario of client/server communication service

The Client/Server Communication service relies on three key configuration files to work correctly, that is, client operation definition, server operation definition, and bean invoker registry mapper.

- Client operation definition

  There are two types of clients, a Java client and a Web client. For Java client, the client operation definition typically resides in dseoper.xml. Use Example 8-6 to define a client operation.

Example 8-6 Client operation definition for Java client

```
<ClientOperation id="clientOp" serverOperation="serverOp"
                 context="clientCtx">
   <refFormat name="csRequestFormat" refId="clientRequestFmt"/>
</ClientOperation>
```

  For a Web client, the client operation definition resides in the configuration file of a Struts Module in bold font, as shown in Example 8-7.

Example 8-7 Client operation definition for Web client

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<struts-config>
   <data-sources />
   <form-beans>
     <form-bean name="signInForm"
                 type="btt.bank.ui.struts.forms.SignInForm"/>
   </form-beans>
   <global-exceptions />
   <global-forwards />
   <action-mappings>
      <action name="signInForm"
          path="/signIn"
          className="com.ibm.btt.struts.config.BTTEJBActionMapping"
          type="com.ibm.btt.struts.actions.EJBSignInAction"
          input="/signin.jsp"
          invokerId="signInInvoker"
          validator="btt.bank.ui.struts.forms.SignInXVal"
          validate="true"
          parameter="signIn">
         <forward name="success" contextRelative="true"
            path="/btt/bank/ui/struts/withdrawal/prepareWithdrawal.do"/>
         <forward name="signin" path="/signin.jsp"/>
      </action>
   </action-mappings>
   <flowcontext contextName="signInCtx" local="true" />
   <plug-in className="com.ibm.btt.struts.plugins.BTTDefaultNotifier" />
   <finals>
     <final type="forward"
```

```
            name="/btt/bank/ui/struts/withdrawal/prepareWithdrawal.do"/>
   </finals>
</struts-config>
```

- Server operation definition

  The server operation definition is used by invokers to invoke server side Single Action EJBs/Business Processes. In our sample, we named this file `serverOp.properties` and placed it in the serverOp.invoker.java package. The sample content of the file will be as shown in Example 8-8.

Example 8-8 Server operation definition

```
implClass=serverOp.invoker.java.ServerOpInvoker
jndiName=ejb/server/ServerOpHome
factory=com.ibm.websphere.naming.WsnInitialContextFactory
location=iiop://localhost:2809
homeClassName=server.ServerOpHome
isLocal=false
csReplyFormat=clientReplyFormat
```

- Bean Invoker Registry Mapper

  You now have the client and server operation definition. The Bean Invoker Registry Mapper defines the mapping between serverOperation/invokerId and the server operation definition file. The sample could be as follows:

  ```
  serverOp=serverOp.invoker.java.serverOp:RB
  ```

  The BeanInvokerRegistry.properties resides in the com.ibm.btt.cs.invoker.base package.

Because it has the serverOperation/invokerId attribute, the Client/Server Communication service will know which corresponding invoker to instantiate and run. Further, because each client has a unique session ID, the service will know to which client the server's reply will be passed.

The client/server mechanism is as follows:

1. In the client, use the format identified as csRequestFormat to format the request data.
2. In the server, the client/server communication mechanism refers to the file com.ibm.btt.cs.invoker.base.BeanInvokerRegistryMapper.properties. Use the key identified by the client operation's serverOperation attribute or invokerId to find the corresponding properties file and then instantiate the invoker defined in the properties file.
3. The mechanism then uses the invoker’s parseRequestData method to unformat the request data for the Single Action EJB. The corresponding Single Action EJB should have the method to accept the request data.

4. Trigger the invoker to execute the Single Action EJB.
5. In the server, it uses the invoker's processRespondData method to format the reply data.
6. The client unformats the reply data using the csReplyFormat and saves the data to the client operation context.

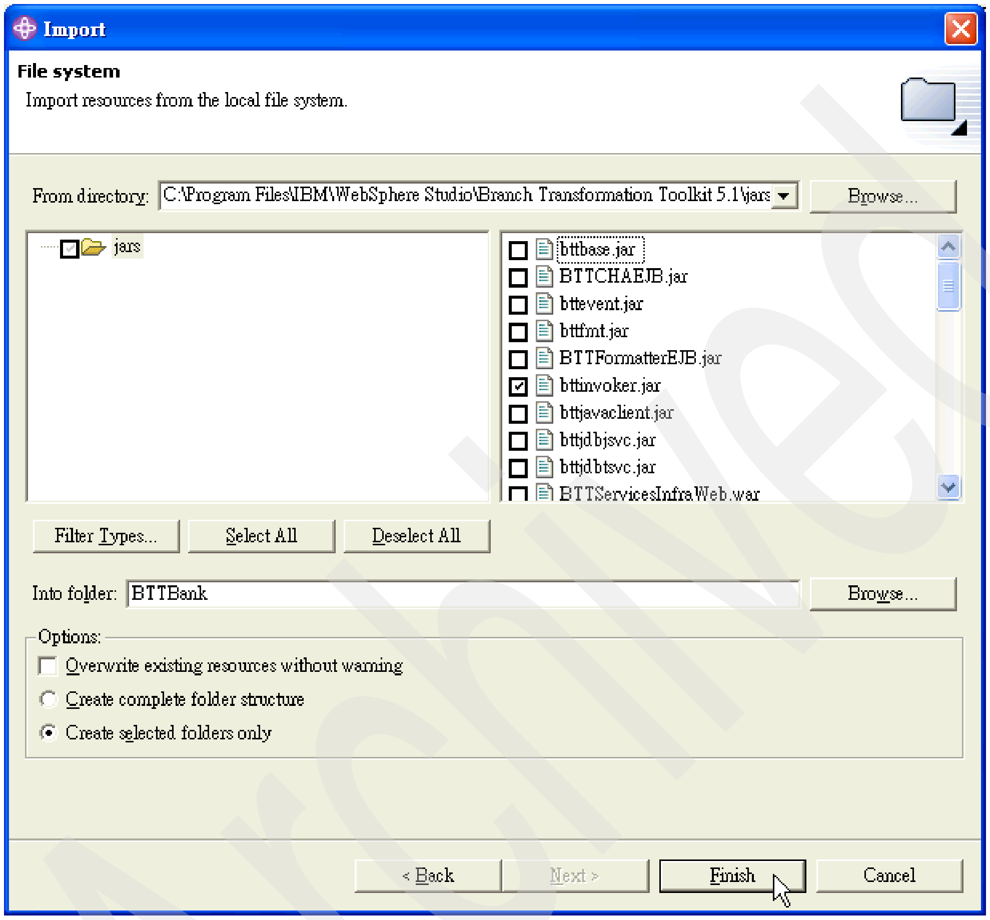
Creating the withdrawal Single Action EJB

This section describes how to build the withdrawal Single Action EJB as one of the business logic in our application. It is elementary, with little functionality. It simply returns hard-coded data in a reply to the client. The main purpose is to test the overall flow in the client/server environment.

To create a Single Action EJB, follow these steps:

1. Open WebSphere Studio Application Developer Integration Edition. In J2EE perspective, select the **BTTBank** project in Project Navigator view. Right-click it and select **Import** from the context menu.
2. In the Import pop-up window, select **File system** as the import source, and then click **Next**.

3. In the next page, click **Browse** and navigate to **<BTT_install_dir>/jars**. Click **OK**. Select **bttinvoker.jar**, and then click **Finish**, as shown in Figure 8-60.



From directory: C:\Program Files\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1\jars

- [✓] jars

- [ ] bttbase.jar
- [ ] BTTCHAEJB.jar
- [ ] bttevent.jar
- [ ] bttfmt.jar
- [ ] BTTFormatterEJB.jar
- [✓] bttinvoker.jar
- [ ] bttjavaclient.jar
- [ ] bttjdbjsvc.jar
- [ ] bttjdbtsvc.jar
- [ ] BTTServicesInfraWeb.war

Into folder: BTTBank

Options:

- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

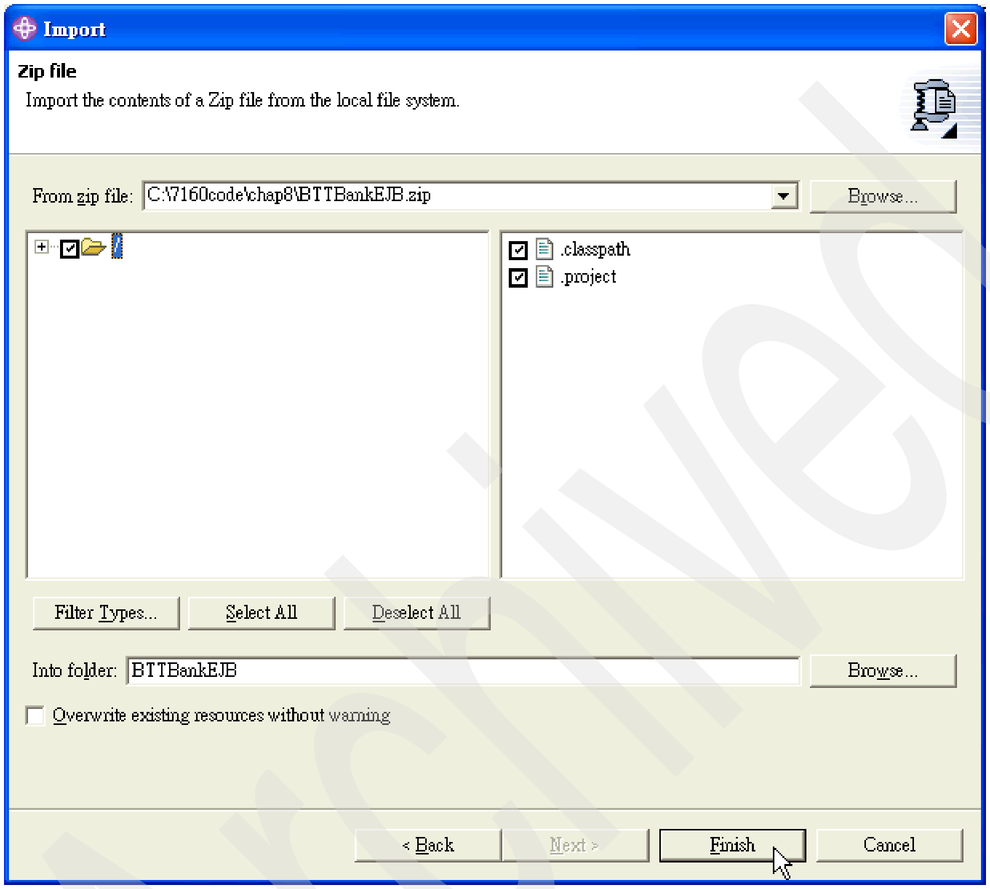
Figure 8-60 Import bttinvoker.jar into the BTTBank project

Importing the prebuilt project

The sample Java code used in this chapter is provided in the \7160code\chap8 directory of our redbook sample code. Refer to Appendix C, “Additional material” on page 505 for instructions on how to install the sample code in your computer. To import the prebuilt project, perform the following tasks:

1. Right-click the **BTTBankEJB** project. Select **Import** from the context menu. Use zip file as the import source, and then click **Next**.

2. In the next window, browse to the **c:\7160code\chap8** folder and select **BTTBankEJB.zip**. Click **Finish**, as shown in Figure 8-61.


Import
Zip file
Import the contents of a Zip file from the local file system.

From zip file: C:\7160code\chap8\BTTBankEJB.zip

- ✓ .classpath
- ✓ .project

Into folder: BTTBankEJB

- [ ] Overwrite existing resources without warning

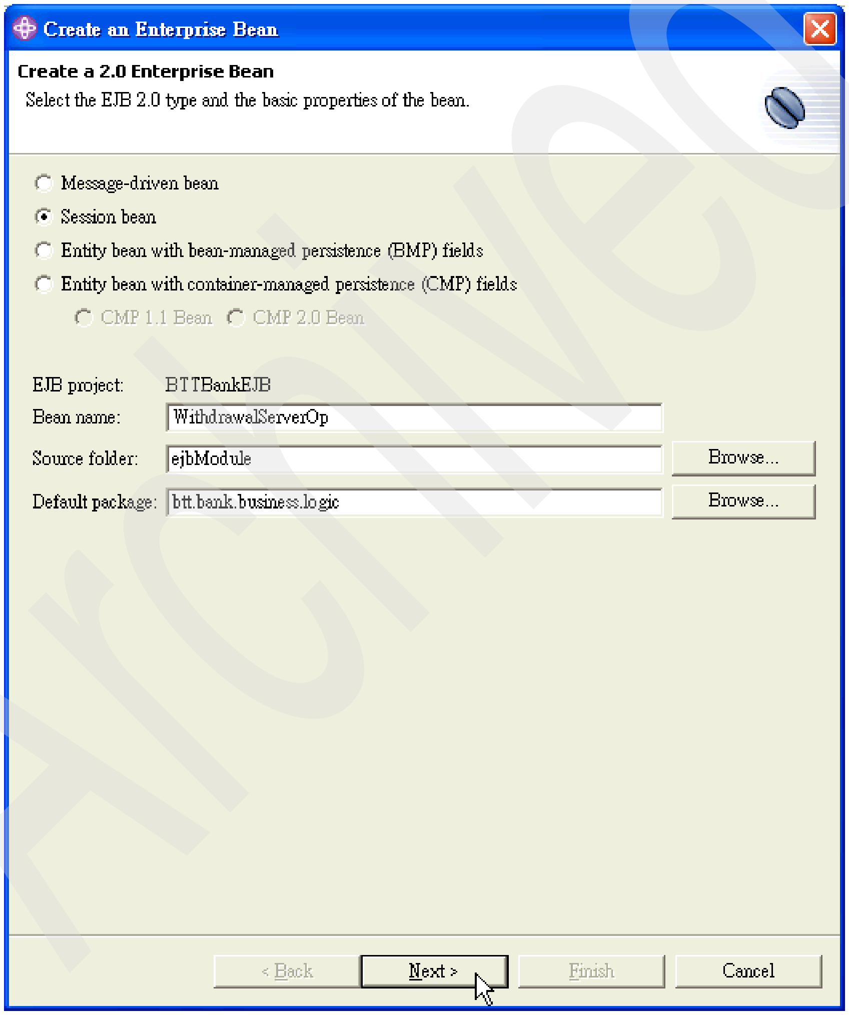
Figure 8-61 Import the prebuilt project

3. You can see a pop-up window asking you to confirm the overwrite. Click **Yes To All**.
4. Select **Project** → **Rebuild Project** from the main menu to fix the errors.

Creating the WithdrawalServerOp session bean

1. In the J2EE perspective, switch to the **Project Navigator** view. Open the **EJB Deployment Descriptor** in the BTTBankEJB project.
2. Select the **Beans** tab and click **Add** to add an EJB.

3. Enter the following properties to create the WithdrawalServerOp session bean, as shown in Figure 8-62:
   - Bean Type: `Session Bean`
   - EJB project: `BTTBankEJB`
   - Bean name: `WithdrawalServerOp`
   - Source folder: `ejbModule`
   - Default package: `btt.bank.business.logic`
4. Click **Next**. See Figure 8-62


Create an Enterprise Bean

Create a 2.0 Enterprise Bean

Select the EJB 2.0 type and the basic properties of the bean.

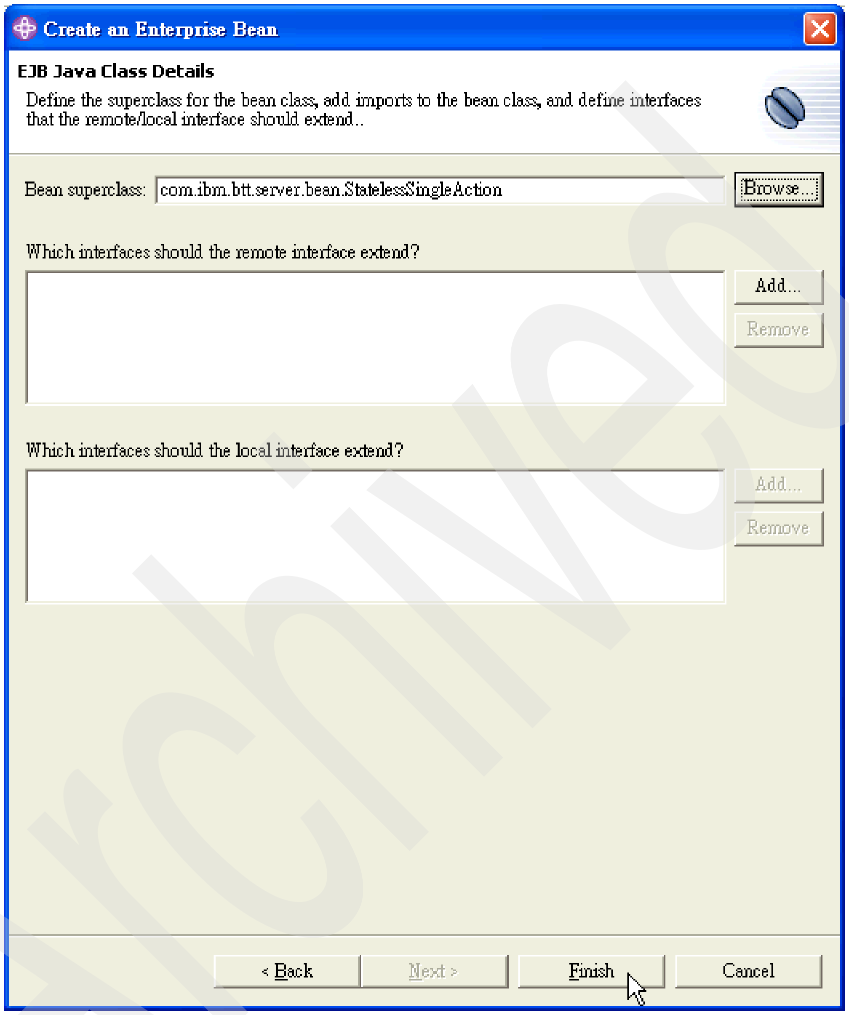
( ) Message-driven bean
(•) Session bean
( ) Entity bean with bean-managed persistence (BMP) fields
( ) Entity bean with container-managed persistence (CMP) fields
( ) CMP 1.1 Bean ( ) CMP 2.0 Bean

EJB project: BTTBankEJB
Bean name: WithdrawalServerOp
Source folder: ejbModule
Default package: btt.bank.business.logic

Figure 8-62 Properties for the WithdrawalServerOp EJB

5. In the Enterprise Bean Details page, accept the default values and click **Next**.

6. In the next dialog box, use **com.ibm.btt.server.bean.StatelessSingleAction** as the superclass of this EJB, as shown in Figure 8-63, and click **Finish**.



Define the superclass for the bean class, add imports to the bean class, and define interfaces that the remote/local interface should extend..

Bean superclass: com.ibm.btt.server.bean.StatelessSingleAction

Which interfaces should the remote interface extend?

Which interfaces should the local interface extend?

Figure 8-63 Use com.ibm.btt.server.bean.StatelessSingleAction as the superclass

Configuring WithdrawalServerOp session bean

1. In the EJB JAR Descriptor Editor, click the **Beans** tab, select **WithdrawalServerOp** from the EJB list and add the items shown in Table 8-3 to the Environment Variables section:

Table 8-3 Environment variables for WithdrawalServerOp session bean

Name	Description	Type	Value
ID		String	WithdrawalServerOp
sessionCtxName		String	javaSessionCtx
dseIniPath		String	c:\\dse\\dse.ini
context		String	withdrawalServerCtx
contextMode		String	remote

2. Save and close the **EJB JAR Descriptor Editor**.

Modifying source code

To modify the stateless session bean WithdrawalServerOpBean, perform the following tasks:

1. In the EJB Projects BTTBankEJB, expand the folder **ejbModule/btt.bank.business.logic**.
2. Double-click **WithdrawalServerOpBean.java**.
3. Add the import statements from Example 8-9 to the existing import statements.

Example 8-9 Additional import statements for WithdrawalServerOpBean.java

```
import java.util.Date;
import java.util.Hashtable;
import com.ibm.btt.base.BTTSystemData;
import com.ibm.btt.base.Context;
import com.ibm.btt.formatter.client.FormatElement;
import com.ibm.btt.server.bean.BTTSAEException;
```

4. Change the ejbCreate and ejbRemove methods to look as shown in bold in Example 8-10.

Example 8-10 Modified ejbCreate and ejbRemove methods

```
/**
 * ejbCreate
 */
public void ejbCreate() throws BTTSAEException {
   super.ejbCreate();
}

/**
 * ejbRemove
 */
public void ejbRemove() {
   super.ejbRemove();
}
```

5. Implement the execute method as shown in Example 8-11. The execute method is divided into three main sections:
   - The first section deals with context hierarchy.
   - The second section sees the update of the context data with the request data.
   - The third section provides hard-coded data as the response to this operation.
   - The final part is the error handler.

Example 8-11 Execute method for the WithdrawalServerOpBean

```
public Hashtable execute(BTTSystemData sysData, Hashtable reqData)
             throws Exception {
   Hashtable result = null;
   try {
      // initialize context hierarchy
      initialize(sysData);
      Context SAEContext = getContext();
      if (SAEContext.getParent() == null) {
         Context parent = Context.getContextByInstanceID(getInstanceId());
         SAEContext.chainTo(parent);
      }
      // set request data to withdrawalServerCtx
      SAEContext.setValueAt("BranchId", (String)reqData.get("BranchId"));
      SAEContext.setValueAt("AccountNumber",
              (String)reqData.get("AccountNumber"));
      SAEContext.setValueAt("Date", (Date)reqData.get("Date"));
      SAEContext.setValueAt("Amount", (Float)reqData.get("Amount"));
      System.out.println("withdrawalServerOp SAE context:\n" +
             SAEContext.getKeyedCollection());
      // set hard-coded response data to withdrawalServerCtx
      SAEContext.setValueAt("TrxReplyCode", "00");
      SAEContext.setValueAt("AccountBalance", "10000");
      SAEContext.setValueAt("TrxErrorMessage", "withdrawalOK");
      result = ((FormatElement)getFormat("afterRecJournalFmt"))
        .formatHashtable(SAEContext);
   }
   catch (Exception e) {
      e.printStackTrace();
      throw e;
   }
   return result;
}
```

6. Save your changes and close the **Source Editor**.

To modify the EJB home interface WithdrawalServerOpHome, perform the following tasks:

1. Double-click **WithdrawalServerOpHome.java**.
2. Change the Home interface to that shown in Example 8-12.

Example 8-12 Modified WithdrawalServerOpHome interface

```
package btt.bank.business.logic;

import com.ibm.btt.server.bean.BTTSAEException;

/**
 * Home interface for Enterprise Bean: WithdrawalServerOp
 */
public interface WithdrawalServerOpHome extends javax.ejb.EJBHome {
   /**
    * Creates a default instance of Session Bean: WithdrawalServerOp
    */
   public btt.bank.business.logic.WithdrawalServerOp create()
    throws javax.ejb.CreateException, java.rmi.RemoteException,
      BTTSAEException;
}
```

3. Save your changes and close the source editor.

To modify the EJB remote interface WithdrawalServerOp, perform the following tasks:

1. Double-click **WithdrawalServerOp.java**.
2. Change the remote interface as shown in Example 8-13.

Example 8-13 Modified WithdrawalServerOp interface

```
package btt.bank.business.logic;

import java.rmi.RemoteException;
import java.util.Hashtable;
import com.ibm.btt.base.BTTSystemData;
import com.ibm.btt.server.bean.BTTSAEException;
```

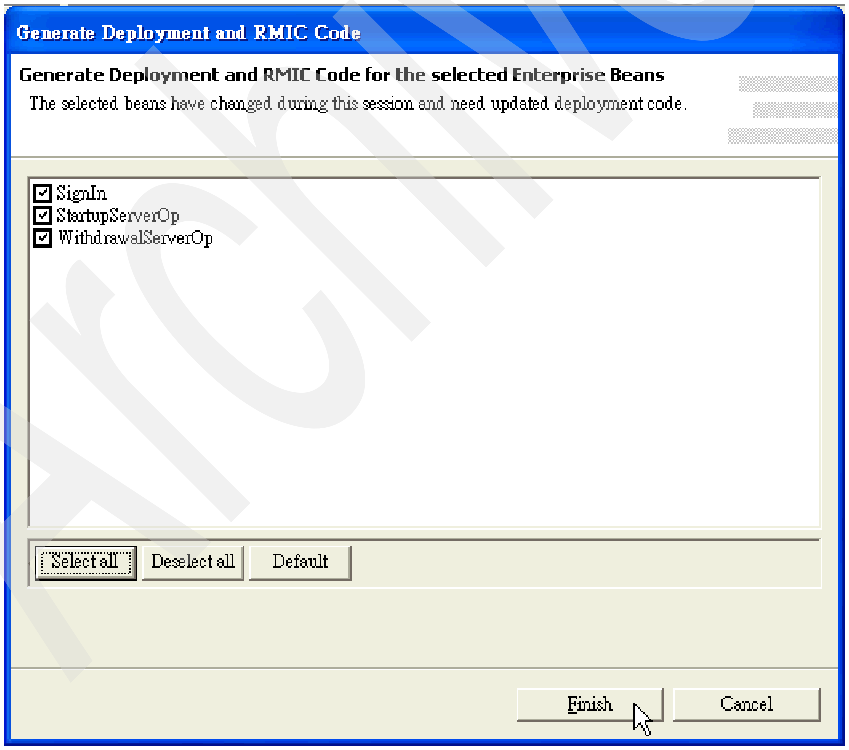
```
/**
 * Remote interface for Enterprise Bean: WithdrawalServerOp
 */
public interface WithdrawalServerOp extends javax.ejb.EJBObject {

   public Hashtable execute(BTTSystemData sysData, Hashtable reqData)
                       throws Exception;
}
```

3. Press **Ctrl+S** to save the changes.

Generating Deployment and RMIC Code

To generate the deplyment and RMIC code, follow these steps:

1. Right-click the **BTTBankEJB** project and select **Generate** → **Deployment and RMIC Code.**
2. In the pop-up dialog box, click **Select all** and then **Finish**, as shown in Figure 8-64.



Generate Deployment and RMIC Code

Generate Deployment and RMIC Code for the selected Enterprise Beans

The selected beans have changed during this session and need updated deployment code.

☑ SignIn
☑ StartupServerOp
☑ WithdrawalServerOp

Figure 8-64 Generate Deployment and RMIC Code for the BTTBankEJB project

8.3.8 Developing the Web facade with Struts Tools BTT Extensions

Struts is a Model-View-Controller implementation that uses servlets and JavaServer Pages (JSP) technology. It is a set of cooperating classes, servlets, and JSP tags that make up a reusable MVC 2 design. Struts also contains an extensive tag library and utility classes that work independently of the framework.

A Struts configuration file is an XML document that describes all or part of a Struts application. Because the toolkit Struts Extensions component provides customization to the Apache Struts Framework, some settings in extended Struts configuration files are hidden from the Struts Configuration File Editor provided by WebSphere Studio Application Developer, which is a standard Struts configuration file editor.

The Struts Tools BTT Extensions is a WebSphere Studio Application Developer plug-in with which you can modify Struts configuration files to include specific settings for the toolkit's Struts Extensions. The Struts Tools BTT Extensions has a friendly user interface that saves you from editing the XML source of Struts configuration files directly.

This section describes how to import a prebuilt project that includes some skeleton code to simplify the development effort. It then takes you through a typical Struts development. You can use the features provided by Struts Tools BTT Extensions to configure the settings. Thus, the BTT Struts component and the invoker can work together to access the logic/flow provided by the business layer.

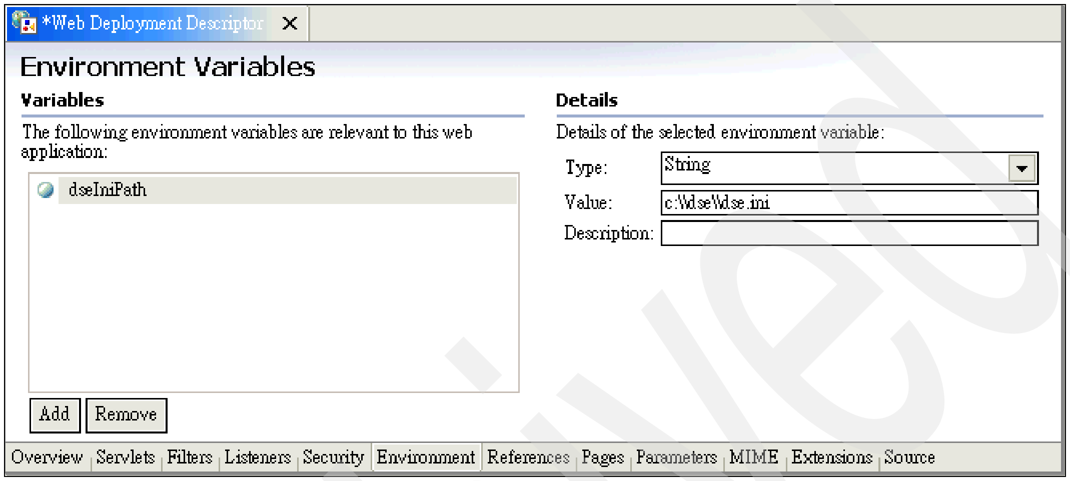
Importing the prebuilt project

To import the prebuilt project that includes some skeleton code into the BTTBankWeb project, following these instructions:

1. Right-click the **BTTBankWeb** project. Select **Import** from the context menu. Use zip file as the import source, and click **Next**.
2. In the next window, browse to the **c:\7160code\chap8** folder and select **BTTBankWeb.zip**. Click **Finish**.
3. You will see a pop-up window asking for confirmation to overwrite. Click **Yes To All**.
4. Right-click the **BTTBankWeb** project, and select **Rebuild Project** to fix some warnings.
5. Navigate to the **BTTBankWeb/WebContent/WEB-INF** folder, and double-click **web.xml** to open it.

6. Click the **Environment** tab, and then **Add** to add a variable named `dseIniPath`. Enter the following values as it properties, as shown in Figure 8-65:

   – Type: `String`
   – Value: `c:\\dse\\dse.ini`


*Web Deployment Descriptor

Environment Variables

Variables

The following environment variables are relevant to this web application:

dseIniPath

Details

Details of the selected environment variable:

Type: String

Value: c:\\dse\\dse.ini

Description:

Figure 8-65 Add the dseIniPath variable

7. Press Ctrl+S to save your work.

Creating Struts module for withdrawal operation

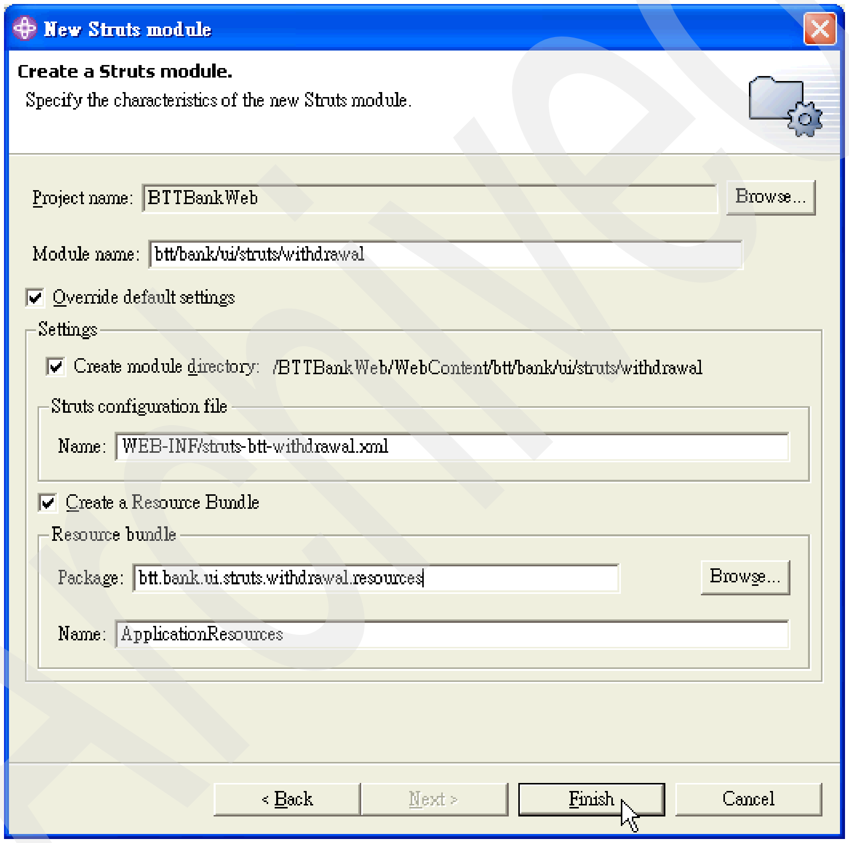
To create the Struts module, follow these steps:

1. Select **File** → **New** → **Other** from the main menu.
2. Select **Web** → **Struts** in the left panel of the New dialog box.
3. Select **Struts Module** in the right panel and click **Next**.

4. Enter the following values for the Struts module, as shown in Figure 8-66, and click **Finish**:

   – Project name: `BTTBankWeb`
   – Module name: `btt/bank/ui/struts/withdrawal`
   – Struts configuration file: `WEB-INF/struts-btt-withdrawal.xml`
   – Package: `btt.bank.ui.struts.withdrawal.resources`

Note: Check the **Override default settings** option to change the values of both the **Struts configuration file** and the **Package**.


New Struts module

Create a Struts module.

Specify the characteristics of the new Struts module.

Project name: BTTBankWeb

Module name: btt/bank/ui/struts/withdrawal

[✓] Override default settings

Settings

[✓] Create module directory: /BTTBankWeb/WebContent/btt/bank/ui/struts/withdrawal

Struts configuration file

Name: WEB-INF/struts-btt-withdrawal.xml

[✓] Create a Resource Bundle

Resource bundle

Package: btt.bank.ui.struts.withdrawal.resources

Name: ApplicationResources

Figure 8-66 Create the btt/bank/ui/struts/withdrawal module

Designing the Struts Web application using top-down approach

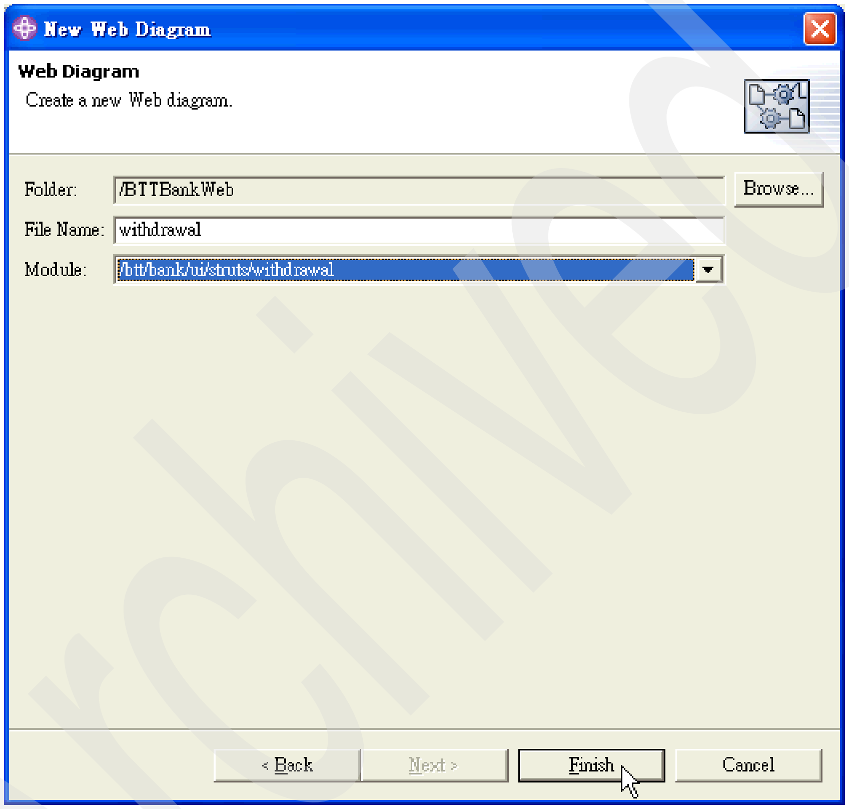
In this section, use Web diagram to lay out the design of the Web application. A Web diagram is a file that helps you visualize the application flow of a Struts-based Web application. As a result of the levels of indirection involved in a Struts application, being able to visually see the application's flow can help you understand the application better.

Creating a Web diagram for newly created Struts module

To create a Web diagram, perform the following tasks:

1. Select **File** → **New** → **Other** from the main menu.
2. Select **Web** → **Struts** in the left panel of the New dialog.
3. Select **Web Diagram** in the right panel and click **Next**.

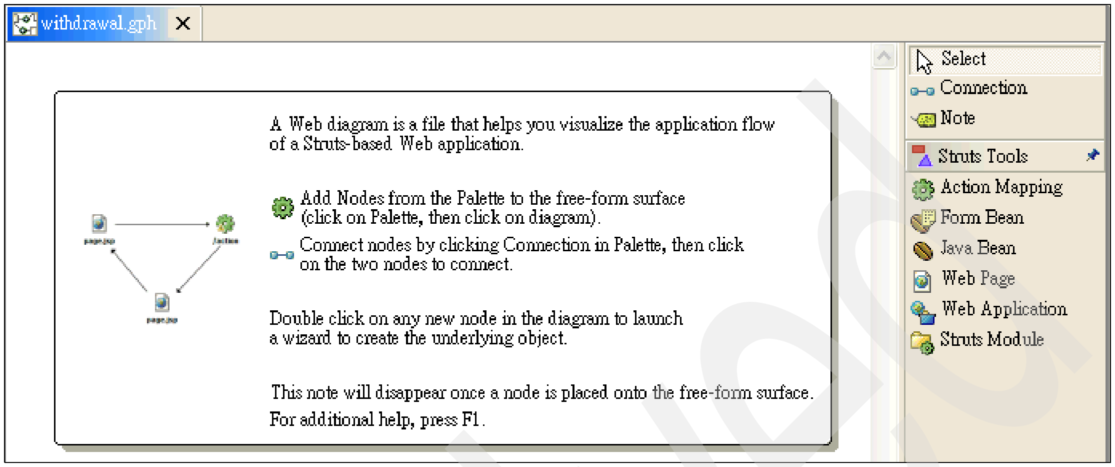
4. Enter the following values for the Web diagram, as shown in Figure 8-67, and click **Finish**:
   - Folder: `/BTTBankWeb`
   - File Name: `withdrawal`
   - Module: `/btt/bank/ui/struts/withdrawal`



Folder: /BTTBankWeb
File Name: withdrawal
Module: /btt/bank/ui/struts/withdrawal

Figure 8-67   Create a new Web diagram

The Web diagram editor should be open, as shown in Figure 8-67. This figure contains a palette for Web diagram editor. It contains Select, Connection, Note icons, and Struts tools drawer with Action Mapping, Form Bean, Java Bean, Web Page, Web Application, and Struts Module icons.



A Web diagram is a file that helps you visualize the application flow of a Struts-based Web application.

Add Nodes from the Palette to the free-form surface (click on Palette, then click on diagram).

Connect nodes by clicking Connection in Palette, then click on the two nodes to connect.

Double click on any new node in the diagram to launch a wizard to create the underlying object.

This note will disappear once a node is placed onto the free-form surface.

For additional help, press F1.

Figure 8-68 Web diagram with the struts drawer open in the palette

Adding the Web components

Use the icons in the palette to add two actions, two Web pages, and a form bean on the empty surface:

1. When you drop an action, you can change its name to the name shown in Figure 8-69.

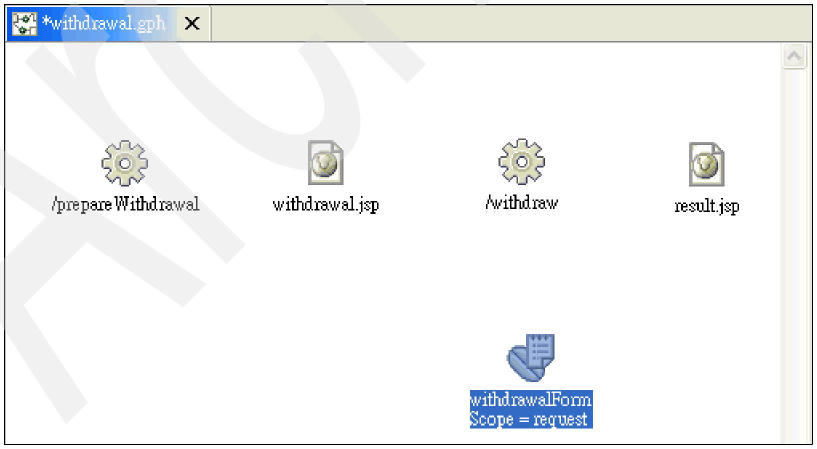


Figure 8-69 Web diagram with initial components

2. When you drop a JSP page, you should change its name to the name shown not only in the figure, but prepend the module name as well. For example, use `/btt/bank/ui/struts/withdrawal/withdrawal.jsp` instead of `withdrawal.jsp`.
3. When you drop a form bean, you are prompted for a name and scope. Enter `withdrawalForm` as name and select **request** as scope.

Components in gray are not yet implemented, meaning they are only available in the Web diagram and not as an underlying file such as a Java class or JSP.

Note: You should add /prepareWithdrawal action because you are using Struts modules. If you do not have these prepare actions, and you go to the JSP pages directly, the subsequent submit action /withdraw cannot be found by the Struts framework. The root cause is that the Struts framework would not have initialized the struts-btt-withdrawal.xml module. Thus, before going to withdrawal.jsp by selecting the /prepareWithdrawal action, Struts framework can initialize the withdrawal Struts module. Then, the submission from withdrawal.jsp can correctly direct to the /withdraw action in the struts-btt-withdrawal.xml module.

Designing the application flow

When the components are laid out, connect them to define the flow of the application. Figure 8-70 on page 327 shows the layout with connections.

1. Create a connection from withdrawal.jsp to /withdraw action.

   This should be done to indicate the action that will be invoked when the form is submitted.

   Select **Connection** from the palette, click **withdrawal.jsp**, and drag the connection to the /withdraw action.
2. Create a local forward back to withdrawal.jsp.

   This will be used to forward the user back to the log in page when business exceptions occur in the log in action.

   a. Select **Connection**, click the **/withdraw** action, and drag it back to withdrawal.jsp.

   b. Rename the forward to `failure` and press **Enter**.
3. Create a local forward to the result.jsp.

   a. Select **Connection**, click the **/withdraw** action, and drag it to result.jsp.

   b. Rename the forward to `success` and press **Enter**.

      You will see a dotted arrow line from the /withdraw action to the result.jsp page, as shown in Figure 8-70.

4. Associate the /withdraw action with the withdrawalForm form bean.

   a. To create a connection, select the **Connection** icon from the Palette.

   b. Single click the **/withdraw** action, and drag and drop the connection into the withdrawalForm.

      You will see a dotted arrow line from /withdraw action to the withdrawalForm form bean as shown in Figure 8-70.

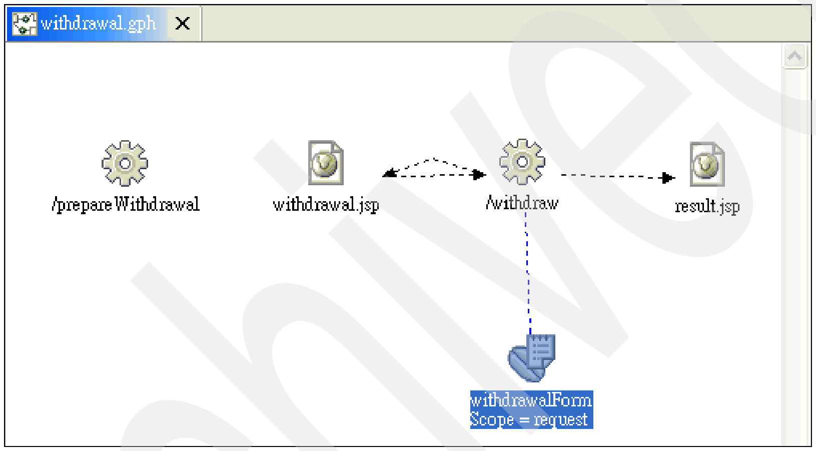


Figure 8-70 Web diagram with components connected

5. Save the Web diagram by pressing Ctrl+S.

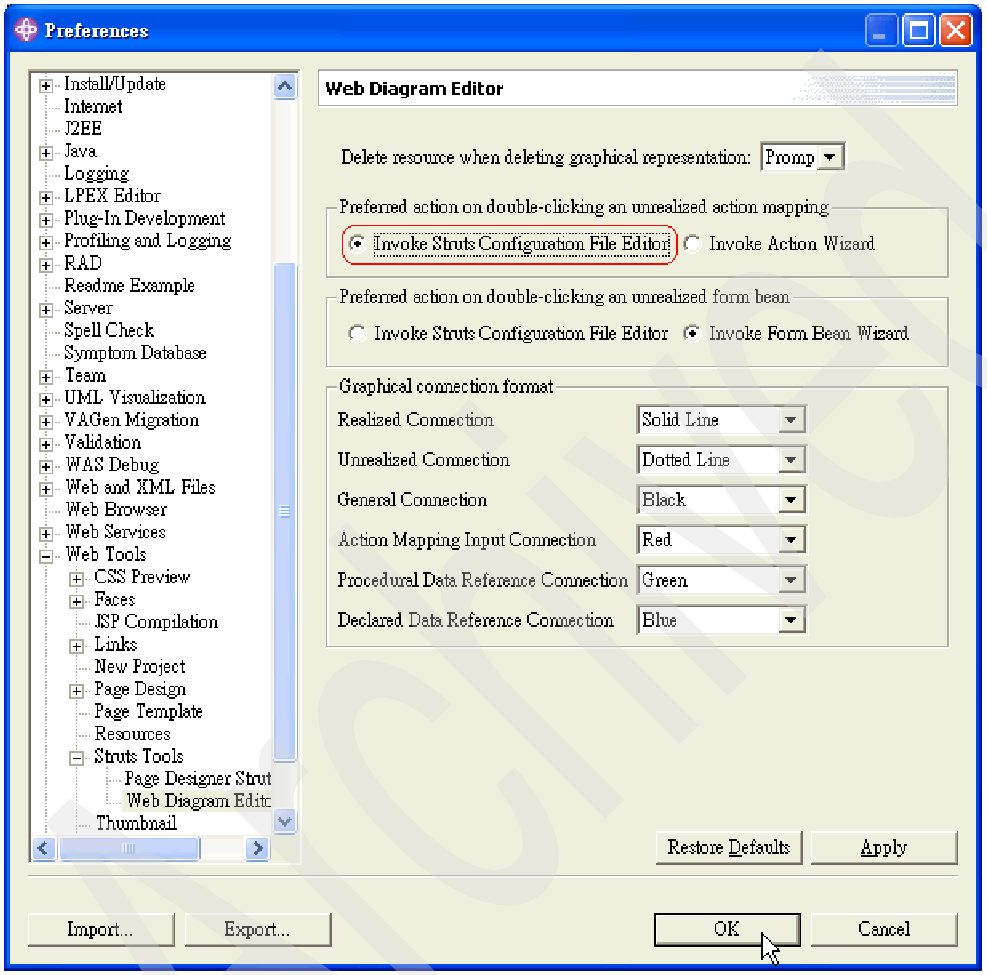
Implementing the Struts Web diagram

Branch Transformation Toolkit Struts Extension includes BTTEJBActionMapping and EJBAction Java classes that handle the connectivity between Struts Action Mappings and Branch Transformation Toolkit Invokers. We use the built-in EJBAction.

Thus, before actually starting, change the way the WebSphere Studio Application Developer Integration Edition v5.1.1 realizes an action mapping, when double-clicking the icon. Follow these instructions to make the change:

1. Select **Window** → **Preferences** from the main menu bar.
2. Expand **Web Tools** → **Struts Tools**, and then select **Web Diagram Editor** in the left panel of the Preferences window.

3. Select the **Invoke Struts Configuration File Editor** option as the Preferred action used when double-clicking an unrealized action mapping as shown in Figure 8-71.


Preferences

Web Diagram Editor

Delete resource when deleting graphical representation: Promp

Preferred action on double-clicking an unrealized action mapping
[✓ Invoke Struts Configuration File Editor] Invoke Action Wizard

Preferred action on double-clicking an unrealized form bean
Invoke Struts Configuration File Editor [✓ Invoke Form Bean Wizard]

Graphical connection format

Realized Connection	Solid Line
Unrealized Connection	Dotted Line
General Connection	Black
Action Mapping Input Connection	Red
Procedural Data Reference Connection	Green
Declared Data Reference Connection	Blue

Figure 8-71 Use Struts Configuration File Editor to realize action mappings

After this change, you should consider the order of the component realization. When the Web diagram is laid out, start implementing the components. This can be done in different orders, and the support you get from the Struts tools depends on the order you choose:

- You can implement the form beans first. When you later implement the JSPs, you can choose which fields from the form beans that should be present on the pages.

- You can implement the JSPs first. When you later implement the form beans , you can choose which fields from the JSP pages should be added to the form beans as properties.

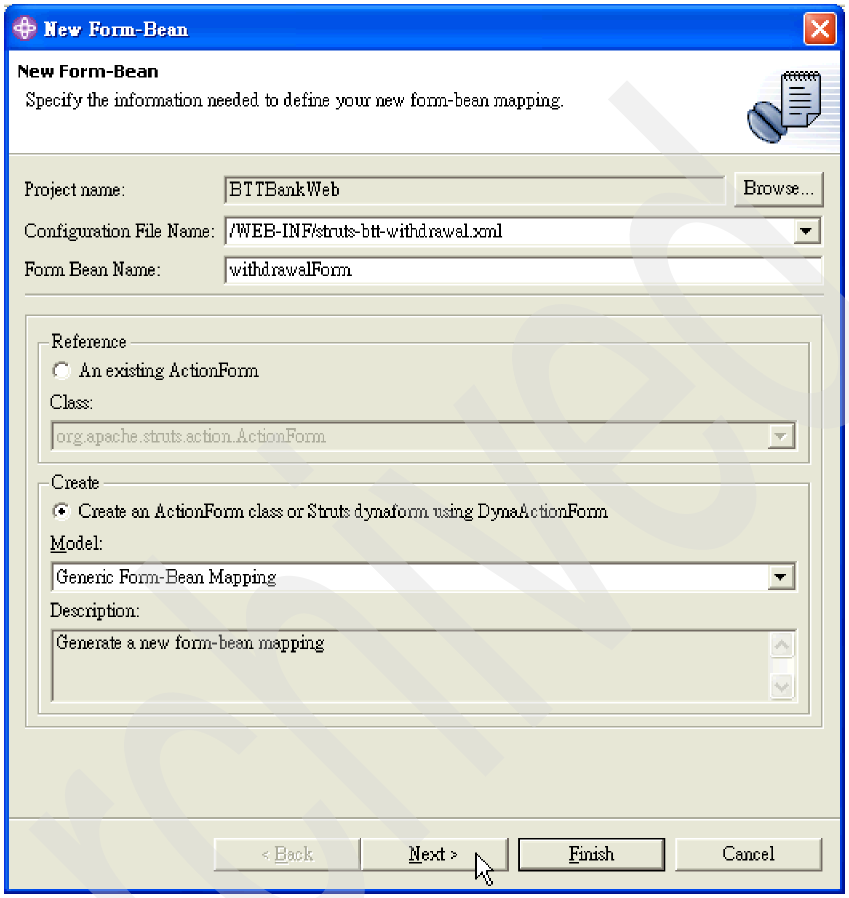
In our sample, we chose to implement the form beans first to have full control over their contents and structure.

Developing a form bean

Implement the withdrawalForm form bean by performing the following tasks:

1. Double-click the **withdrawalForm** form bean in the Web diagram.
2. The New Form Bean dialog should be displayed with all the fields populated by default. Ensure that the **Create New ActionForm class or Struts dynamform using DynaActionForm** radio button is selected, and **Generic**

Form-Bean Mapping is selected in the Model field. Accept the default, as shown in Figure 8-72, and click **Next**.



New Form-Bean

New Form-Bean

Specify the information needed to define your new form-bean mapping.

Project name: BTTBankWeb

Configuration File Name: /WEB-INF/struts-btt-withdrawal.xml

Form Bean Name: withdrawalForm

Reference

( ) An existing ActionForm

Class: org.apache.struts.action.ActionForm

Create

(•) Create an ActionForm class or Struts dynaform using DynaActionForm

Model: Generic Form-Bean Mapping

Description: Generate a new form-bean mapping

Figure 8-72 Realize Struts components - Withdrawal Form Bean - Bean information

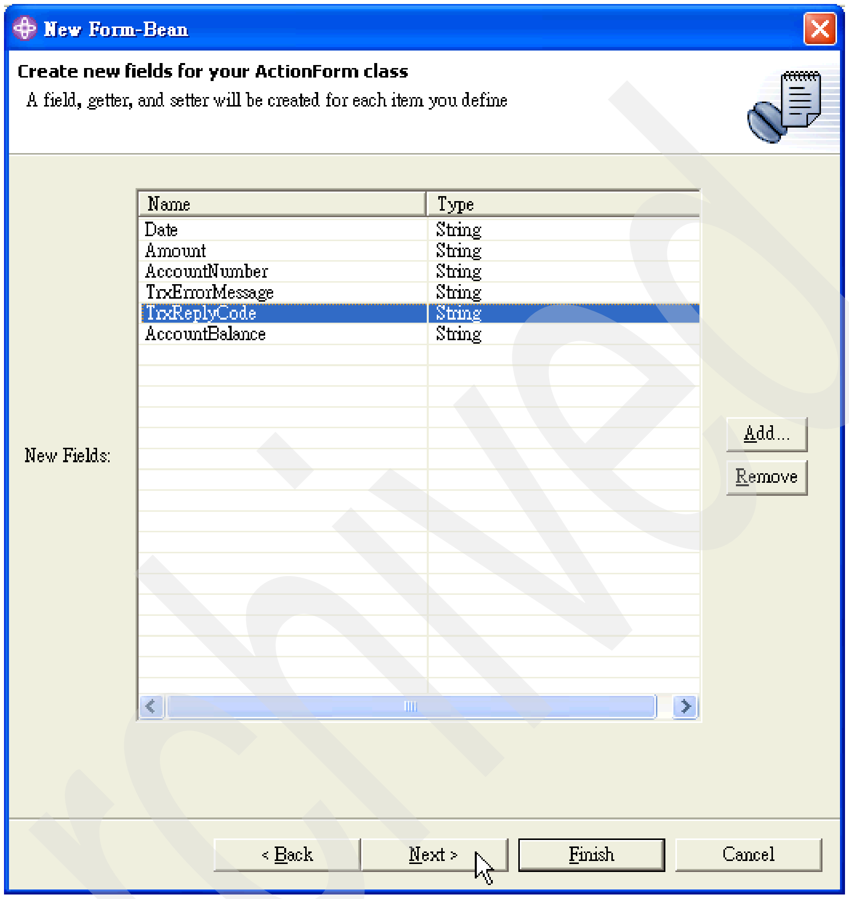
3. When the Choose New Fields dialog box opens, you can choose an existing Form in a HTML/JSP file and add it to the form bean directly. However, because you have not realized the withdrawal.jsp yet, skip this and create the fields directly in the bean. Click **Next**.

4. When the Create New Fields dialog box opens, follow these steps:
   a. Click **Add**.
   b. Enter the fields shown in Table 8-4:

Table 8-4 Fields added to the withdrawalForm

Name	Type
Date	String
AccountNumber	String
Amount	String
TrxErrorMessage	String
AccountBalance	String
TrxReplyCode	String

c. When complete, the dialog box should look as shown in Figure 8-73. Click **Next** to continue.



Create new fields for your ActionForm class

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

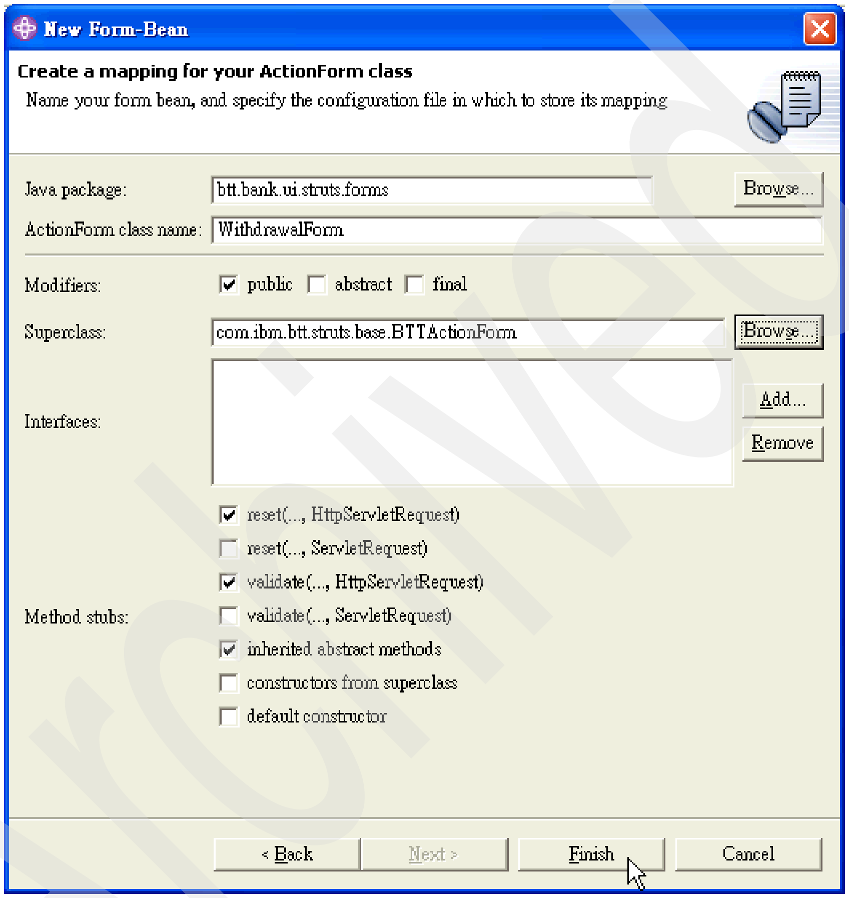
New Fields:

Name	Type
Date	String
Amount	String
AccountNumber	String
TrxErrorMessage	String
TrxReplyCode	String
AccountBalance	String

Figure 8-73 Realize Struts components - Withdrawal Form Bean - Create new fields

5. When the Create Mapping for the ActionForm class dialog box opens, enter the following information, as shown in Figure 8-74, and then click **Finish**:

   – Java package: `btt.bank.ui.struts.forms`
   – ActionForm class name: `WithdrawalForm`
   – Superclass: `com.ibm.btt.struts.base.BTTActionForm`


New Form-Bean

Create a mapping for your ActionForm class

Name your form bean, and specify the configuration file in which to store its mapping

Java package: btt.bank.ui.struts.forms

ActionForm class name: WithdrawalForm

Modifiers: [✓ public] abstract final

Superclass: com.ibm.btt.struts.base.BTTActionForm

Interfaces:

Method stubs:
- [✓ reset(..., HttpServletRequest)]
- reset(..., ServletRequest)
- [✓ validate(..., HttpServletRequest)]
- validate(..., ServletRequest)
- [✓ inherited abstract methods]
- constructors from superclass
- default constructor

Figure 8-74 Create a mapping for ActionForm class

The following actions have been completed by the wizard:

- A class WithdrawalForm has been created in the package btt.bank.ui.struts.forms.

- The Struts configuration file struts-btt-withdrawal.xml has been updated with the form bean information, as shown in Example 8-14.

Example 8-14 Struts configuration file struts-btt-withdrawal.xml snippet

```
<!-- Form Beans -->
<form-beans>
   <form-bean name="withdrawalForm" type="btt.bank.ui.struts.forms.WithdrawalForm">
   </form-bean>
</form-beans>
```

Now that the Web diagram is updated, you can see that the form bean appears in the diagram and that the color has changed. The color change in Figure 8-75 for the withdrawalForm denotes that the Struts component has been realized.

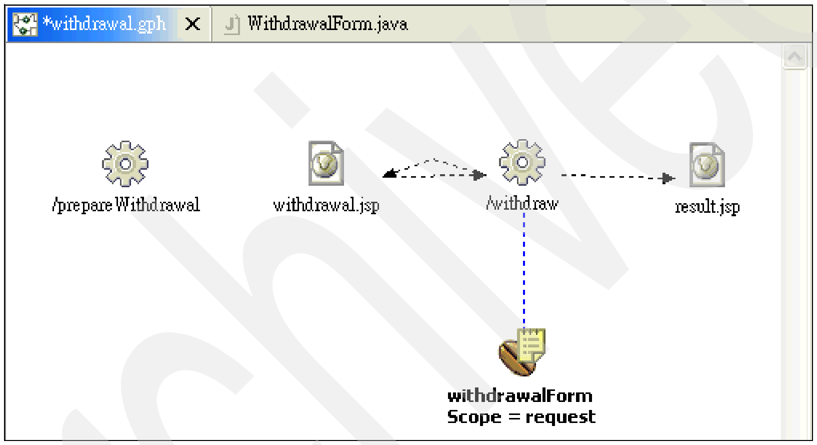


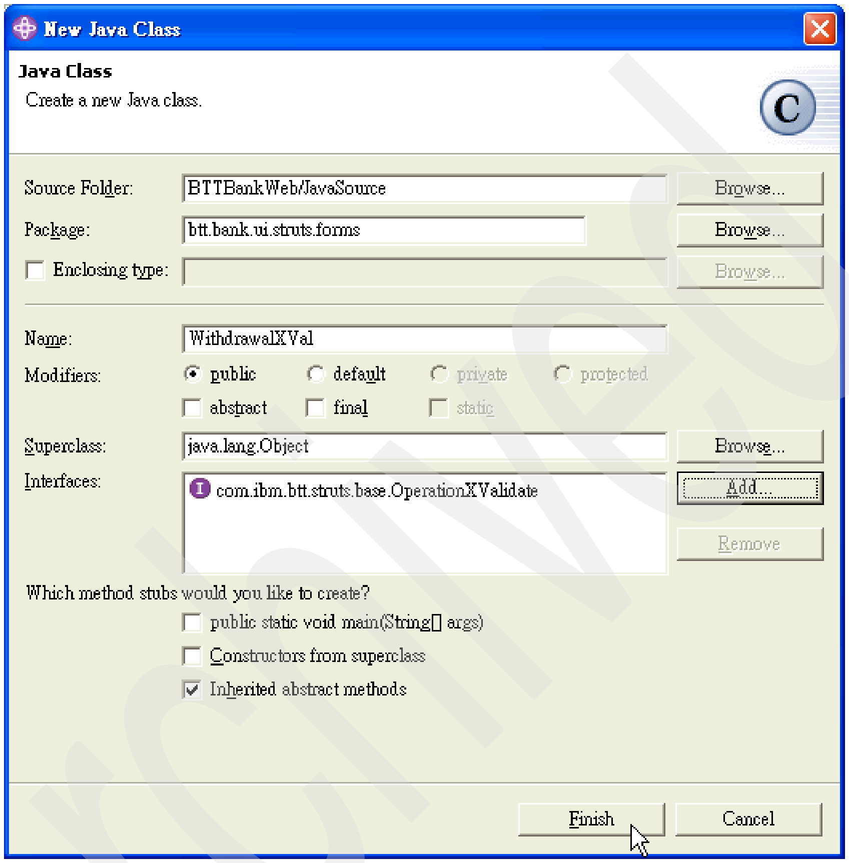
Figure 8-75 Realize Struts components - Withdrawal form bean realized

Creating the corresponding validator

To create the validator that works with BTTActionForm, perform the following tasks:

1. Right-click **btt.bank.ui.struts.forms** package and select **New** → **Class**.

2. In New Java Class dialog box, name the new class `WithdrawalXVal` and add **com.ibm.btt.struts.base.OperationXValidate** to the interfaces as shown in Figure 8-76. Click **Finish**.



Source Folder: BTTBankWeb/JavaSource
Package: btt.bank.ui.struts.forms
☐ Enclosing type:
Name: WithdrawalXVal
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces: com.ibm.btt.struts.base.OperationXValidate
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

Figure 8-76 Create the WithdrawalXVal validator

3. Add the content of the xValidate method as shown in Example 8-15.

Example 8-15 The xValidate method snippet

```
/* (non-Javadoc)
* @see
com.ibm.btt.struts.base.OperationXValidate#xValidate(com.ibm.btt.base.Context)
*/
public String[] xValidate(Context ctx) {
   String[] xErrors = null;
   String sError = null;
   try {
      String value = ctx.getValueAt("Amount").toString();
```

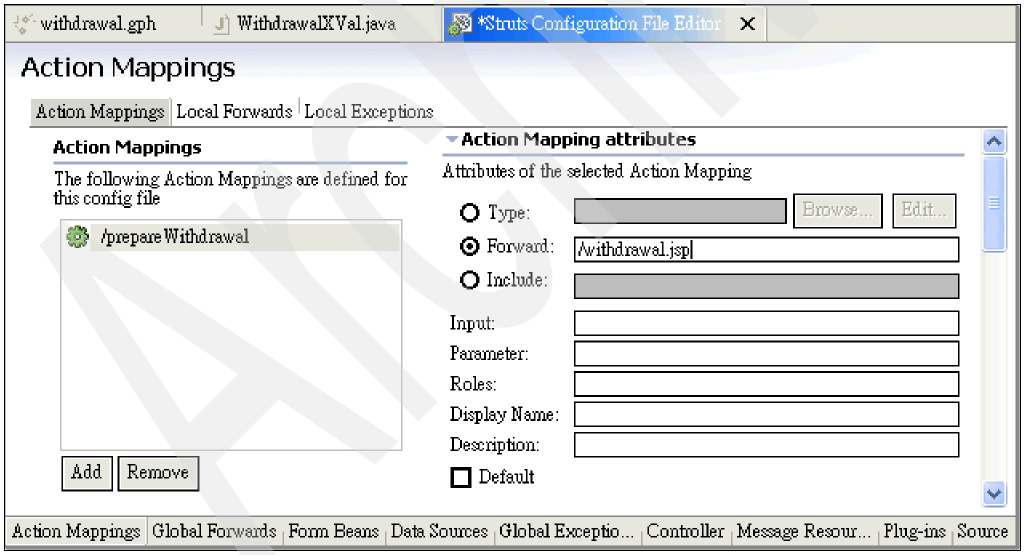
```
            if (Double.parseDouble(value.toString()) <= 0) {
               sError = "Only_non-zero_amounts_are_";
            }
         } catch (Exception e) {
            sError = "Only_non-zero_amounts_are_";
         }
         if (sError != null) {
            xErrors = new String[1];
            xErrors[0] = sError;
         }
         return xErrors;
      }
```

4. Save and close **WithdrawalXVal.java**.

Realizing the Struts actions

To realize the Struts action named /prepareWithdrawal, follow these steps:

1. Double-click the **/prepareWithdrawal** action in the Web diagram. The **struts-btt-withdrawal.xml** opens in the Struts Configuration File Editor.
2. In the Action Mapping attributes section, select the **Forward** radio button and type `/withdrawal.jsp` in the text box, as shown in Figure 8-77.



The following Action Mappings are defined for this config file

/prepareWithdrawal

Attributes of the selected Action Mapping

Type: [✓ Forward: /withdrawal.jsp] Include:

Input:

Parameter:

Roles:

Display Name:

Description:

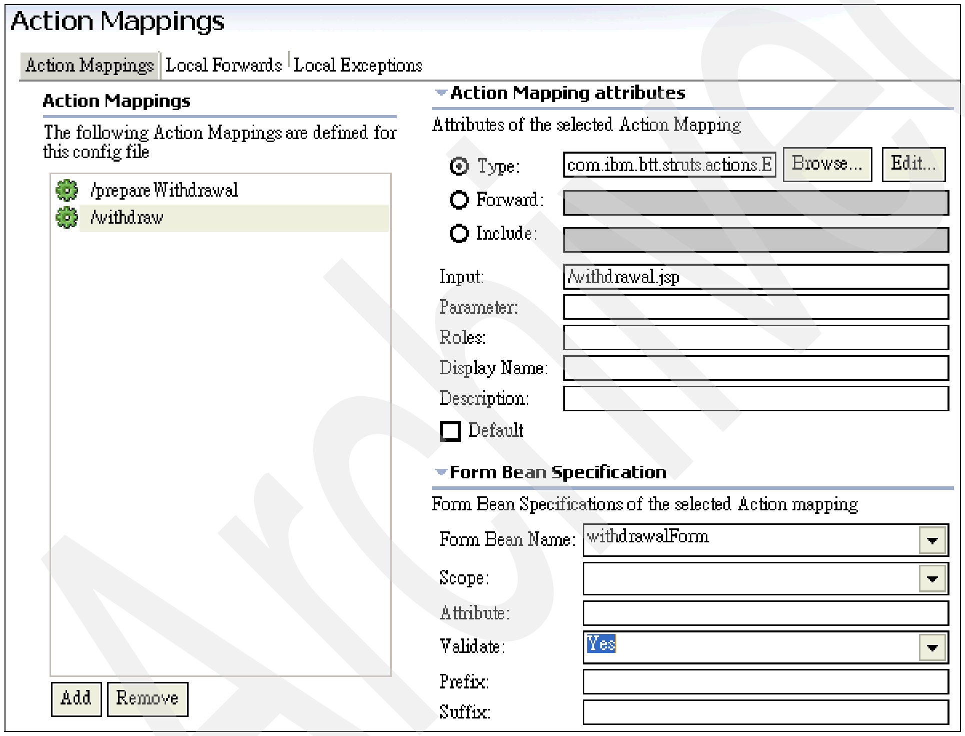
☐ Default

Figure 8-77   Realize Struts components - Realize /prepareWithdrawal action

3. Press Ctrl+S to save the change.

After switching back to the withdrawal Web diagram, to realize the Struts action named /withdraw, do the following:

1. Double-click **/withdraw** action in the Web diagram. The **struts-btt-withdrawal.xml** opens in the Struts Configuration File Editor.
2. Select the **Type** radio button in the Action Mapping attributes page, and in the Type field, enter the value `com.ibm.btt.struts.actions.EJBAction`. In the Input field, enter the value `/withdrawal.jsp` as well.
3. Select **withdrawalForm** for the Form Bean Name field in Form Bean Specification section. Select **Yes** for the Validate field, as shown in Figure 8-78.


Action Mappings
Action Mappings
Local Forwards
Local Exceptions
Action Mappings
The following Action Mappings are defined for this config file
/prepareWithdrawal
/withdraw
Add
Remove
Action Mapping attributes
Attributes of the selected Action Mapping
Browse...
Edit...
Form Bean Specification
Form Bean Specifications of the selected Action mapping

Type: com.ibm.btt.struts.actions.E

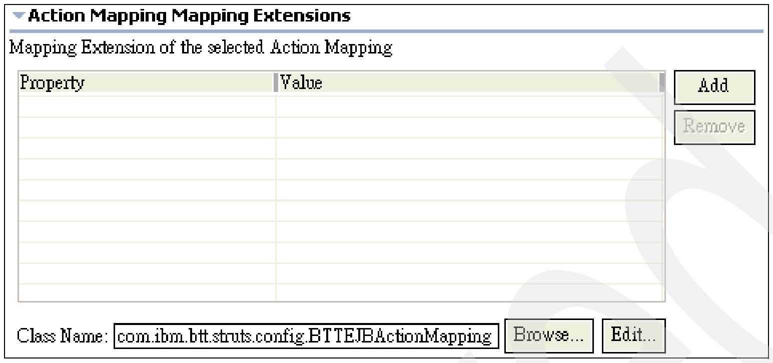
Input: /withdrawal.jsp

Form Bean Name: withdrawalForm

Validate: Yes

Figure 8-78 Select withdrawalForm for the Form Bean Name field

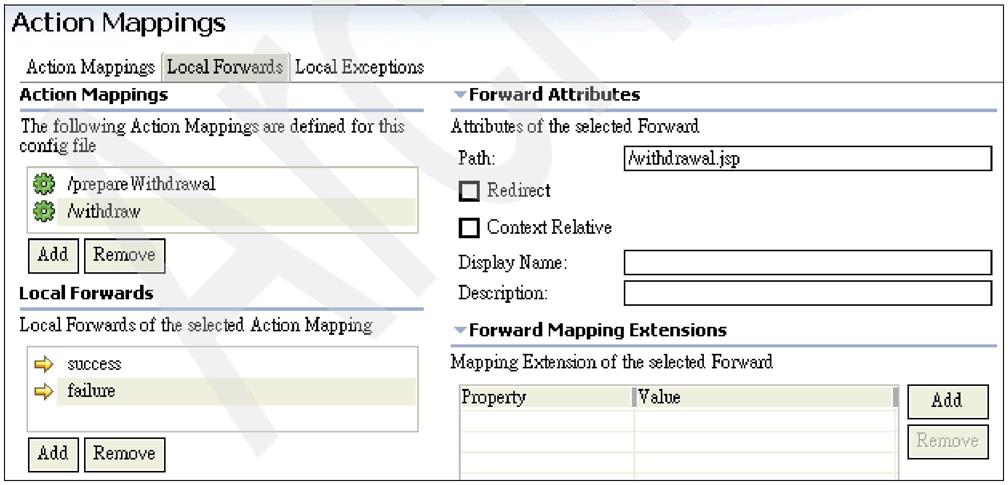
4. In the Action Mapping Extensions section, in the Class Name field, enter the value `com.ibm.btt.struts.config.BTTEJBActionMapping`, as shown in Figure 8-79.


Action Mapping Mapping Extensions
Mapping Extension of the selected Action Mapping
Property
Value
Add
Remove
Browse...
Edit...

Class Name: com.ibm.btt.struts.config.BTTEJBActionMapping

Figure 8-79 Enter com.ibm.btt.struts.config.BTTEJBActionMapping in the Class Name

5. Click the **Local Forwards** tab.
6. Click **Add** in the Local Forwards section, name the new forward as `success` and in the Path field, enter the value `/result.jsp` in the Forward Attributes section.
7. Click **Add** again to add the forward failure, with /withdrawal.jsp in the Path field, as shown in Figure 8-80.



Path: /withdrawal.jsp

Figure 8-80 Realize Struts components - realize /withdraw action - setting up Local Forwards

8. Save and close the **struts-btt-withdrawal.xml** file.

You have completed the following tasks so far:

- The Struts configuration file struts-btt-withdrawal.xml has been updated with the /prepareWithdrawal and /withdraw Action Mapping information, as shown in Example 8-16.

Example 8-16 struts-btt-withdrawal.xml snippet

```
<!-- Action Mappings -->
<action-mappings>
   <action path="/prepareWithdrawal" forward="/withdrawal.jsp">
   </action>
   <action path="/withdraw"
       type="com.ibm.btt.struts.actions.EJBAction"
       name="withdrawalForm"
       input="/withdrawal.jsp"
       className="com.ibm.btt.struts.config.BTTEJBActionMapping">
      <forward name="success" path="/result.jsp">
      </forward>
      <forward name="failure" path="/withdrawal.jsp">
      </forward>
   </action>
</action-mappings>
```

- The Web diagram has been updated, and now the /prepareWithdrawal, /withdraw Action and the local forwards failure and success appear in color to indicate that they have been realized, as shown in Figure 8-81.

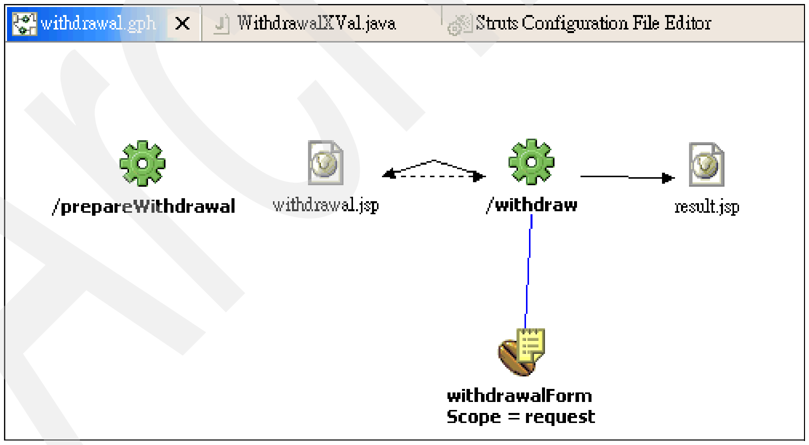


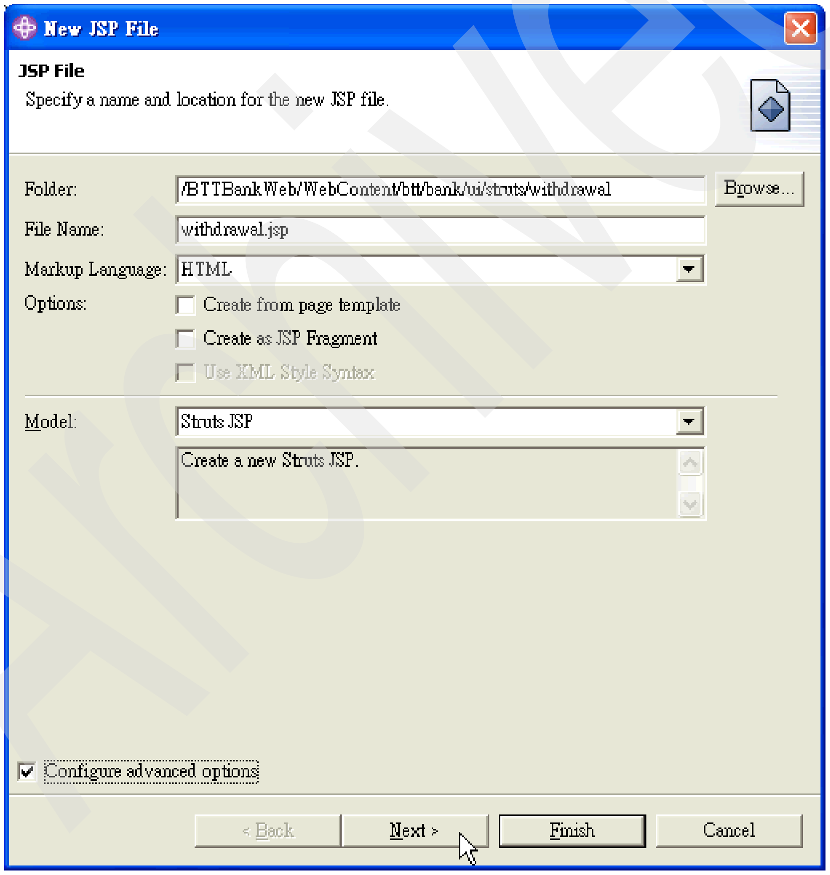
Figure 8-81 Realizing Struts components - /prepareWithdrawal and /withdraw actions realized

Realizing the JSPs

This section indicates briefly how to realize the withdrawal.jsp and result jsp.

To realize the withdrawal.jsp, perform the following tasks:

1. Expand the **BTTBankWeb/WebContent/btt/bank/ui/struts** folder, right-click and select **New** → **Folder**. Type `withdrawal` as the folder name, and click **Finish**.
2. Double-click **withdrawal.jsp** in the Web diagram.
3. When the new JSP File wizard appears, the wizard provides the default values. Accept these values. Ensure that **Struts JSP** is selected in the Model field, and **Configure advanced options** is checked. Click **Next**. See Figure 8-82.


New JSP File

JSP File

Specify a name and location for the new JSP file.

Folder: /BTTBankWeb/WebContent/btt/bank/ui/struts/withdrawal

File Name: withdrawal.jsp

Markup Language: HTML

Options:
- [ ] Create from page template
- [ ] Create as JSP Fragment
- [ ] Use XML Style Syntax

Model: Struts JSP

Create a new Struts JSP.

- [x] Configure advanced options

Figure 8-82 New JSP file wizard

4. In the tag libraries page, add the tag libraries the JSP requires. The wizard has already added the two most commonly used Struts tag libraries, that is, html and bean. You can add more tag libraries, if needed.

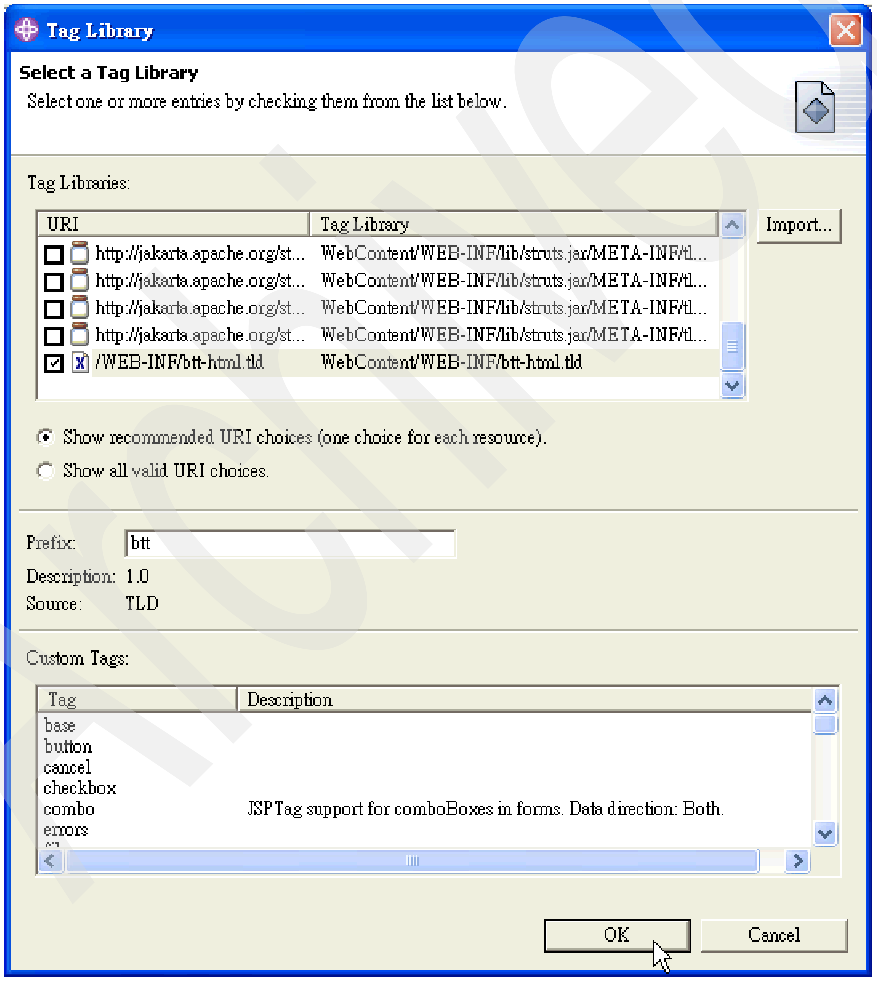
   However, in our sample, since only the html and Branch Transformation Toolkit-specific tag library is required, perform the following tasks:

   a. Select **bean** tag library, and then click **Remove**.

   b. Click **Add**. The Select Tag Library dialog box opens.

   c. Select **/WEB-INF/btt-html.tld** and change the prefix to `btt`.

   d. Click **OK**, as shown in Figure 8-83.



Select a Tag Library

Select one or more entries by checking them from the list below.

Tag Libraries:

	URI	Tag Library
☐	http://jakarta.apache.org/st...	WebContent/WEB-INF/lib/struts.jar/META-INF/tl...
☐	http://jakarta.apache.org/st...	WebContent/WEB-INF/lib/struts.jar/META-INF/tl...
☐	http://jakarta.apache.org/st...	WebContent/WEB-INF/lib/struts.jar/META-INF/tl...
☐	http://jakarta.apache.org/st...	WebContent/WEB-INF/lib/struts.jar/META-INF/tl...
☑	/WEB-INF/btt-html.tld	WebContent/WEB-INF/btt-html.tld

[● Show recommended URI choices (one choice for each resource).]
○ Show all valid URI choices.

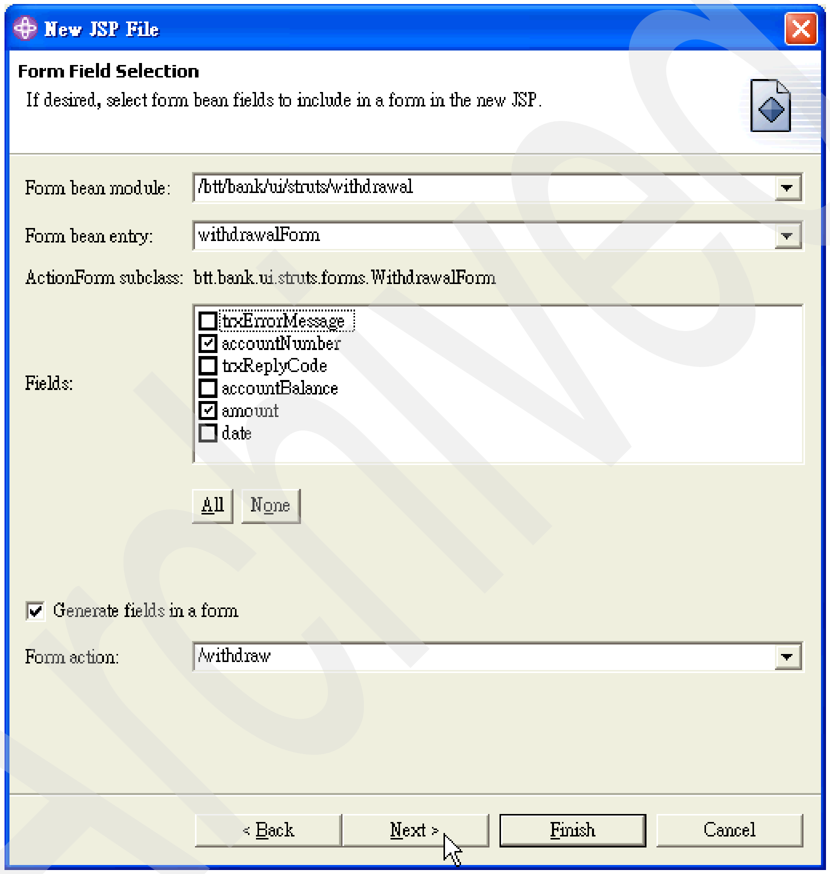
Prefix: btt
Description: 1.0
Source: TLD

Custom Tags:

Tag	Description
base	
button	
cancel	
checkbox	
combo	JSP Tag support for comboBoxes in forms. Data direction: Both.
errors	

Figure 8-83 Add the Branch Transformation Toolkit specific tag library

5. In the next page, click **Next** to accept the default, or uncheck **Use workbench encoding** to use UTF-8, and then click **Next**.
6. When the JSP File Choose Method Stubs to generate dialog box opens, accept the default settings and click **Next**.
7. In the Form Field Selection page shown in Figure 8-84, withdrawalForm is selected by default. Check the **accountNumber**, and **amount** fields, both of which are input fields. Click **Next**.


New JSP File

Form Field Selection

If desired, select form bean fields to include in a form in the new JSP.

Form bean module: /btt/bank/ui/struts/withdrawal

Form bean entry: withdrawalForm

ActionForm subclass: btt.bank.ui.struts.forms.WithdrawalForm

Fields:
- [ ] trxErrorMessage
- [✓] accountNumber
- [ ] trxReplyCode
- [ ] accountBalance
- [✓] amount
- [ ] date

[✓] Generate fields in a form

Form action: /withdraw

Figure 8-84 Choosing form bean fields for withdrawal.jsp

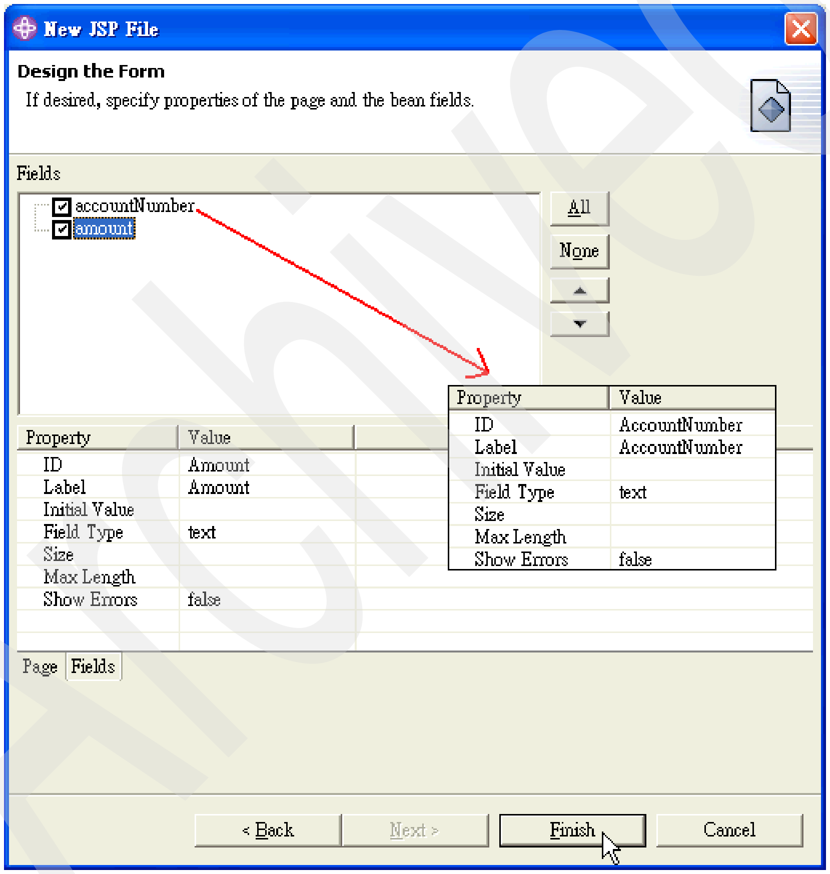
8. In the next dialog box shown in Figure 8-85, design the input form used in the withdrawal.jsp page. For the accountNumber field, input the following parameters:

   – ID: `AccountNumber`
   – Label: `AccountNumber`

   For the amount field, input the following parameters:

   – ID: `Amount`
   – Label: `Amount`

9. Click **Finish**.



Property	Value
ID	Amount
Label	Amount
Initial Value	
Field Type	text
Size	
Max Length	
Show Errors	false

Property	Value
ID	AccountNumber
Label	AccountNumber
Initial Value	
Field Type	text
Size	
Max Length	
Show Errors	false

Figure 8-85 Designing input fields

The withdrawal.jsp is created and opened in the Page Designer. As you can see in the editor, the wizard-generated withdrawal.jsp file is simple. You can tailor it by adding more elements and Branch Transformation Toolkit-specific features as described in the section "Using the Struts Tools BTT Extensions" on page 347.

To implement the result.jsp, perform the following tasks:

1. Double-click the **result.jsp** in the Web diagram.
2. When the New JSP File wizard appears, the wizard provides the default values. Accept these values. Ensure that **Struts JSP** is selected in the Model field, and **Configure advanced options** is checked. Click **Next**.
3. In the tag libraries page, use **Struts html** and Branch Transformation Toolkit **btt** tag libraries.
4. In the next dialog box, click **Next** to accept the default or *deselect* **Use workbench encoding** to use UTF-8. Click **Next**.
5. When the JSP File Choose Method Stubs to generate dialog box opens, accept the default settings and click **Next**.
6. In the Form Field Selection page, you are asked to supply the name of the form bean from which withdrawal.jsp should get its fields, and specify which fields to use.

   Choose **withdrawalForm** in the Form bean entry drop-down list. Select the **trxErrorMessage**, **trxReplyCode**, and **accountBalance** properties.

   – trxErrorMessage: Input the following parameters here:
     - Field type: `text`
     - ID: `TrxErrorMessage`
     - Label: `TrxErrorMessage`

   – trxReplyCode: Input the following parameters here:
     - Field type: `text`
     - ID: `TrxReplyCode`
     - Label: `TrxReplyCode`

   – accountBalance: Input the following parameters here:
     - Field type: `text`
     - ID: `AccountBalance`
     - Label: `AccountBalance`
7. Select **/withdraw** for the Form action field. Click **Finish**.

The withdrawal.jsp and result.jsp pages are now created, as is the Web diagram, as shown in Figure 8-86.

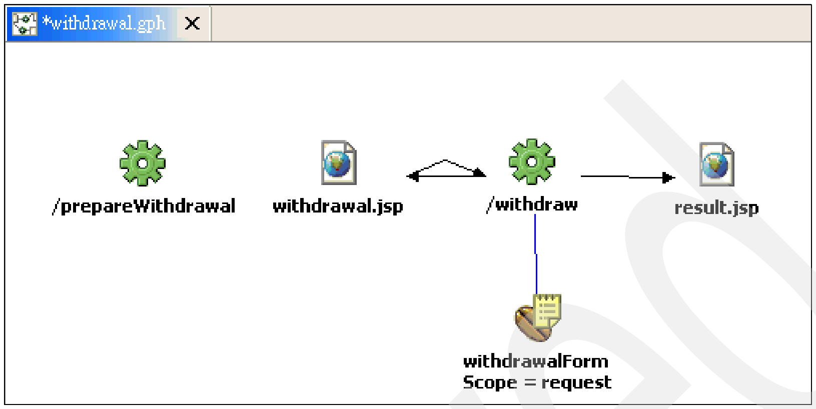
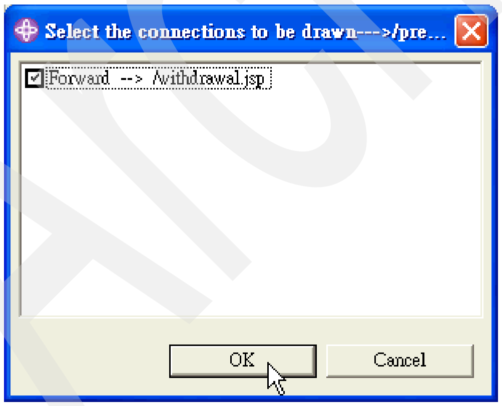


Figure 8-86 Realize Struts components - Realizing JSP pages

Completing the Web diagram

As you can see in Figure 8-86, the /prepareWithdrawal action has no connection to other components. Bring it's Forward attribute to this diagram.

1. Right-click **/prepareWithdrawal** action. Select **Draw** → **Draw Selected From** from the context menu.
2. Select **Forward** → **/withdrawal.jsp** check box and click **OK**. See Figure 8-87.


Select the connections to be drawn--->/pre...

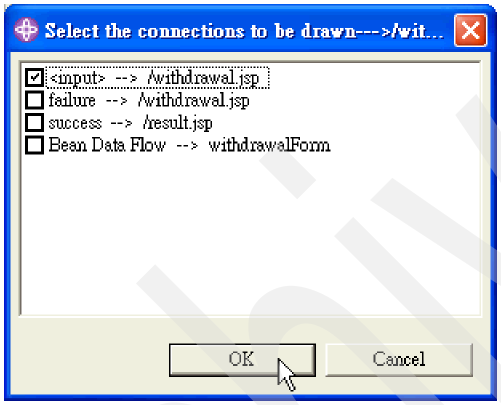
☑ Forward --> /withdrawal.jsp

Figure 8-87 Add Forward attribute to Web diagram

Note: If you perform the **Draw** → **Draw Selected From** action and nothing happens, you should close the Web diagram and reopen it to fix the Web diagram editor.

Perform one more task to add the Action Input forward for /withdraw action.

1. Right-click **/withdraw** action. Select **Draw** → **Draw Selected From** from the context menu.
2. Select **<input>** → **/withdrawal.jsp** check box and click **OK**, as shown in Figure 8-88.



[✓] <input> --> /withdrawal.jsp
[ ] failure --> /withdrawal.jsp
[ ] success --> /result.jsp
[ ] Bean Data Flow --> withdrawalForm

Figure 8-88 Add the Action Input forward for /withdraw action

3. Press Ctrl+S to save your work. Your final Web diagram should look as shown in Figure 8-89.

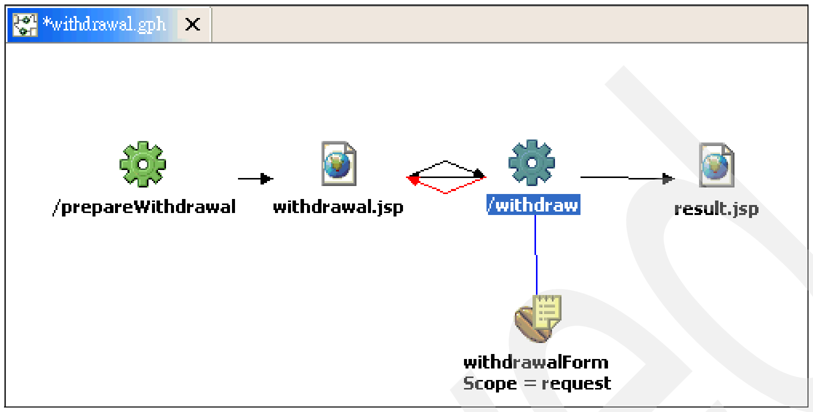


Figure 8-89 Realize Struts components - /prepareWithdrawal with Forward attribute displayed and /withdraw actions realized with Action Input forward

Using the Struts Tools BTT Extensions

The Struts Tools BTT Extensions provides a graphical and easier way to work with toolkit-extended Struts configuration files. The Struts Tools BTT Extensions has the following features:

- Screen flow context

  Specify the screen flow context used in Struts actions mappings from the Screen Flow Context tab.

- Action mapping

  Specify the BTT Extended Struts Action Type of each Struts standard action from the Action Mapping tab.

- Final nodes

  Specify the final nodes in the Struts process from the Final Nodes tag.

- WSIF message mapper

  Define the WSIFmessage mapper for your WSIF Action from the WSIF Message Mapper tab.

- Action condition

  Specify the conditions trigger an action from the Action Condition tab.

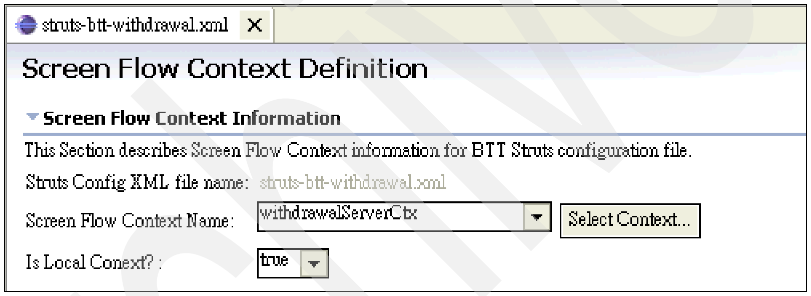
- Processor mapper

  Specify the processor mapper that maps the flow context between the parent flow and the subflow.

Configuring Struts Tools BTT Extensions for the sample application

To configure BTT Extensions for struts-btt-withdrawal.xml, do the following:

1. Close **struts-btt-withdrawal.xml** if it is not already closed.
2. In the Project Navigator view, expand the **BTTBankWeb/WebContent/WEB-INF** folder. Right-click the **struts-btt-withdrawal.xml** file, and select **Open With** → **Struts Tools BTT Extensions**.
3. Select the **Screen Flow Context** tab. Use the following values in the Screen Flow Context Definition page, as shown in Figure 8-90:
   – Screen Flow Context Name: `withdrawalServerCtx`
   – Is Local Context?: `true`


struts-btt-withdrawal.xml
Screen Flow Context Definition
Screen Flow Context Information
Select Context...

This Section describes Screen Flow Context information for BTT Struts configuration file.

Struts Config XML file name: struts-btt-withdrawal.xml

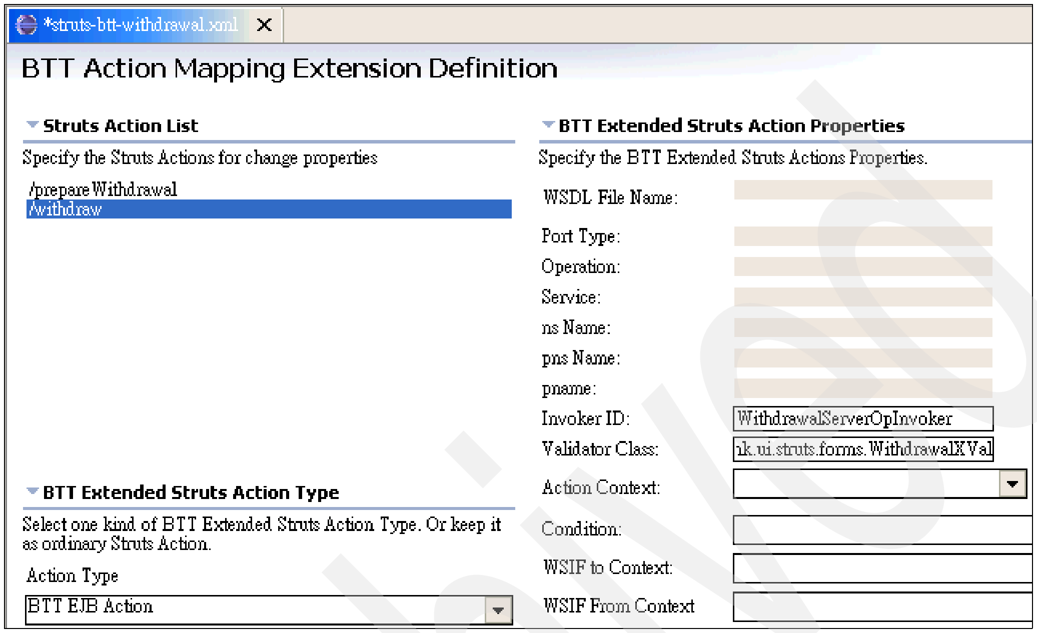
Screen Flow Context Name: withdrawalServerCtx

Is Local Conext? : true

Figure 8-90 Configure Screen Flow Context

4. Click the **Action Mapping** tab. Select **/withdraw** in the Struts Action List section, and enter the value `WithdrawalServerOpInvoker` in the Invoker ID

field. Enter `btt.bank.ui.struts.forms.WithdrawalXVal` in the Validator Class field, as shown in Figure 8-91.



*struts-btt-withdrawal.xml

BTT Action Mapping Extension Definition

Struts Action List

Specify the Struts Actions for change properties

/prepareWithdrawal
/withdraw

BTT Extended Struts Action Type

Select one kind of BTT Extended Struts Action Type. Or keep it as ordinary Struts Action.

Action Type: BTT EJB Action

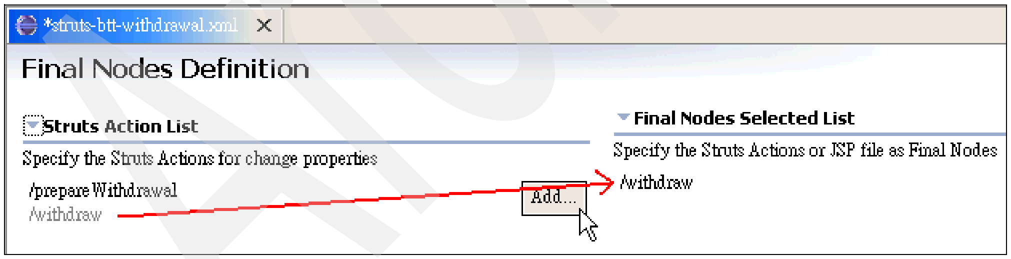
BTT Extended Struts Action Properties

Specify the BTT Extended Struts Actions Properties.

WSDL File Name:	
Port Type:	
Operation:	
Service:	
ns Name:	
pns Name:	
pname:	
Invoker ID:	WithdrawalServerOpInvoker
Validator Class:	k.ui.struts.forms.WithdrawalXVal
Action Context:	
Condition:	
WSIF to Context:	
WSIF From Context	

Figure 8-91 Configure BTT action mapping extension

5. Click the **Final Nodes** tab. Select **/withdraw** in the Struts Action List section, and click **Add** to add this as the BTT Final Node, as shown in Figure 8-92.



*struts-btt-withdrawal.xml

Final Nodes Definition

Struts Action List

Specify the Struts Actions for change properties

/prepareWithdrawal
/withdraw

Final Nodes Selected List

Specify the Struts Actions or JSP file as Final Nodes

/withdraw

Figure 8-92 Configure final nodes

6. Press Ctrl+S to save the change.

Note: After you save the file, you might see the error message, **“**`Exactly one of "forward", "include" or "type" must be specified for /prepareWithdrawal.`**”** To fix the problem, follow these steps:

1. Reopen struts-btt-withdrawal.xml using **Struts Configuration File Editor**. Click the **Source** tab.
2. Delete **type="org.apache.struts.action.Action"** from the action /prepareWithdrawal.
3. Press Ctrl+S to save the change.

Tailoring the JSPs

As mentioned earlier, Branch Transformation Toolkit provides its tag library to work with the entire framework of the Struts Tools BTT Extensions. The JSP wizard provided by WebSphere Studio Application Developer Integration Edition 5.1.1 does not know about the Branch Transformation Toolkit’s add-on. You should, therefore, tailor the generated JSPs to utilize the feature provided by the Struts Tools BTT Extensions.

The withdrawal.jsp that this application uses is a little different from the withdrawal.jsp page the wizard has generated:

- This JSP defines the two Struts tag libraries you use:

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

- It then defines the start of the HTML section using the following tag:

  ```
  <html:html>
  ```

  Change this to `<btt:html>` and change the closing tag to `</btt:html>`.

- Under the <BODY> tag, display error messages by adding the `<btt:errors>` tag:

  ```
  <btt:errors/>
  ```

- It then defines the start of the FORM section using the following tag:

  ```
  <html:form>
  ```

  Change this to `<btt:form>` and change the closing tag to `</btt:form>`.

- The `<html:text>` tag is used to populate the input field with the contents of the corresponding field from the WithdrawalForm form bean. Likewise, Branch Transformation Toolkit provides `the <btt:text>` tag for the same purpose:

  ```
  <TR>
     <TH>AccountNumber</TH>
     <TD><html:text property="AccountNumber" /></TD>
  ```

```
</TR>
<TR>
   <TH>Amount</TH>
   <TD><html:text property="Amount" /></TD>
</TR>
```

Change this to:

```
<TR>
   <TD align="right"><btt:label text="AccountNumber"/></TD>
   <TD align="left"><btt:text property="AccountNumber"/></TD>
</TR>
<TR>
   <TD align="right"><btt:label text="Amount"/></TD>
   <TD align="left"><btt:text property="Amount"/></TD>
</TR>
```

- The Submit button of the form is rendered using the `<html:submit>` tag:

  ```
  <html:submit property="submit" value="Submit" />
  ```

  You can use more suitable values such as:

  ```
  <html:submit property="Withdraw" value="Withdrawal"/>
  ```

The complete withdrawal JSP is shown in Example 8-17.

Example 8-17 The complete withdrawal.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<btt:html>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
%>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>withdrawal.jsp</TITLE>
</HEAD>

<BODY>
<btt:errors/>
<btt:form action="/withdraw">
   <TABLE border="0">
      <TBODY>
         <TR>
            <TD align="right"><btt:label text="AccountNumber"/></TD>
            <TD align="left"><btt:text property="AccountNumber"/></TD>
```

```
            </TR>
            <TR>
               <TD align="right"><btt:label text="Amount"/></TD>
               <TD align="left"><btt:text property="Amount"/></TD>
            </TR>
            <TR>
               <TD><html:submit property="Withdraw" value="Withdrawal"/></TD>
               <TD><html:reset /></TD>
            </TR>
         </TBODY>
      </TABLE>
</btt:form>
</BODY>
</btt:html>
```

For result.jsp, follow the same procedure. The end result for this is similar to that shown in Example 8-18.

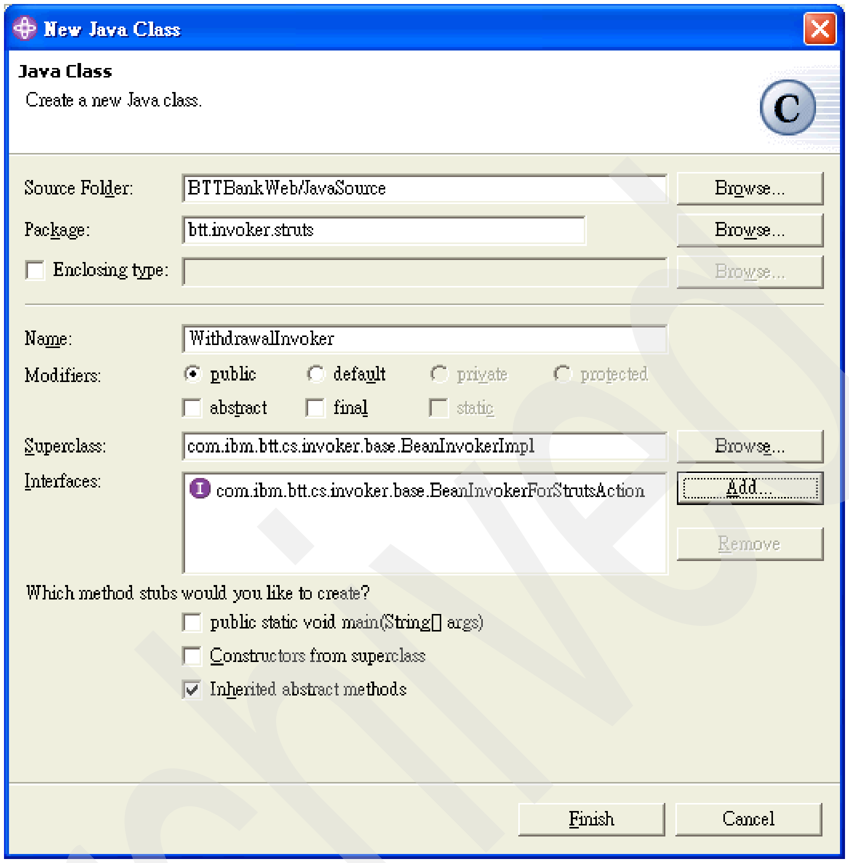
Example 8-18 The complete result.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<btt:html>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
%>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>result.jsp</TITLE>
</HEAD>

<BODY>
<btt:errors/>
<btt:form action="/withdraw">
   <TABLE border="0">
      <TBODY>
         <TR>
            <TD align="right"><btt:label text="TrxErrorMessage" /></TD>
            <TD align="left"><btt:text readonly="true"
                                 property="TrxErrorMessage" /></TD>
         </TR>
         <TR>
            <TD align="right"><btt:label text="TrxReplyCode" /></TD>
            <TD algin="left"><btt:text readonly="true"
                                 property="TrxReplyCode" /></TD>
         </TR>
         <TR>
            <TD align="right"><btt:label text="AccountBalance" /></TD>
            <TD align="left"><btt:text readonly="true"
                                 property="AccountBalance" /></TD>
         </TR>
         <TR>
            <TD><html:button property="Back" value="Back"
               onclick="javascript:window.location='prepareWithdrawal.do'"/>
            </TD>
            <TD></TD>
         </TR>
      </TBODY>
   </TABLE>
</btt:form>
</BODY>
</btt:html>
```

Creating an invoker for Struts actions

To create a Single Action EJB Invoker, perform the following tasks:

1. In the Project Navigator view, expand the **BTTBankWeb/JavaSource** folder, right-click **btt.invoker.struts** package. Select **New** → **Class** from the context menu.
2. Input the following properties for the new class, as shown in Figure 8-93 on page 354:
   - Name: `WithdrawalInvoker`
   - Superclass: `com.ibm.btt.cs.invoker.base.BeanInvokerImpl`
   - Interfaces: `com.ibm.btt.cs.invoker.base.BeanInvokerForStrutsAction`


New Java Class

Java Class

Create a new Java class.

Source Folder: BTTBankWeb/JavaSource

Package: btt.invoker.struts

Enclosing type: [ ]

Name: WithdrawalInvoker

Modifiers: [✓ public] default private protected; abstract [ ] final [ ] static [ ]

Superclass: com.ibm.btt.cs.invoker.base.BeanInvokerImpl

Interfaces: com.ibm.btt.cs.invoker.base.BeanInvokerForStrutsAction

Which method stubs would you like to create?

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

[ ] Constructors from superclass

[✓] Inherited abstract methods

Figure 8-93 Create WithdrawalInvoker

3. Click **Finish**.
4. Complete it as shown in Example 8-19.

Example 8-19 WithdrawalInvoker.java

```
/*
 * Created on Nov 25, 2005
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package btt.invoker.struts;

import java.util.Hashtable;
import btt.bank.business.logic.WithdrawalServerOp;
import btt.bank.business.logic.WithdrawalServerOpHome;
import com.ibm.btt.base.Constants;
import com.ibm.btt.base.Context;
import com.ibm.btt.base.DSEInvalidRequestException;
import com.ibm.btt.base.DSEObjectNotFoundException;
import com.ibm.btt.cs.invoker.base.BeanInvokerForStrutsAction;
import com.ibm.btt.cs.invoker.base.BeanInvokerImpl;

/**
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class WithdrawalInvoker
   extends BeanInvokerImpl
   implements BeanInvokerForStrutsAction {

   /* (non-Javadoc)
    * @see com.ibm.btt.cs.invoker.base.BeanInvokerImpl#executeEJB()
    */
   public Object executeEJB() throws Exception {
      WithdrawalServerOp bean = (WithdrawalServerOp) getBeanInvokerProxy();
      Hashtable result = (Hashtable) bean.execute(getSystemData(),getEjbParameters());
      // save the result
      Context repCtx = getContext();
      repCtx.setValueAt("AccountBalance", result.get("AccountBalance"));
      repCtx.setValueAt("TrxReplyCode", result.get("TrxReplyCode"));
      repCtx.setValueAt("TrxErrorMessage", result.get("TrxErrorMessage"));
      return result;
   }

   /* (non-Javadoc)
    * @see com.ibm.btt.cs.invoker.base.BeanInvoker#processRespondData(java.lang.Object)
    */
   public Object processRespondData(Object ejbResult)
      throws DSEInvalidRequestException {
         //The ejbResult is SAE execution result.
         String sTrxReplyCode = (String)((Hashtable)ejbResult).get("TrxReplyCode");
         String forwardName = null;
         if (sTrxReplyCode.equals("00")){
            forwardName = "success";
         }else{
            forwardName = "failure";
         }
         return forwardName;
   }

   /* (non-Javadoc)
    * @see com.ibm.btt.cs.invoker.base.BeanInvoker#createBeanInvokerProxy()
    */
   public Object createBeanInvokerProxy() throws DSEInvalidRequestException {
      //** 1: Get EJBHome Object
      WithdrawalServerOpHome home = (WithdrawalServerOpHome) getHomeObject();

      //** 2: Create Bean Proxy.
      WithdrawalServerOp bean = null;
      try {
         bean = (WithdrawalServerOp) home.create();
      } catch (Exception e) {
         throw new DSEInvalidRequestException(Constants.COMPID, "", e.getMessage());
      }
      return bean;
   }

   /* (non-Javadoc)
    * @see 
com.ibm.btt.cs.invoker.base.BeanInvokerForStrutsAction#parseRequestData(com.ibm.btt.base.Contex
t)
    */
   public void parseRequestData(Context reqCtx)
      throws DSEInvalidRequestException, DSEObjectNotFoundException {
         Hashtable htEjbPara = getEjbParameters();
         try {
            this.setContext(reqCtx);
            htEjbPara.put("Date",new java.util.Date());
            htEjbPara.put("AccountNumber",reqCtx.getValueAt("AccountNumber"));
            htEjbPara.put("Amount",new 
Float(Float.parseFloat((String)reqCtx.getValueAt("Amount"))));
         } catch (Exception e) {
            e.printStackTrace();
            throw new DSEInvalidRequestException(Constants.COMPID, "", e.getMessage());
         }
   }

   /* (non-Javadoc)
    * @see 
com.ibm.btt.cs.invoker.base.BeanInvokerForStrutsAction#setSessionObject(java.lang.Object)
    */
   public void setSessionObject(Object arg0)
      throws DSEInvalidRequestException, DSEObjectNotFoundException {
      // TODO Auto-generated method stub
   }
}
```

To configure the Single Action EJB Invoker properties file, perform the following tasks:

1. Create a properties file named `WithdrawalServerOpInvoker.properties` in the package btt.invoker.struts, and input the content shown in Example 8-20.

Example 8-20 WithdrawalServerOpInvoker.properties

```
implClass=btt.invoker.struts.WithdrawalInvoker
jndiName=ejb/btt/bank/business/logic/WithdrawalServerOpHome
factory=com.ibm.websphere.naming.WsnInitialContextFactory
location=iiop://localhost:2809
homeClassName=btt.bank.business.logic.WithdrawalServerOpHome
isLocal=false
csReplyFormat=
```

2. In the Project Navigator view, expand the **BTTBankWeb/JavaSource** folder. In com.ibm.btt.cs.invoker.base package, double-click the **BeanInvokerRegistryMapper.properties** file to open it.
3. Add the following line:

   ```
   WithdrawalServerOpInvoker=btt.invoker.struts.WithdrawalServerOpInvoker:RB
   ```

4. Save and close the **Properties File Editor**.

Running the BTT Bank sample application

This section describes the process involved in running the sample application and exploring the functionality built using the WebSphere Studio Application Developer Integration Edition and its support for rapid development of Struts-based Web applications.

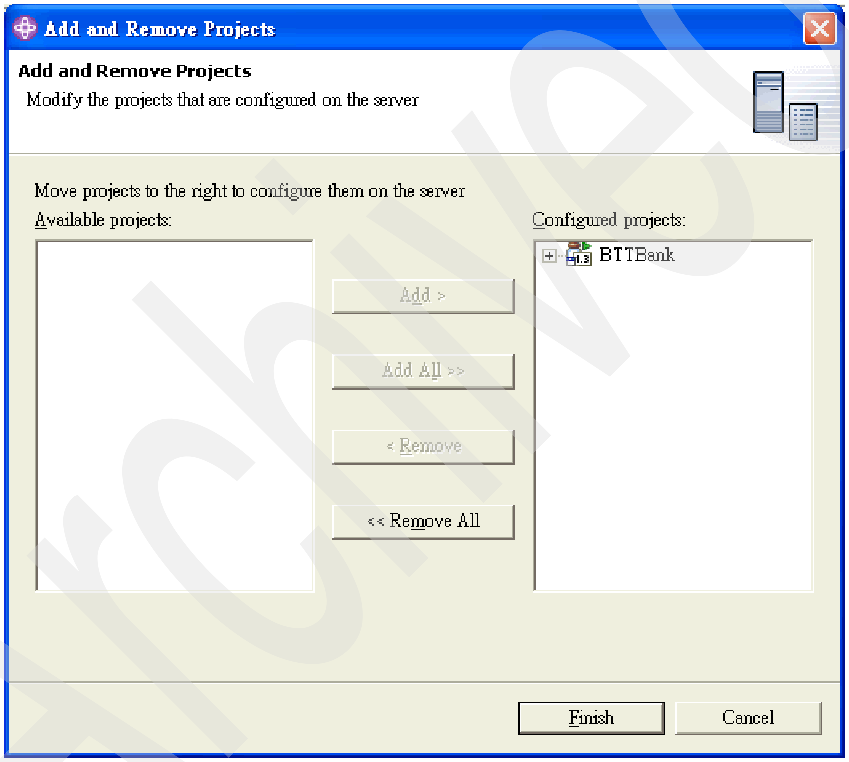
Preparing the runtime environment

To prepare the runtime environment, follow these steps:

- Copy the configuration files by performing the following tasks:
  a. Open the **Java** perspective, and expand the **BTTBankBTT/business** folder.
  b. Copy all the files, including **dse.ini**, **dsectxt.xml**, **dsedata.xml**, **dsefmts.xml**, and **dsetype.xml**, to the new folder, dse, created in your C drive.
- Associate the BTTBank EAR project with the server configuration.
  a. From the Server perspective, right-click **WBI SF** in the Server Configuration panel.
  b. Select **Add and remove projects** from the context menu.

   c. In the pop-up dialog, select **BTTBank** and click **Add**. You will see that BTTBank appears in the Configured projects list. Click **Finish**, as shown in Figure 8-94 on page 358.

- Regenerate Deployment and RMIC Code. Use the following steps to regenerate Deployment and RMIC Code for the BTTBankEJB, BTTCHAEJB and BTTFormatterEJB projects:
   a. Right-click the project.
   b. Select **Generate** → **Deployment and RMIC Code**.
   c. Click **Select all**, and then **Finish** in the pop-up dialog box in Figure 8-94.


Add and Remove Projects

Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

Available projects:

Configured projects:

BTTBank

Figure 8-94 Add BTTBank into the server configuration

Running the sample code

To ensure that the server is running, do the following:

1. In the Server perspective, right-click **WBI SF** either from the Server Configuration view or the Servers view.
2. Select **Start** from the context menu.

In the console view, you will see a lot of messages. This means that the server is running. When you see the message "`Server server1 open for e-business`", it means the server is fully started.

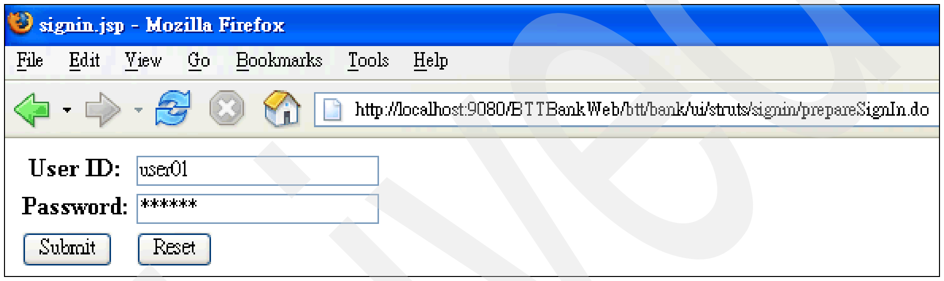
You can test the application using your Web browser:

1. Get into the sign-in page using the following URL, as shown in Figure 8-95:

   `http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/signin/prepareSignIn.do`

   Only two sets of ID/Password work for this sample:

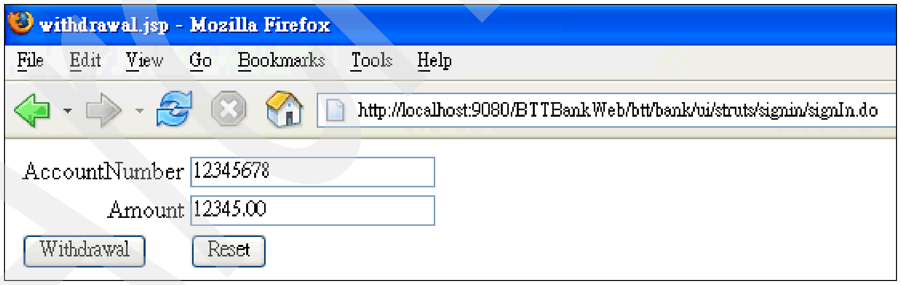
   – ID: `user01`, Password: `user01`
   – ID: `user02`, Password: `user02`


signin.jsp - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/signin/prepareSignIn.do
Submit
Reset

User ID: user01
Password: ******

Figure 8-95 The sign-in page

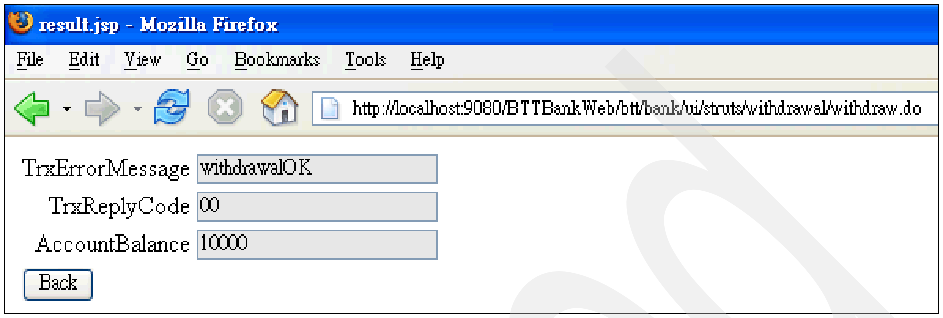
2. Click **Submit** in the sign-in page. The withdrawal page shown in Figure 8-96 appears.


withdrawal.jsp - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/signin/signIn.do
Withdrawal
Reset

AccountNumber 12345678
Amount 12345.00

Figure 8-96 The withdrawal page

3. Input the arbitrary Account Number and Amount, and then click **Withdrawal**. The result page with the hard-coded values will be displayed as shown in Figure 8-97.


result.jsp - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/withdrawal/withdraw.do
Back

TrxErrorMessage	withdrawalOK
TrxReplyCode	00
AccountBalance	10000

Figure 8-97 The result page

8.3.9 Adding the Journal service

The Branch Transformation Toolkit provides a set of service objects that enable an application to complete an operation. These services include client/server connectivity, financial devices for input and output operations, table mapping database service, electronic journal, generic pool, and so on.

In Branch Transformation Toolkit V5.1, services can be categorized into client side services and server side services. For those services that belong to the client side, only Java clients can utilize them. Server side services can be invoked by a business process or a Single Action EJB.

This section discusses the electronic journal service. Our discussion is divided into two parts, that is, setting up the service infrastructure and constructing the dummy journal service.

Setting up the service infrastructure

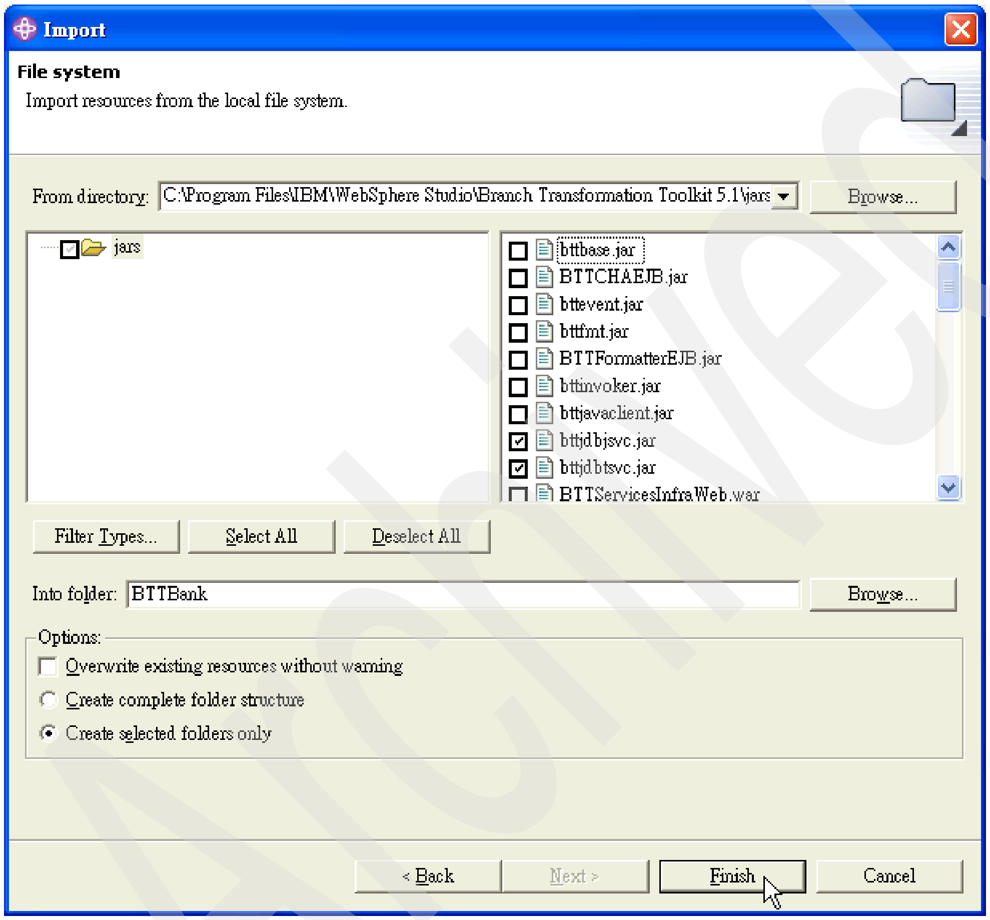
To add a dummy journal service such as the CHA and Formatter services, the Service EJB project bttsvcinfra.jar should be imported into the application. There are four different service invocation types in Branch Transformation Toolkit v5.1. If you are using the WSIF invocation type, the extra Service Web project BTTServicesInfraWeb.war should be imported as well.

Importing utility JARs

To import the utility JAR files, follow these steps:

1. In the J2EE perspective, select **BTTBank** project in the Project Navigator view, right-click, and select **Import** from the context menu.

2. In the Import pop-up window, select **File system** as the import source, and then click **Next**.
3. In the next dialog box, click **Browse** and navigate to **<BTT_install_dir>/jars**, and then click **OK**. Select **bttjdbjsvc.jar** and **bttjdbtsvc.jar**. Click **Finish**, as shown in Figure 8-98.


Import

File system

Import resources from the local file system.

From directory: C:\Program Files\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1\jars

- jars
- [ ] bttbase.jar
- [ ] BTTCHAEJB.jar
- [ ] bttevent.jar
- [ ] bttfmt.jar
- [ ] BTTFormatterEJB.jar
- [ ] bttinvoker.jar
- [ ] bttjavaclient.jar
- [x] bttjdbjsvc.jar
- [x] bttjdbtsvc.jar
- [ ] BTTServicesInfraWeb.war

Into folder: BTTBank

Options:
- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

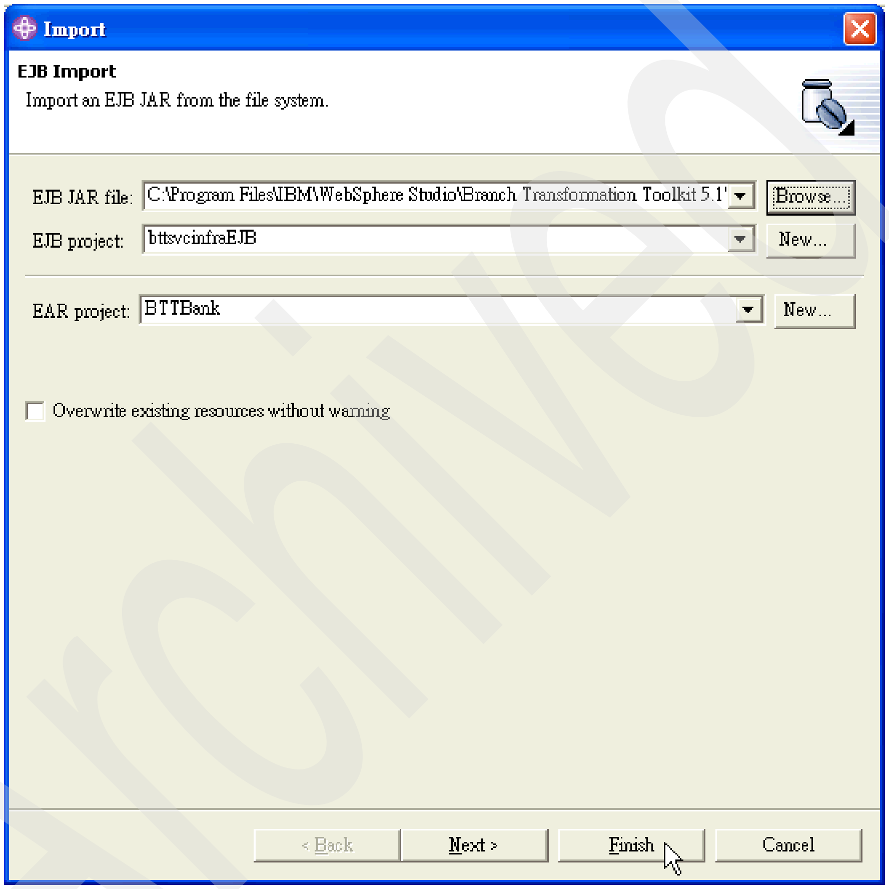
Figure 8-98 Import bttjdbjsvc.jar and bttjdbtsvc.jar into the BTTBank project

Importing the Service EJB project bttsvcinfra.jar

1. Open the **J2EE** perspective and get into the J2EE Hierarchy view.
2. Right-click **BTTBank** in the Enterprise Applications folder. Select **Import** → **Import EJB Jar**.

3. Input the following values for EJB import as shown in Figure 8-98 on page 361, and click **Finish**:
   – EJB JAR File: `<BTT_install_dir>\jars\bttsvcinfra.jar`
   – EJB project: `bttsvcinfraEJB`
   – EAR project: `BTTBank`

.



EJB JAR file: C:\Program Files\IBM\WebSphere Studio\Branch Transformation Toolkit 5.1'
EJB project: bttsvcinfraEJB
EAR project: BTTBank
☐ Overwrite existing resources without warning

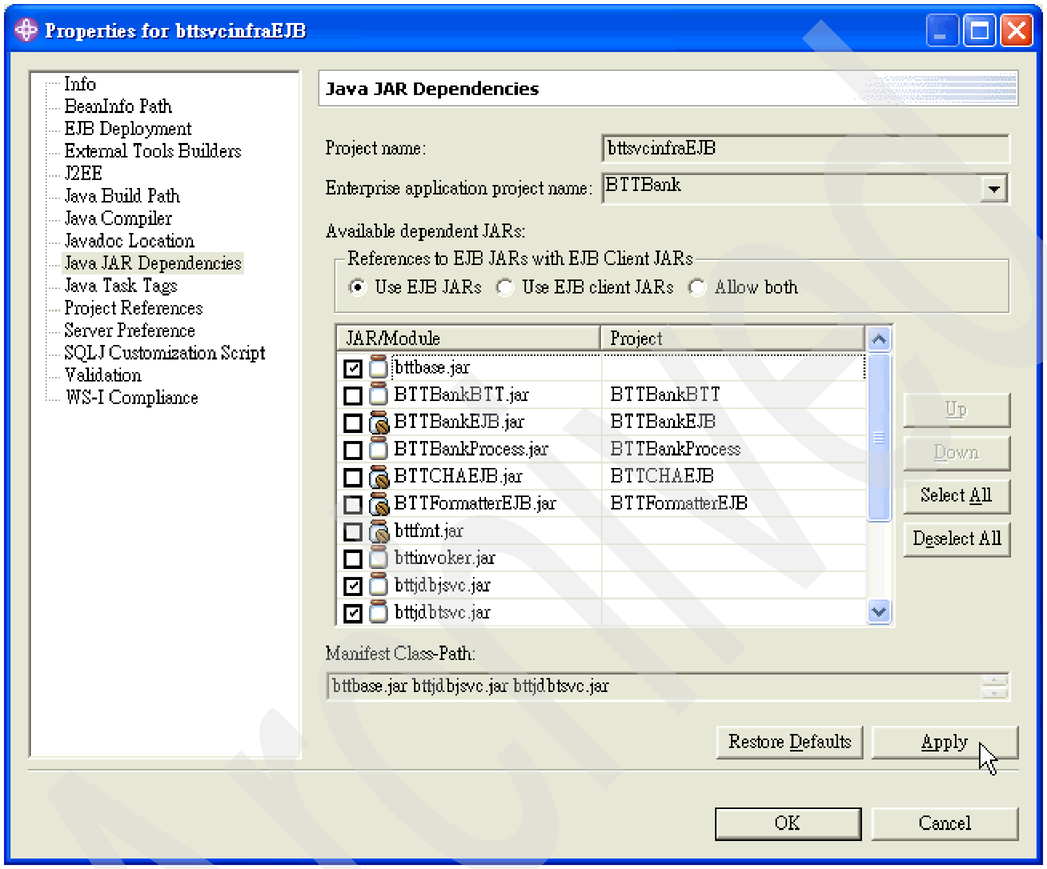
Figure 8-99 Import the bttsvcinfraEJB project

Setting up project properties for bttsvcinfraEJB

To set up the project properties, follow these steps:

1. Select the **Project Navigator** view tab. The bttsvcinfra EJB Project appears.
2. Right-click the **bttsvcinfraEJB** project, and select **Properties** from the context menu.

3. In the pop-up window, select **Java JAR Dependencies**. Select the **Use EJB JARs** radio button, and then ensure that the **bttbase.jar**, **bttjdbjsvc.jar**, and **bttjdbtsvc.jar** check boxes have been selected in the JAR/Module list. Click **Apply**, as shown in Figure 8-100.



Project name: bttsvcinfraEJB

Enterprise application project name: BTTBank

Available dependent JARs:

References to EJB JARs with EJB Client JARs

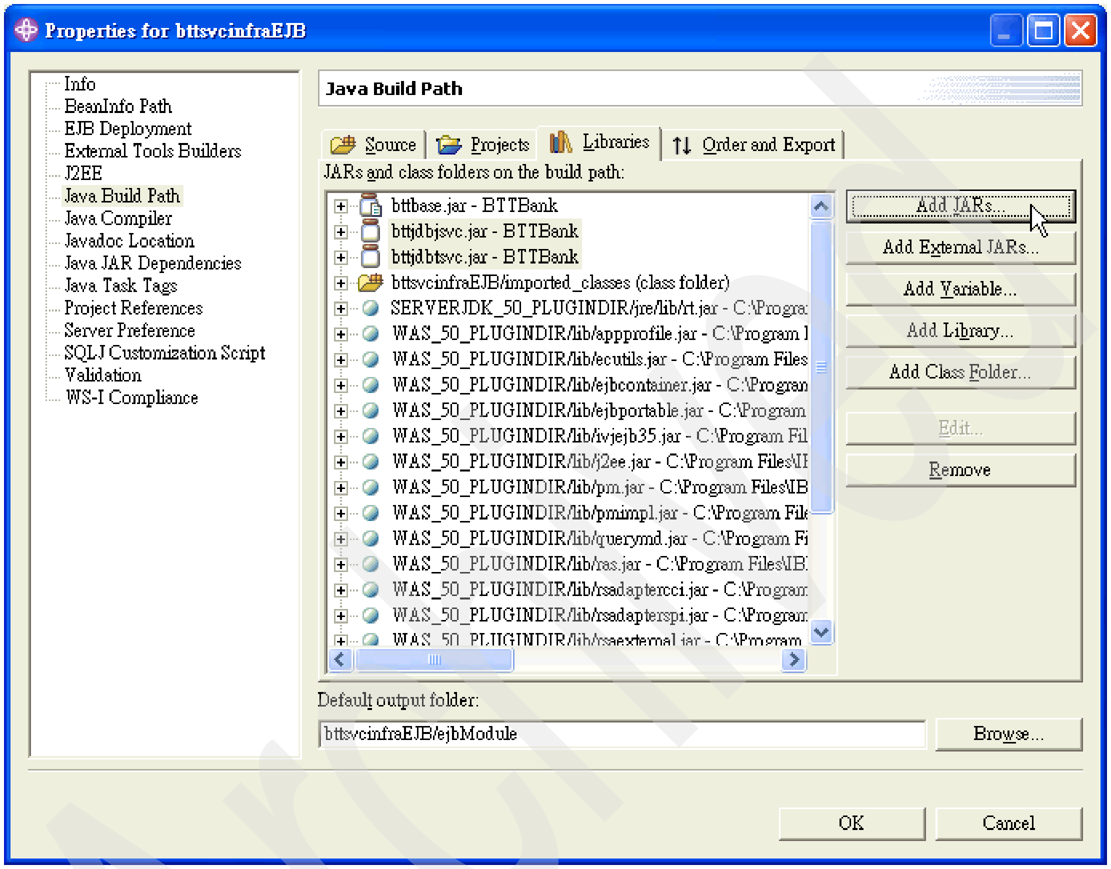
[✓ Use EJB JARs] Use EJB client JARs Allow both

JAR/Module	Project
✓ bttbase.jar	
☐ BTTBankBTT.jar	BTTBankBTT
☐ BTTBankEJB.jar	BTTBankEJB
☐ BTTBankProcess.jar	BTTBankProcess
☐ BTTCHAEJB.jar	BTTCHAEJB
☐ BTTFormatterEJB.jar	BTTFormatterEJB
☐ bttfmt.jar	
☐ bttinvoker.jar	
✓ bttjdbjsvc.jar	
✓ bttjdbtsvc.jar	

Manifest Class-Path: bttbase.jar bttjdbjsvc.jar bttjdbtsvc.jar

Figure 8-100 Add bttbase.jar, bttjdbjsvc.jar, and bttjdbtsvc.jar as dependent JARs

4. In the same pop-up window, select **Java Build Path**, and then click the **Libraries** tab. Ensure that three JARs, that is, **bttbase.jar**, **bttjdbjsvc.jar**, and **bttjdbtsvc.jar** are added to the build path, as shown in Figure 8-101.



JARs and class folders on the build path:

```
bttbase.jar - BTTBank
bttjdbjsvc.jar - BTTBank
bttjdbtsvc.jar - BTTBank
bttsvcinfraEJB/imported_classes (class folder)
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - C:\Progra
WAS_50_PLUGINDIR/lib/appprofile.jar - C:\Program I
WAS_50_PLUGINDIR/lib/ecutils.jar - C:\Program Files
WAS_50_PLUGINDIR/lib/ejbcontainer.jar - C:\Program
WAS_50_PLUGINDIR/lib/ejbportable.jar - C:\Program
WAS_50_PLUGINDIR/lib/ivjejb35.jar - C:\Program Fil
WAS_50_PLUGINDIR/lib/j2ee.jar - C:\Program Files\I
WAS_50_PLUGINDIR/lib/pm.jar - C:\Program Files\IB
WAS_50_PLUGINDIR/lib/pmimpl.jar - C:\Program File
WAS_50_PLUGINDIR/lib/querymd.jar - C:\Program Fi
WAS_50_PLUGINDIR/lib/ras.jar - C:\Program Files\IB
WAS_50_PLUGINDIR/lib/rsadaptercci.jar - C:\Program
WAS_50_PLUGINDIR/lib/rsadapterspi.jar - C:\Program
WAS_50_PLUGINDIR/lib/rsaexternal.jar - C:\Program
```

Default output folder: bttsvcinfraEJB/ejbModule

Figure 8-101 Add bttbase.jar, bttjdbjsvc.jar, and bttjdbtsvc.jar into build path

5. Click **OK**.

Generating deployment and RMIC code for bttsvcinfraEJB project

To generate the deployment and RMIC code, follow these steps:

1. In the Project Navigator view, right-click the **bttsvcinfraEJB** project and select **Generate → Deployment and RMIC Code**.
2. In the pop-up dialog window, click **Select all**, and then **Finish**.

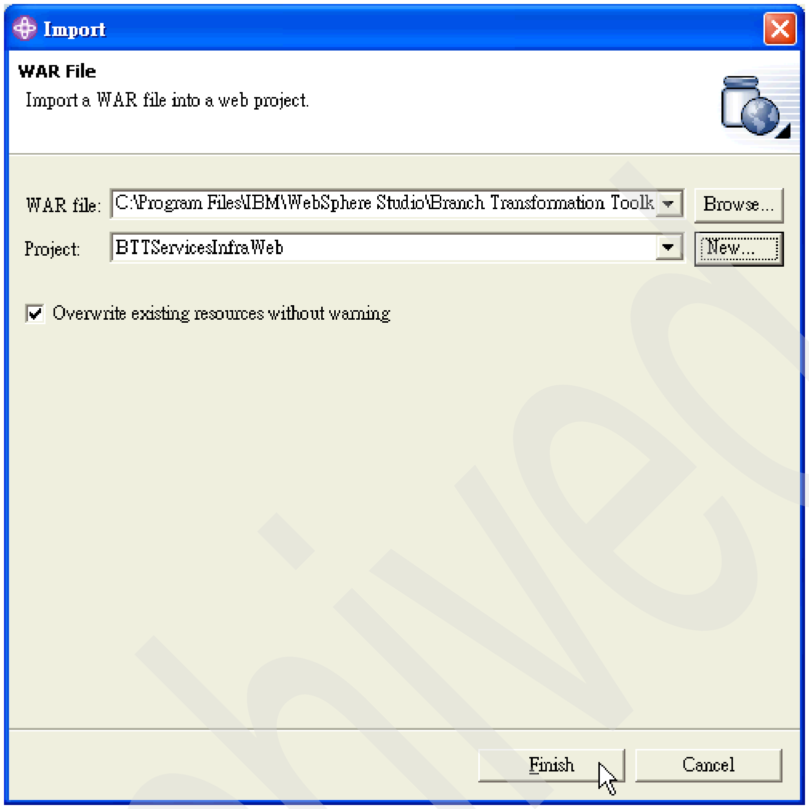
Importing service Web project BTTServicesInfraWeb.war

To import the service Web project, follow these steps:

1. Switch to the **J2EE Hierarchy** view.
2. Right-click **BTTBank** in the Enterprise Applications folder. Select **Import** → **Import Web Module**.
3. In the Import WAR file pop-up window, click **Browse** and navigate to **<BTT_install_dir>\jars** directory. Select **BTTServicesInfraWeb.war** and click **Open**.
4. In the same pop-up window, click **New** to create a new dynamic Web project, `BTTServicesInfraWeb`. Click **Finish**.

   > **Note:** After creating the project, you may see a pop-up dialog asking you to repair the server configuration. Click **OK** to add the new project to the BTTBank EAR project.

5. The WAR Import pop-up will look as shown in Figure 8-102.


Import

WAR File

Import a WAR file into a web project.

WAR file: C:\Program Files\IBM\WebSphere Studio\Branch Transformation Toolk

Project: BTTServicesInfraWeb

☑ Overwrite existing resources without warning

Figure 8-102 Import the BTTServicesInfraWeb Web module

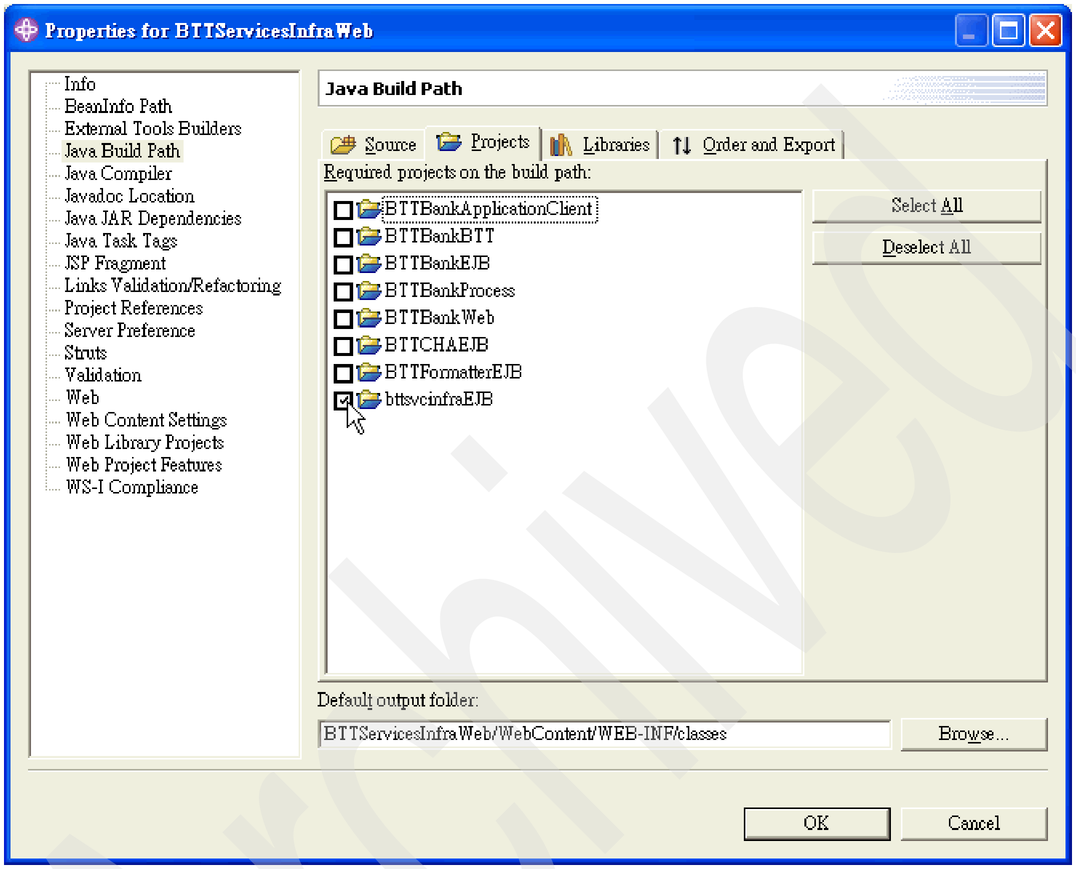
6. Click **Finish**.

Setting up project properties for BTTServicesInfraWeb

To set up the project properties, follow these steps:

1. Select the **Project Navigator** view tab. The BTTServicesInfraWeb Project appears.
2. Right-click the **BTTServicesInfraWeb** project, and select **Properties** from the context menu.
3. In the pop-up window, select **Java JAR Dependencies**. Ensure the **bttbase.jar**, **bttsvcinfra.jar** check box is selected in the JAR/Module list. Click **Apply**.
4. In the same pop-up window, select **Java Build Path**, and then click the **Libraries** tab. Click **Add JARs** to add the bttbase.jar from BTTBank project.

5. Click the **Projects** tab. Select **bttsvcinfraEJB** project, and then click **OK**, as shown in Figure 8-103.



Required projects on the build path:

- [ ] BTTBankApplicationClient
- [ ] BTTBankBTT
- [ ] BTTBankEJB
- [ ] BTTBankProcess
- [ ] BTTBankWeb
- [ ] BTTCHAEJB
- [ ] BTTFormatterEJB
- [✓] bttsvcinfraEJB

Default output folder: BTTServicesInfraWeb/WebContent/WEB-INF/classes

Figure 8-103 Add bttsvcinfraEJB project into build path

Constructing the Dummy Journal service

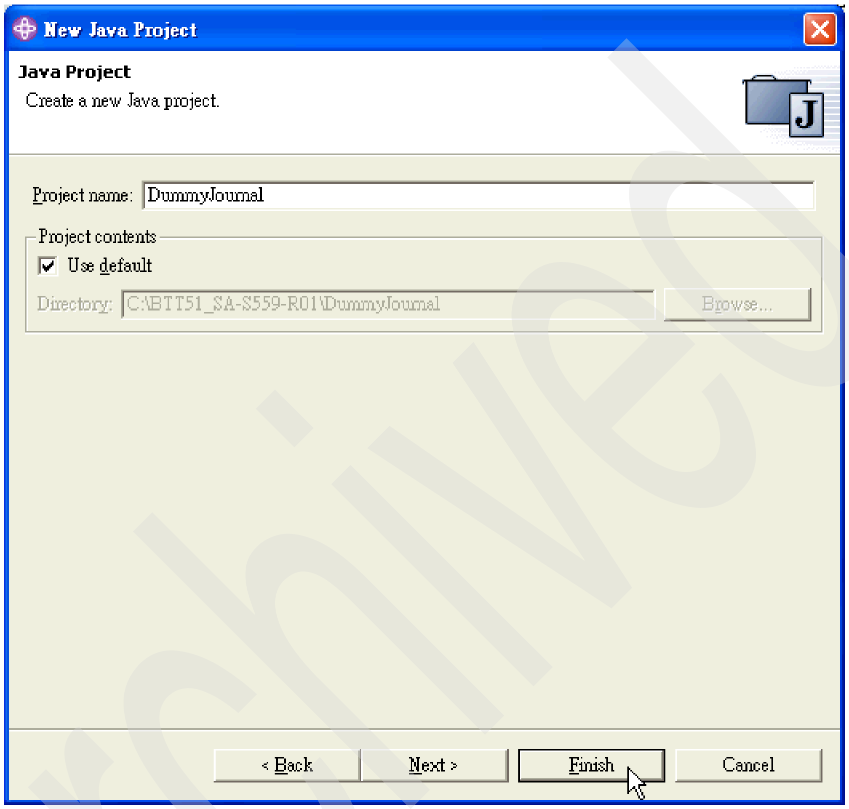
This section discusses creating the Java classes and properties files for the Dummy Journal service. A new Java project should be created to host the implementation of the service. This increases the reusability of the implementation.

Creating the DummyJournal Java project

To create the DummyJournal Java project, follow these steps:

1. From the WebSphere Studio Application Developer Integration Edition menu, select **File** → **New** → **Project**.

2. In the pop-up window, select **Java** from the left navigation panel, and then select **Java Project** from the right panel. Click **Next**.
3. Type `DummyJournal` in the **Project name** field. Click **Finish**. See Figure 8-104.


New Java Project

Java Project

Create a new Java project.

Project name: DummyJournal

Project contents

[✓] Use default

Directory: C:\BTT51_SA-S559-R01\DummyJournal

Figure 8-104 Create DummyJournal Java project

Setting up project properties for DummyJournal

To set up the project, follow these steps:

1. Select the **Project Navigator** view tab or the **Package Explorer** view tab if you switched to Java perspective. The DummyJournal project opens.
2. Right-click the **DummyJournal** project, and select **Properties** from the context menu.
3. In the pop-up window, select **Java Build Path**, and then click the **Libraries** tab. Click **Add JARs** to add five JARs, that is, **bttbase.jar**, **bttfmt.jar**, **bttsvcinfra.jar**, **bttjdbjsvc.jar**, and **bttjdbtsvc.jar** from BTTBank project.
4. Add `JavaSource` as the source folder. Select the **Source** tab, and click **Add Folder**. In the Source Folder Selection dialog, click **Create New Folder**. Type

`JavaSource` in the folder name field, and then click **OK**. Another pop-up dialog box opens, asking you whether you want to add DummyJournal/bin as the output folder. Click **OK**, as shown in Figure 8-105.

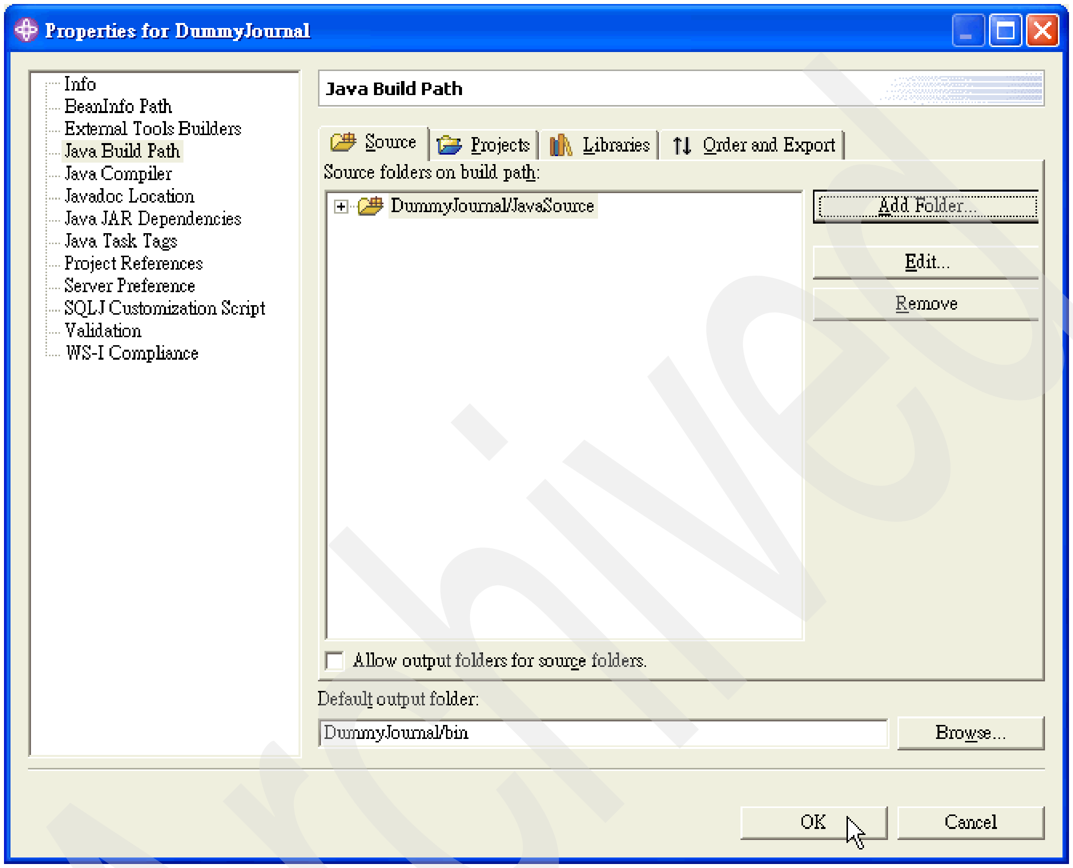


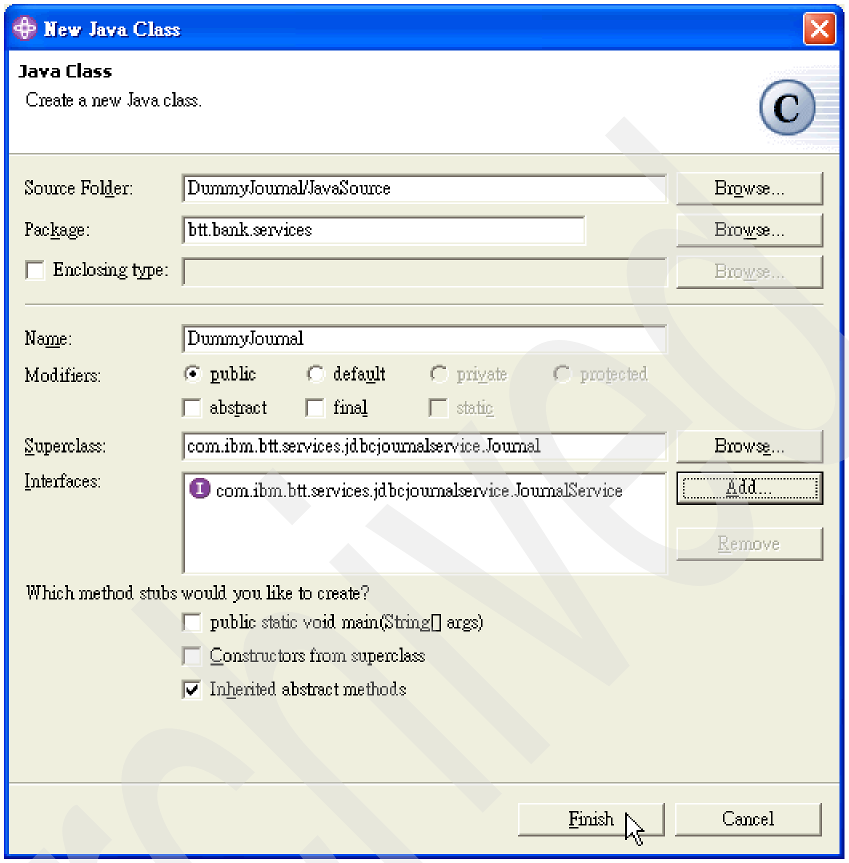
Figure 8-105 Add JavaSource as the project's source folder

Implementing DummyJournal service

To create the DummyJournal class, perform the following tasks:

1. Expand **DummyJournal** project, and right-click the **JavaSource** folder. Select **New** → **Package** from the context menu. Type `btt.bank.services` in the Name field of the New Java Package dialog. Click **Finish**.
2. Right-click the **btt.bank.services** package, select **New** → **Class** to create a Java class. Input the following properties to create the class, and click **Finish**:
   - Name: `DummyJournal`
   - Superclass: `com.ibm.btt.services.jdbcjournalservice.Journal`
   - Interfaces: `com.ibm.btt.services.jdbcjournalservice.JournalService`

See Figure 8-106.



Source Folder: DummyJournal/JavaSource
Package: btt.bank.services
☐ Enclosing type:
Name: DummyJournal
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: com.ibm.btt.services.jdbcjournalservice.Journal
Interfaces: com.ibm.btt.services.jdbcjournalservice.JournalService
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

Figure 8-106 Create the DummyJournal class

To implement the DummyJournal class, perform the following steps:

1. Add the import statements shown in Example 8-21.

Example 8-21 Additional import statements for DummyJournal.java

```
import javax.swing.JOptionPane;
import com.ibm.btt.base.Settings;
import com.ibm.btt.base.DSEObjectNotFoundException;
import com.ibm.btt.base.Tag;
import com.ibm.btt.base.TagAttribute;
```

2. Implement the addRecord(Context arg0, String arg1) method as shown in Example 8-22.

Example 8-22 Implemented addRecord(Context arg0, String arg1) method for DummyJournal.java

```
public int addRecord(Context arg0, String arg1)
      throws DSEInvalidArgumentException, DSEInvalidRequestException,
            DSEInternalErrorException, DSESQLException {
   FormatElement formatter = null;
   Hashtable dataHashtable = null;
   String show = null;
   try {
      show=(String) Settings.getSettings().getValueAt("showMessagesOnServer");
   } catch (DSEObjectNotFoundException e){
      show = "false";
   }
   try {
      formatter = new FormatElement();
      formatter.setName(arg1);
      dataHashtable = formatter.formatHashtable(arg0);
   } catch(Exception e) {
      if (show.equals("true")) {
         JOptionPane.showMessageDialog(null,
                                       e.toString(),
                                       "Exception",
                                       JOptionPane.ERROR_MESSAGE);
      }
   }
   if (dataHashtable != null) {
      if (show.equals("true")) {
         String message = "The 'addRecord' method is now executed\n" +
                          "The following data is added to the journal:\n\n" +
                          dataHashtable.toString()+"\n\n";
         JOptionPane.showMessageDialog(null,
                                       message,
                                       "Dummy journal service",
                                       JOptionPane.INFORMATION_MESSAGE);
      }
   }
   return 0;
}
```

3. Implement the updateRecord(int arg0, Context arg1, String arg2) as shown in Example 8-23.

Example 8-23 Implemented updateRecord(int arg0, Context arg1, String arg2) method for DummyJournal.java

```
public int updateRecord(int arg0, Context arg1, String arg2)
      throws
         DSEInvalidRequestException, DSEInvalidArgumentException,
         DSEInternalErrorException, DSESQLException {
```

```
    FormatElement formatter = null;
    Hashtable dataHashtable = null;
    String show = null;
    try {
       show=(String) Settings.getSettings().getValueAt("showMessagesOnServer");
    } catch (DSEObjectNotFoundException e) {
       show="false";
    }
    try {
       formatter = new FormatElement();
       formatter.setName(arg2);
       dataHashtable = formatter.formatHashtable(arg1);
    } catch(Exception e) {
       if (show.equals("true")) {
          JOptionPane.showMessageDialog(null,
                                        e.toString(),
                                        "Exception",
                                        JOptionPane.ERROR_MESSAGE);
       }
    }
    if (dataHashtable != null) {
       if (show.equals("true")) {
          String message = "to be updated with host reply data.\n" +
                           "The following data added to the journal:\n\n" +
                           dataHashtable.toString()+"\n\n";
          JOptionPane.showMessageDialog(null,
                                        message,
                                        "Dummy journal service",
                                        JOptionPane.INFORMATION_MESSAGE);
       }
    }
    return 0;
}
```

4. Save and close the **Java Editor**.

To create the DummyJournalImpl class, do the following:

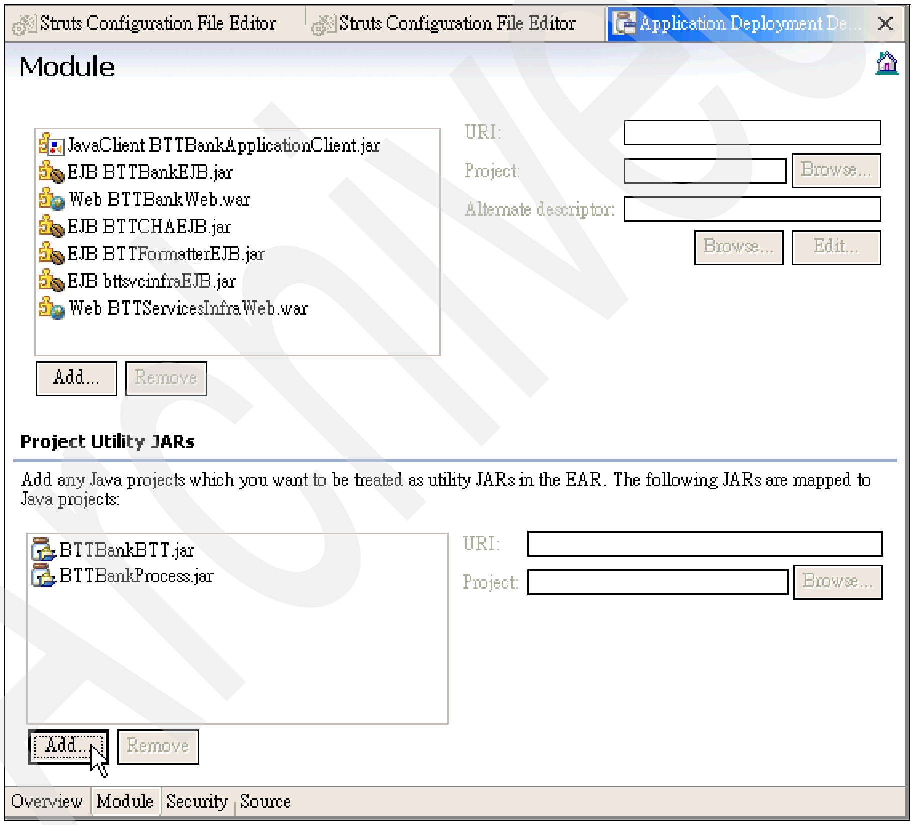
1. Right-click the **btt.bank.services** package, select **New** → **Class** to create a Java class.
2. Input the following properties to create the class:
   – Name: `DummyJournalImpl`
   – Superclass: `com.ibm.btt.services.jdbcjournalservice.JournalImpl`
3. Click **Finish**.

Note: Since it is a dummy journal service, do not put any code in DummyJournalImpl class.

Adding DummyJournal to BTTBank EAR project

To add the DummyJourna to BTTBank project, follow these steps:

1. Switch to the **J2EE** perspective, and expand the **BTTBank** project.
2. Double-click **EAR Deployment Descriptor**.
3. Click the **Module** tab.
4. Click **Add** in the Project Utility Jars section, as shown in Figure 8-107.



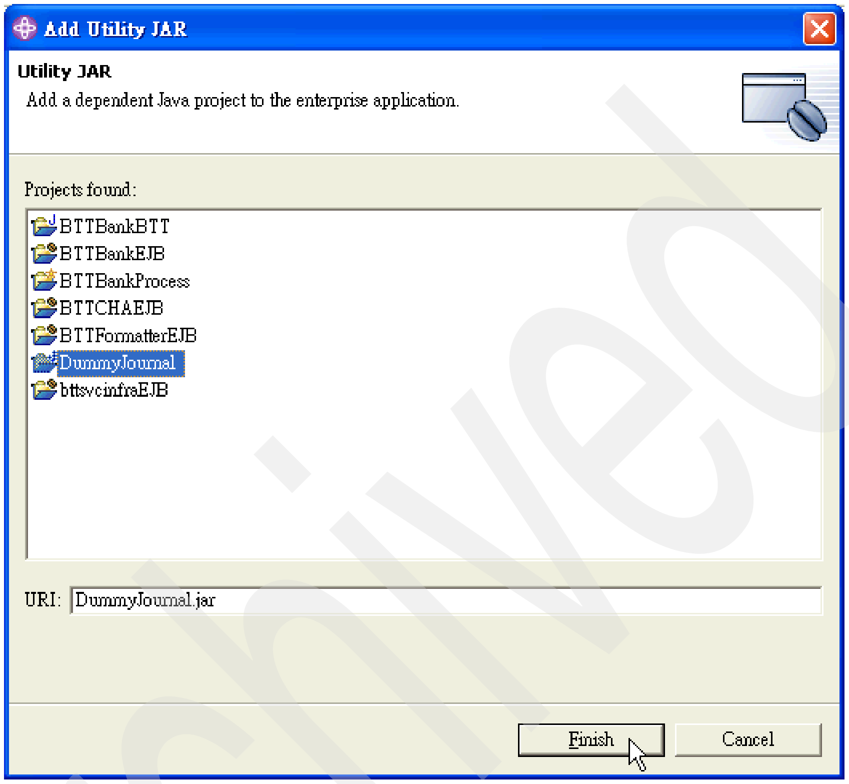
JavaClient BTTBankApplicationClient.jar
EJB BTTBankEJB.jar
Web BTTBankWeb.war
EJB BTTCHAEJB.jar
EJB BTTFormatterEJB.jar
EJB bttsvcinfraEJB.jar
Web BTTServicesInfraWeb.war

Add any Java projects which you want to be treated as utility JARs in the EAR. The following JARs are mapped to Java projects:

BTTBankBTT.jar
BTTBankProcess.jar

Figure 8-107 Click Add to add project utility JARs

5. In the pop-up Add Utility JAR dialog, select **DummyJournal**, and then click **Finish**, as shown in Figure 8-108.



Add a dependent Java project to the enterprise application.

Projects found:

- BTTBankBTT
- BTTBankEJB
- BTTBankProcess
- BTTCHAEJB
- BTTFormatterEJB
- DummyJournal
- bttsvcinfraEJB

URI: DummyJournal.jar

Figure 8-108 Select the DummyJournal project

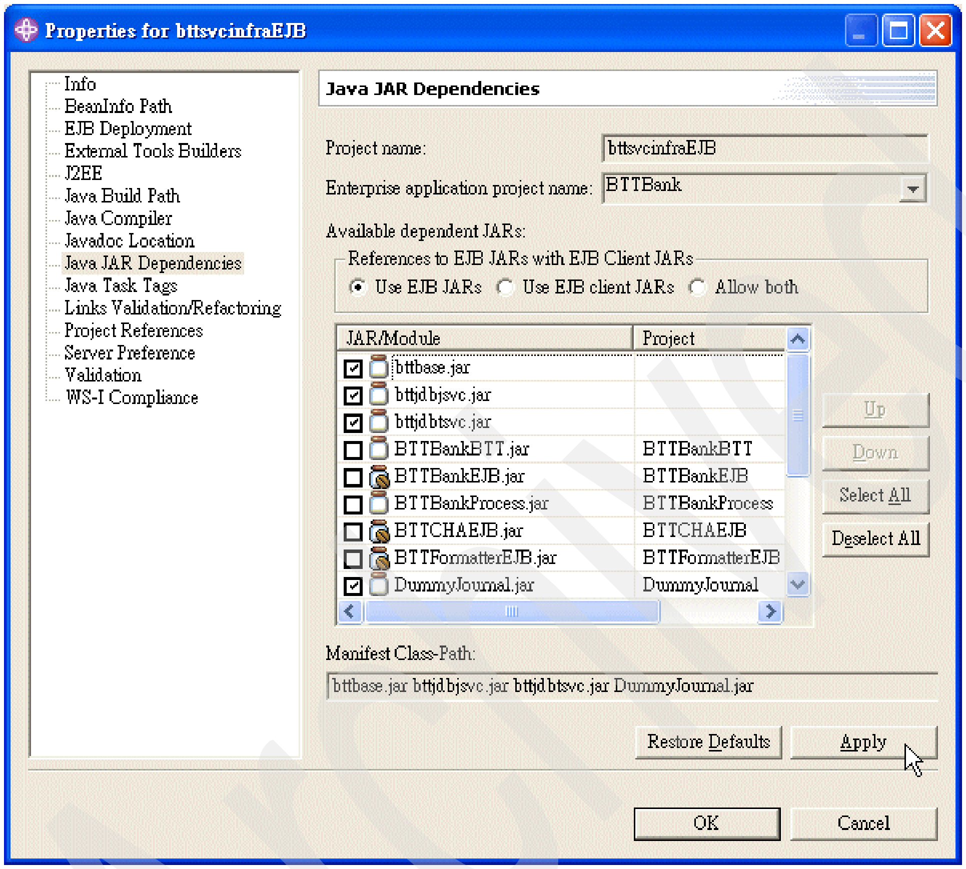
6. Click **OK** when the Repair Server Configuration dialog pops up.
7. Press Ctrl+S to save the change.

Adding DummyJournal to bttsvcinfraEJB EJB project

To add the DummyJournal to the bttvcinfraEJB project, follow these steps:

1. In the J2EE perspective, right-click the EJB module **bttsvcinfraEJB**. Select **Properties** from the context menu.

2. In the pop-up window, select **Java JAR Dependencies** from the left navigation panel, and check **DummyJournal.jar** in the right panel. Click **Apply**, as shown in Figure 8-109.


Properties for bttsvcinfraEJB

- Info
- BeanInfo Path
- EJB Deployment
- External Tools Builders
- J2EE
- Java Build Path
- Java Compiler
- Javadoc Location
- Java JAR Dependencies
- Java Task Tags
- Links Validation/Refactoring
- Project References
- Server Preference
- Validation
- WS-I Compliance

Java JAR Dependencies

Project name: bttsvcinfraEJB

Enterprise application project name: BTTBank

Available dependent JARs:

References to EJB JARs with EJB Client JARs: [✓ Use EJB JARs] Use EJB client JARs Allow both

	JAR/Module	Project
✓	bttbase.jar	
✓	bttjdbjsvc.jar	
✓	bttjdbtsvc.jar	
☐	BTTBankBTT.jar	BTTBankBTT
☐	BTTBankEJB.jar	BTTBankEJB
☐	BTTBankProcess.jar	BTTBankProcess
☐	BTTCHAEJB.jar	BTTCHAEJB
☐	BTTFormatterEJB.jar	BTTFormatterEJB
✓	DummyJournal.jar	DummyJournal

Manifest Class-Path:

bttbase.jar bttjdbjsvc.jar bttjdbtsvc.jar DummyJournal.jar

Figure 8-109 Add DummyJournal project JAR as the dependent JAR

3. Select **Java Build Path** from the left navigation panel, and select the **Projects** tab from the right panel. Select **DummyJournal** and then click **OK**.

Invoking the Dummy Journal service in the withdrawal operation

This section describes the process involved in modifying the execute method of the WithdrawalServerOpBean to utilize the dummy journal service. For the service mechanism to work correctly, you should take care of a series of properties files as well.

Configuring the BTTBankEJB project

To configure the BTTBankEJB project, follow these steps:

1. In the J2EE perspective, right-click the **BTTBankEJB** project. Select **Properties** from the context menu.
2. In the pop-up window, select **Java JAR Dependencies** in the left navigation panel, and check **bttjdbjsvc.jar**, **bttjdbtsvc.jar**, and **DummyJournal.jar** in the right panel. Click **Apply**.
3. Select **Java Build Path** in the left navigation panel, click the **Projects** tab, and check the **DummyJournal** project. Click the **Libraries** tab. Click **Add JARs** to add bttjdbjsvc.jar and bttjdbtsvc.jar from the BTTBank project.
4. Click **OK**.

Adding the Dummy Journal service to withdrawal Single Action EJB

To modify the withdrawal business Single Action EJB, do the following:

1. In BTTBankEJB project, open the **WithdrawalServerOpBean.java** file from the ejbModule/btt/bank/business/logic folder.
2. Add more import statements, as shown in Example 8-24.

Example 8-24 More import statements required by adding journal service

```
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

3. Modify the execute method as shown in bold in Example 8-25.

Example 8-25 Add journal service code into WithdrawalServerOp EJB

```
public Hashtable execute(BTTSystemData sysData, Hashtable reqData)
            throws Exception {
   Hashtable result = null;
   try {
      // initialize context hierarchy
      initialize(sysData);
      Context SAEContext = getContext();
      if (SAEContext.getParent() == null) {
         Context parent = Context.getContextByInstanceID(getInstanceId());
         SAEContext.chainTo(parent);
      }

      // set request data to withdrawalServerCtx
      SAEContext.setValueAt("BranchId", (String)reqData.get("BranchId"));
      SAEContext.setValueAt("AccountNumber",
             (String)reqData.get("AccountNumber"));
      SAEContext.setValueAt("Date", (Date)reqData.get("Date"));
      SAEContext.setValueAt("Amount", (Float)reqData.get("Amount"));
      String hostBuff = ((FormatElement)getFormat("withdrawalCSRequestFmt"))
```

```
                  .format(SAEContext);
      setValueAt("HostBuff", hostBuff);
      System.out.println("withdrawalServerOp SAE context:\n" +
            SAEContext.getKeyedCollection());

      // Add an electronic journal record
      Journal journal = (Journal)getService("JournalService");
      int recordNum = journal.addRecord(getContext(),"preSendJournalFmt");

      // set hard-coded response data to withdrawalServerCtx
      SAEContext.setValueAt("TrxReplyCode", "00");
      SAEContext.setValueAt("AccountBalance", "10000");
      SAEContext.setValueAt("TrxErrorMessage", "withdrawalOK");

      // Update the electronic journal record
      journal.updateRecord(recordNum, getContext(), "afterRecJournalFmt");
      journal.releaseServiceRequester();

      result = ((FormatElement)getFormat("afterRecJournalFmt"))
        .formatHashtable(SAEContext);
   }
   catch (Exception e) {
      e.printStackTrace();
      throw e;
   }
   return result;
}
```

4. Save and close the **Java File Editor**.

In order to let the withdrawal Single Action EJB invoke the Dummy Journal service properly, you should add at least three properties files in the ejbModule folder, as shown in Example 8-26, Example 8-27, and Example 8-28 on page 378.

Example 8-26 ServiceRequesterIDs.properties

```
# Property file for Service Requester Definition
# Definition is:
# ServiceRequesterID = ResourceBundle
JournalService=DummyJournal

# Include Invocation information
LocalJava=LocalJava
RemoteEJB=RemoteEJB
WSIFEJB=WSIFEJB
WSIFSoap=WSIFSoap
```

Example 8-27   DummyJournal.properties

```
# Property file for dummy journal requester
ServiceRequester=btt.bank.services.DummyJournal
ServiceType=Journal
CachingEnabled=false

# The value of service invocation has to match the id in
# ServiceRequesterIDs.properties
ServiceInvocation=RemoteEJB
```

Example 8-28   RemoteEJB.properties

```
# Property file for Remote EJB Invocation Type
ServiceRequester=com.ibm.btt.services.EJBInvocation

# Information of Remote EJB invocation
JndiName=ejb/com/ibm/btt/services/BTTServiceHolderEJBHome
ProviderURL=IIOP://localhost:2809/
```

The flowchart in Figure 8-110 shows the invocation steps of the three properties files.

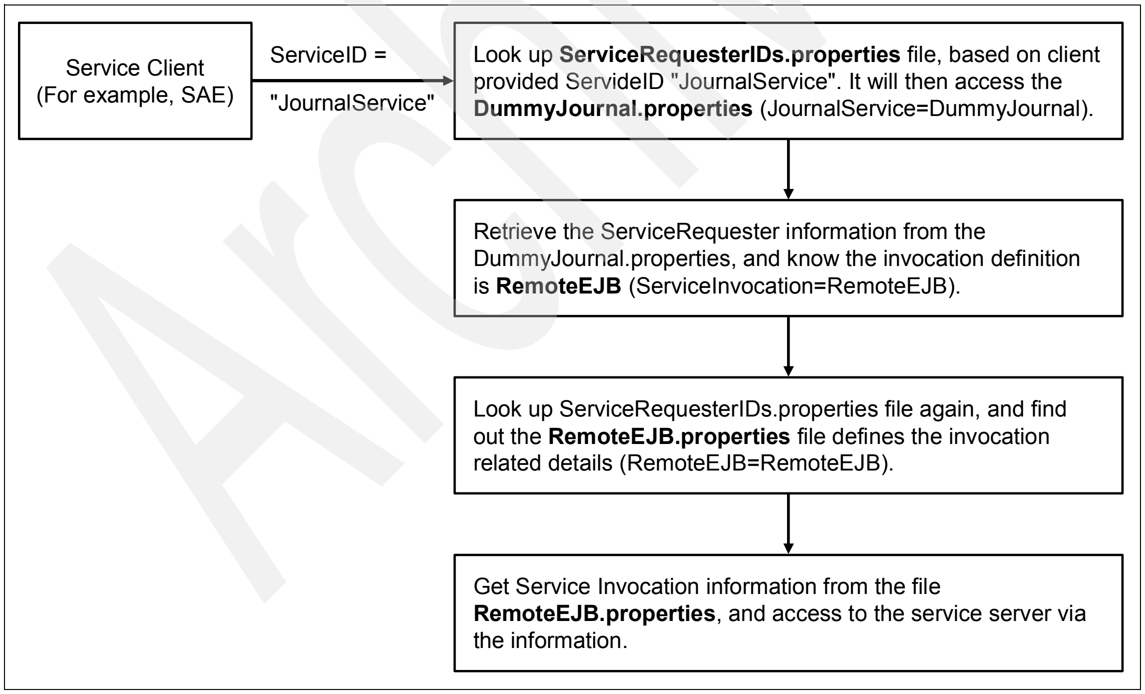

Service Client
(For example, SAE)
ServiceID =
"JournalService"
Look up **ServiceRequesterIDs.properties** file, based on client provided ServideID "JournalService". It will then access the **DummyJournal.properties** (JournalService=DummyJournal).
Retrieve the ServiceRequester information from the DummyJournal.properties, and know the invocation definition is **RemoteEJB** (ServiceInvocation=RemoteEJB).
Look up ServiceRequesterIDs.properties file again, and find out the **RemoteEJB.properties** file defines the invocation related details (RemoteEJB=RemoteEJB).
Get Service Invocation information from the file **RemoteEJB.properties**, and access to the service server via the information.

Figure 8-110   The service invocation process

Registering the Dummy Journal service

The enhancement is almost complete. The final step is to register the service into dse.ini. To do this, perform the following tasks:

1. Create the service definition file `dsesrvce.xml`.
   a. In J2EE perspective, click the **Project Navigator** view. Right-click the **business** folder in the BTTBankBTT project, and select **New** → **Other** from the context menu.
   b. In the New pop-up dialog box, select **Simple** in the left navigation panel, and select **File** in the right panel. Click **Next**.
   c. Type `dsesrvce.xml` in the File name field of the next window. Click **Finish**.
   d. The XML Editor would have started without content. Click the **Source** tab, copy and paste the snippet in Example 8-29 into the editor.
   e. Press Ctrl+S to save the change.
   f. Put this file in the c:\dse directory for runtime execution.

Example 8-29 The dsesrvce.xml file

```
<?xml version="1.0"?>
<dsesrvce.xml>
   <DummyDB2Journal autoCommit="false" id="Journal" schema="SCHEMA01">
      <column dataName="UserId" id="USERID"/>
      <column dataName="TID" id="TERMINALID"/>
      <column dataName="HostBuff" id="DATABUFFER"/>
   </DummyDB2Journal>
</dsesrvce.xml>
```

2. Change the settings in the dse.ini file, by doing the following:
   a. Double-click **BTTBankBusiness.chae** to start the CHA Editor. Click the **dse.ini** tab. Set field id=showMessagesOnServer's value property as `true` in the settings keyed collection. If the field is not there, add it as shown in Example 8-30.

Example 8-30 Adding setting in the dse.ini

```
<dse.ini>
   <kColl id="settings" dynamic="false" >
      ...
      <field id="showMessagesOnServer" value="true" description="" />
      ...
   </kColl>
</dse.ini>
```

b. Locate the services keyed collection in the tags keyed collection, add the Dummy Journal service as shown in bold in Figure 8-31.

Example 8-31 Dummy Jounal service

```
<dse.ini>
   <kColl id="settings" dynamic="false" >
      ...
      <kColl id="tags" dynamic="false" >
         ...
         <kColl id="services" dynamic="false" >
            ...
            <field id="DummyDB2Journal"
                   value="btt.bank.services.DummyJournalImpl"
                   description="compound" />
            ...
         </kColl>
         ...
      </kColl>
      ...
   </kColl>
</dse.ini>
```

c. Add **dsesrvce.xml** into the files keyed collection as shown in Example 8-32:

Example 8-32 Adding dsesrvce.xml

```
<dse.ini>
   <kColl id="settings" dynamic="false" >
      ...
      <!-- ====================================== -->
      <!-- Name of the generic definition files  -->
      <!-- ====================================== -->
      <kColl id="files">
         <field id="data" value="dsedata.xml"/>
         <field id="context" value="dsectxt.xml"/>
         <field id="type" value="dsetype.xml"/>
         <field id="format" value="dsefmts.xml"/>
         <field id="service" value="dsesrvce.xml"/>
      </kColl>
      ...
   </kColl>
</dse.ini>
```

d. Press Ctrl+S to save the change.

e. Copy **dse.ini** to c:\dse directory.

Verifying the working of Dummy Journal

To ensure that the server is ready and running, do the following:

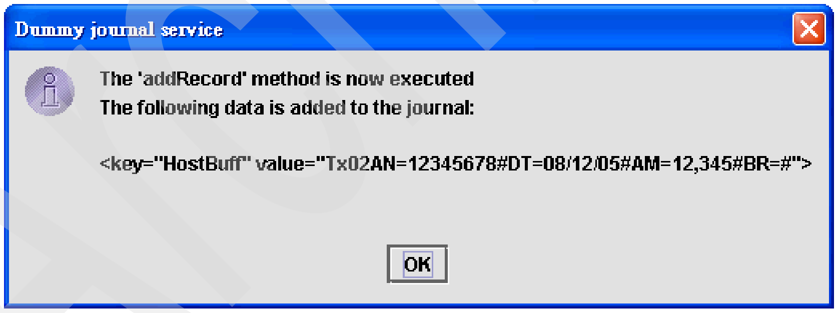
1. In Server perspective, right-click **WBI SF** either from Server Configuration view or Servers view.
2. Select **Start** from the context menu.

   In the console view, you will see a lot of messages popping up. This means that the server is running. When you see the message "*Server server1 open for e-business*", it means the server is started.

You can test the application using your Web browser:

1. Navigate to the sign-in page using the following URL:

   `http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/signin/prepareSignIn.do`
2. Sign in with the following values, and click **Submit**. The withdrawal page appears:
   - Use ID: `user01`
   - Password: `user01`
3. Input arbitrary Account Number and Amount, and then click **Withdrawal**.
4. After the submission, the browser will appear to be hanging. This is because your input is required at the server side. Switch to the Java console called Dummy journal service by pressing Alt+tab. A dialog box opens as shown in Figure 8-111.

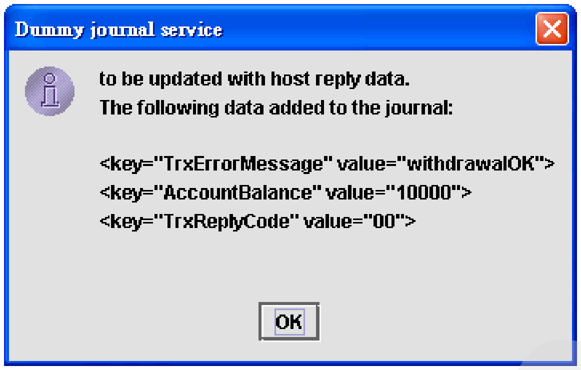


The 'addRecord' method is now executed
The following data is added to the journal:

<key="HostBuff" value="Tx02AN=12345678#DT=08/12/05#AM=12,345#BR=#">

Figure 8-111 Dummy Journal Service addRecord method show result

5. Click **OK**. Another dialog box opens, as shown in Figure 8-112.



```
to be updated with host reply data.
The following data added to the journal:

<key="TrxErrorMessage" value="withdrawalOK">
<key="AccountBalance" value="10000">
<key="TrxReplyCode" value="00">
```

Figure 8-112 Dummy Journal Service updateRecord method show result

6. Click **OK**. You will then see the browser showing the final result.

8.3.10 Using Graphical Builder and Business Process BTT Wizard

This section describes the process of building a deposit operation using BPEL, and demonstrates how to use the Graphical Builder and the Business Process BTT Wizard of the Branch Transformation Toolkit to achieve this goal. Due to the similarity between withdrawal and deposit options, the withdrawalServerCtx and withdrawal.gph Web diagram will be reused.

The Graphical Builder

The Graphical Builder is an Eclipse plug-in that helps construct multi-tiered BTT application. It presents an architectural view for application architects and developers. The Graphical Builder also helps to integrate disparate tools and centralizes the operations. The benefit of using Graphical Builder is that it streamlines the end-to-end development of the Branch Transformation Toolkit application.

Figure 8-113 shows the relationships between the Graphical Builder and various supporting tools.

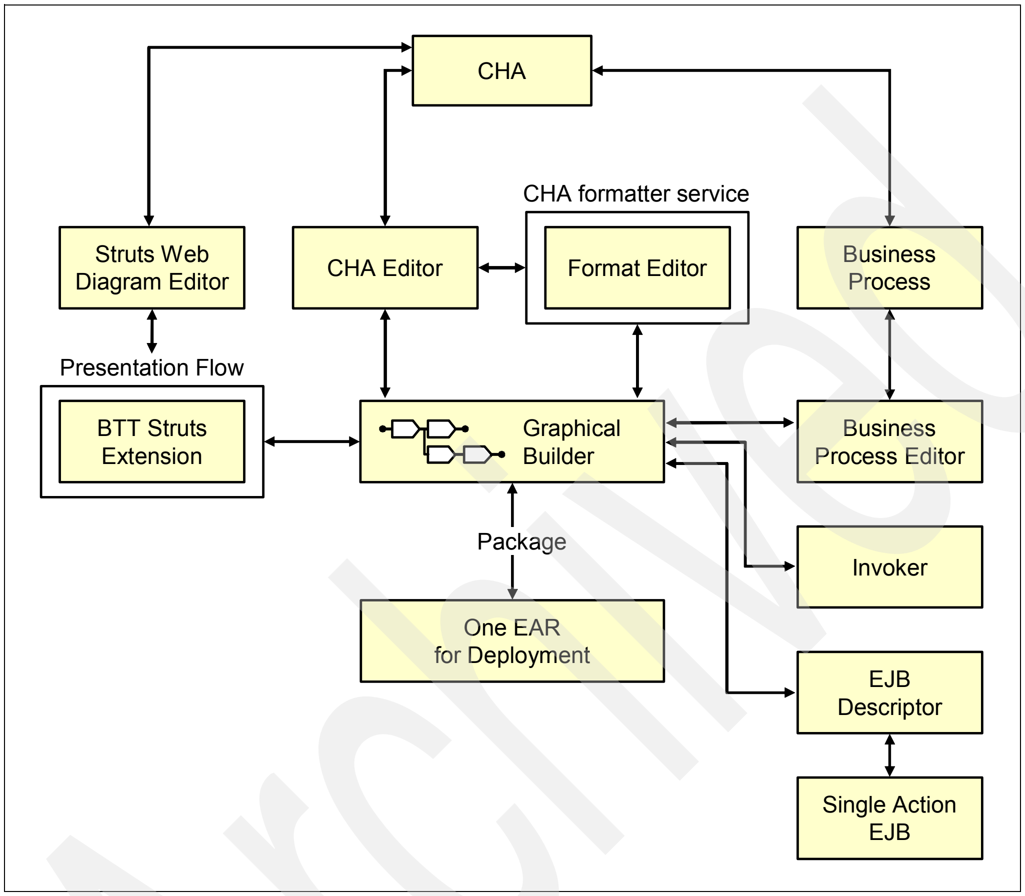


Figure 8-113 Relationships between the Graphical Builder and various supporting tools

The Graphical Builder consists of a number of views:

- Graphical Editor view is the primary view of the Graphical Builder. It consists of a set of tabbed pages in which you can edit values:
  - ETEE page: This page displays various application nodes such as business processes, Single Action EJBs, and so on. Additionally, the Editor panel has palette with a set of objects:
    - Select: A tool that helps select nodes.
    - Marquee: A tool that helps draw a marquee for selecting multiple nodes at a time.

    - Connection: A tool that helps draw linkages between various nodes.
    - CompositionNode drawer: This group contains tools to create presentation nodes, business process nodes, CHA context nodes, formatter nodes, and Single Action EJB nodes in the Editor panel.
  - Configuration page: This page displays the contents of the Graphical Builder's configuration file.
  - Graphical Builder file: This page displays the contents of the Graphical Builder definition file. It displays the Graphical Builder definitions in the order they are created. The tab displays the name of the file.
- Outline view lists all the application nodes in alphabetical order. You can expand each node to display its structure. For example, when you expand a business process node, you will see the CHA contexts and formatters associated with this node.
- Properties view displays the properties of the selected presentation node, business process node, format definition, CHA contexts, data definition, and so on.

Figure 8-114 shows the views of the Graphical Builder:

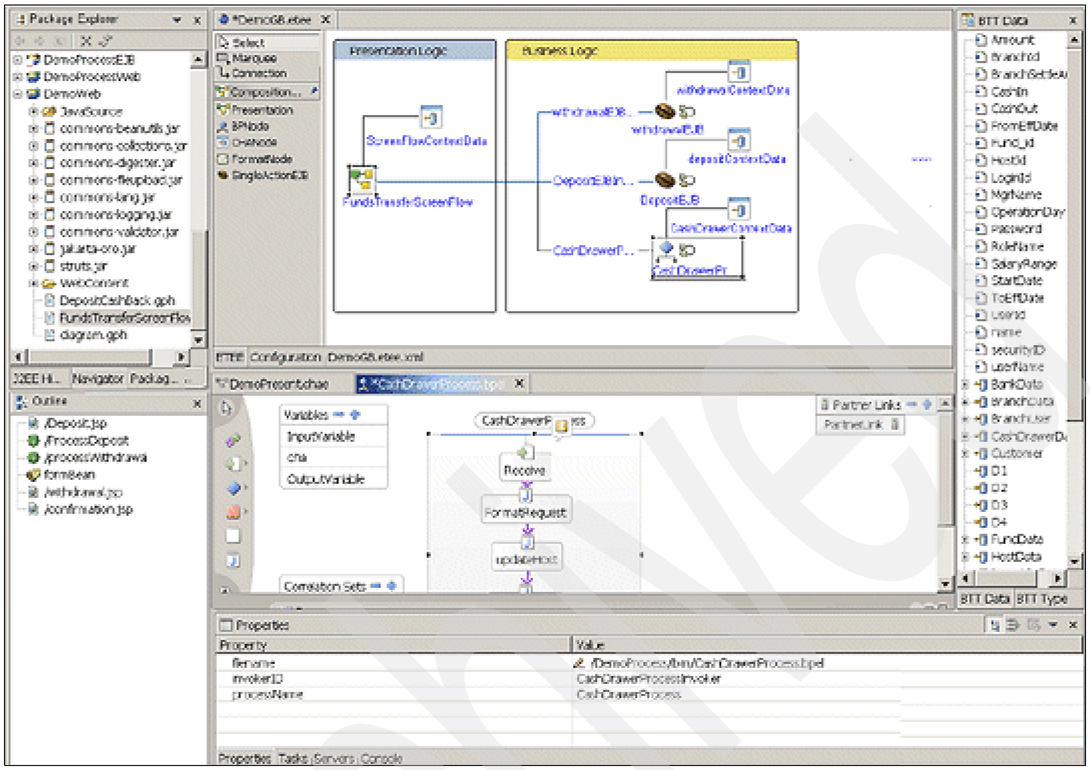


Figure 8-114 Various views related to the Graphical Builder

Reverse engineering the Withdrawal operation

This chapter demonstrates the bottom-up approach to constructing a Branch Transformation Toolkit application. With the Graphical Builder, you can reverse engineer the withdrawal operation in the Graphical Editor view. To do this, perform the following tasks:

1. Start the WebSphere Studio Application Developer Integration Edition.
2. Open the **BTTBankBTT** project, copy all the files from the business folder to the presentation folder. Overwrite the existing files by clicking **Yes To All** in the pop-up dialog box.
3. Select **File** → **New** → **Other** from the main menu.

4. In the pop-up dialog box, select **IBM Branch Transformation Toolkit** in the left navigation panel, and **Graphical Builder Design File** in the right panel, as shown in Figure 8-115. Click **Next**.

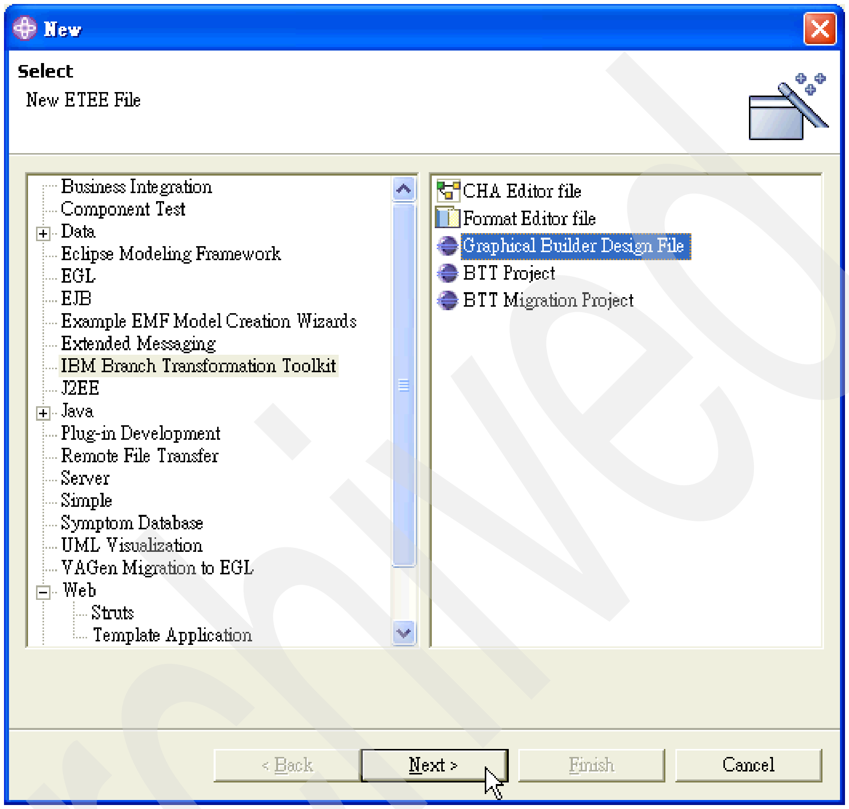


Figure 8-115 Select Graphical Builder design file

5. In the Create new etee file dialog box, select **BTTBankBTT** as the BTT Application project, and type `BankBasicOp` in the Graphical Builder File field. Accept other default values and click **Finish**.

6. The Graphical Editor and other supporting views appear as shown in Figure 8-116.

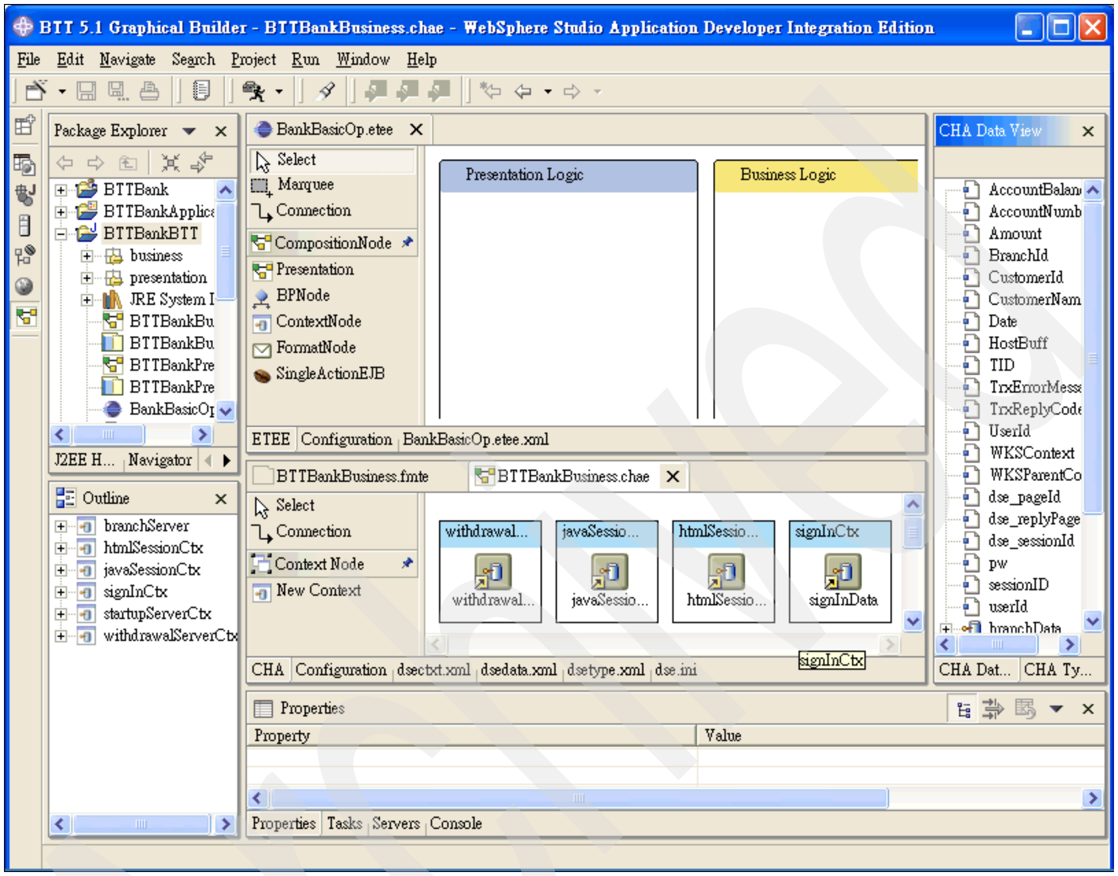


Figure 8-116 The BankBasicOp.etee opened within the Graphical Editor

7. Select the **Presentation** icon (Presentation) in the CompositionNode drawer. Drop it in the Presentation Logic block, as shown in Figure 8-117.

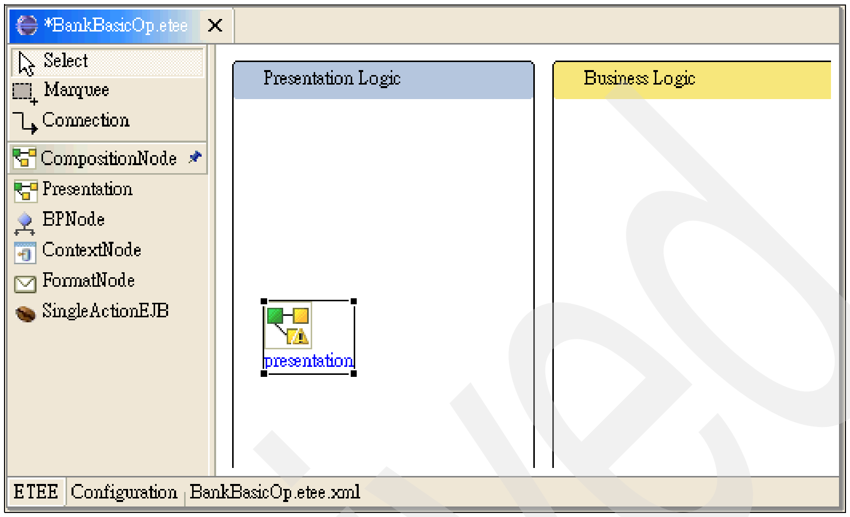


Figure 8-117 Add a presentation component in the Graphical Editor

8. Right-click the **Presentation** icon, and select **Open Properties Dialog** from the context menu. The Presentation Node Dialog appears, as shown in Figure 8-118.

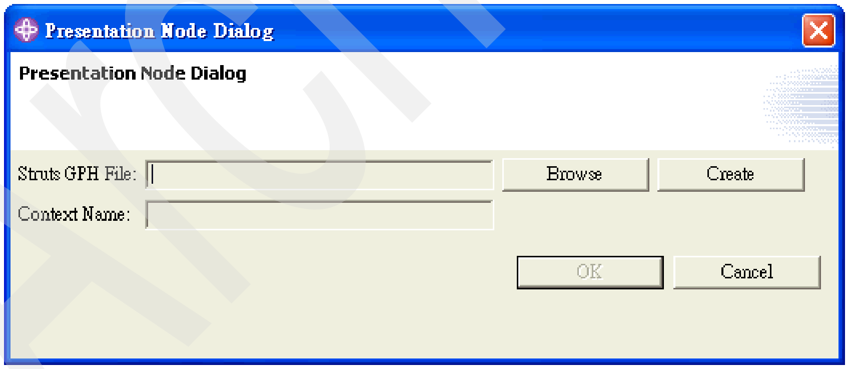


Figure 8-118 The presentation node dialog

9. Click **Browse**, select **withdrawal.gph** in BTTBankWeb project. Click **OK**, as shown in Figure 8-119 on page 389. You will see that the Struts GPH File and

the Context Name field are populated with the correct values, as shown in Figure 8-120. Click **OK**.

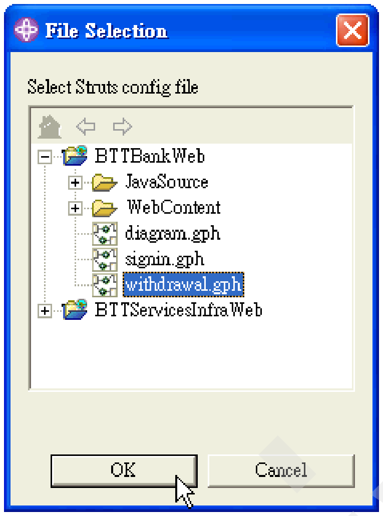
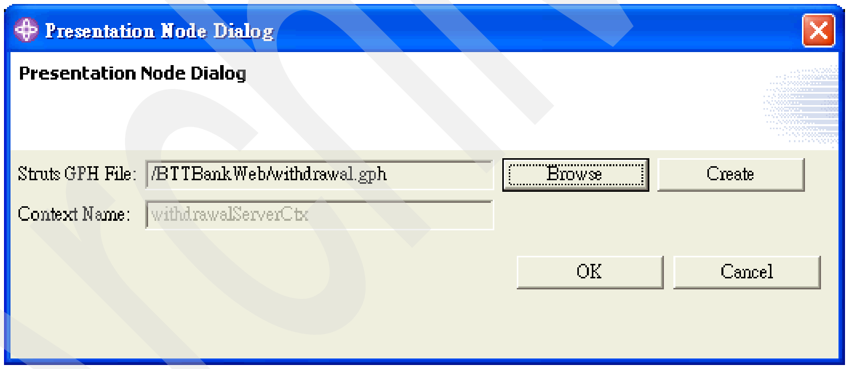


Figure 8-119 Select withdrawal.gph



Struts GPH File: /BTTBankWeb/withdrawal.gph

Context Name: withdrawalServerCtx

Figure 8-120 The Presentation Node Dialog with correct values filled

10. After the dialog box closes, you can see the presentation node renamed as `withdrawal`, and the withdrawalServerCtx associated with it, as shown in Figure 8-121.

Note: If you look closely at the diagram, you will find that there is an exclamation mark in the withdrawal node. You will see the warning message "*1: EJBAction '/withdraw' needs to be implemented*", when you move your mouse over the exclamation mark. This will be fixed automatically, once the peer Business Logic component is set up.

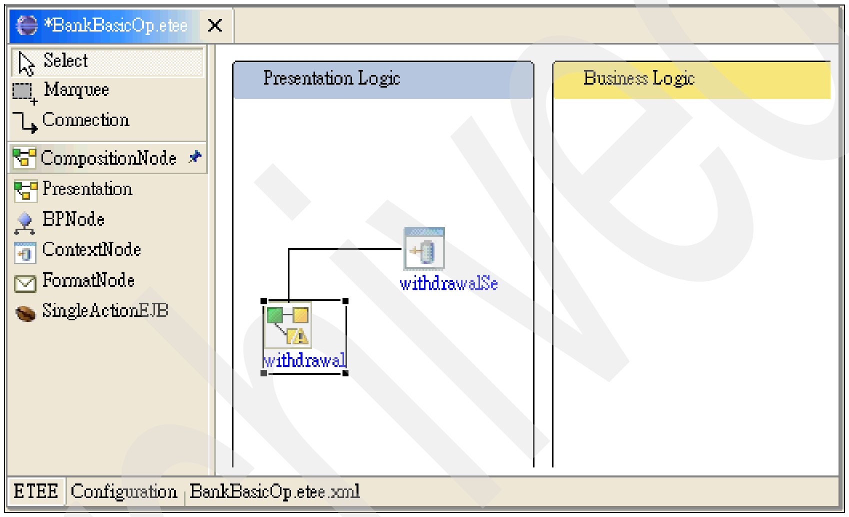


Figure 8-121 The withdrawal presentation node and withdrawalServerCtx context appear

11. Select the **SingleActionEJB** icon (SingleActionEJB) from the CompositionNode drawer. Drop it in the Business Logic block, as shown in Figure 8-122.

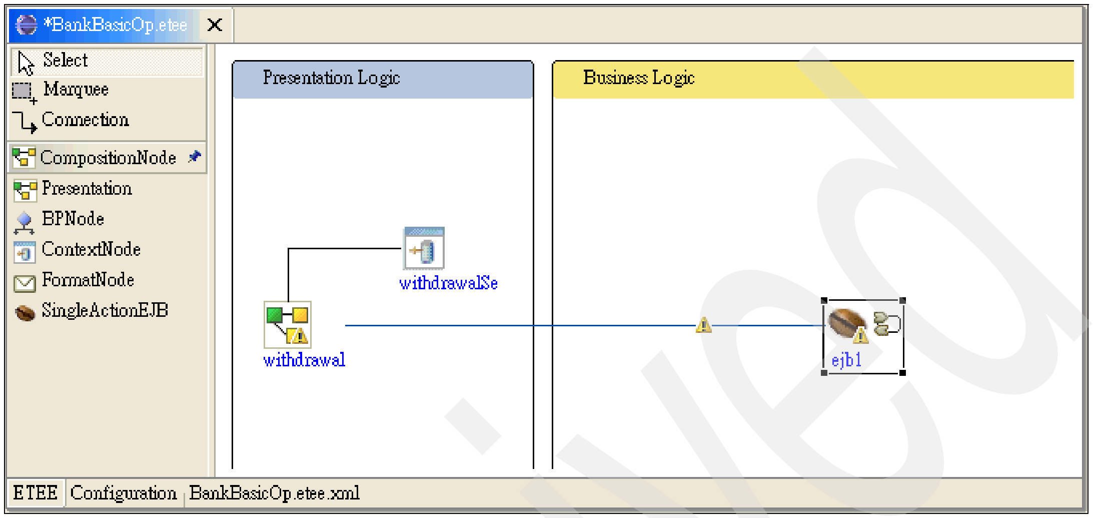
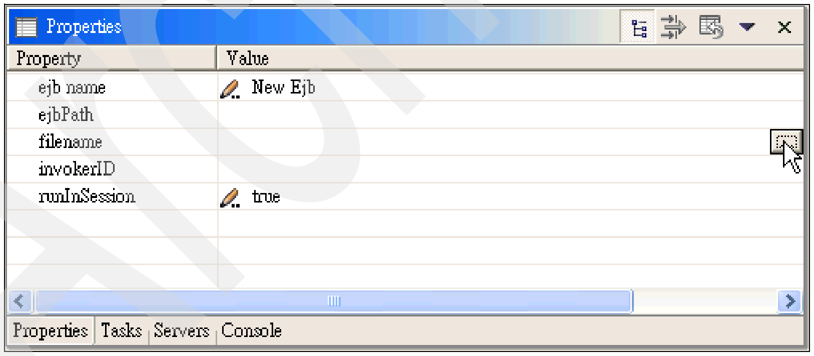


Figure 8-122 A new EJB node added into the Graphical Editor

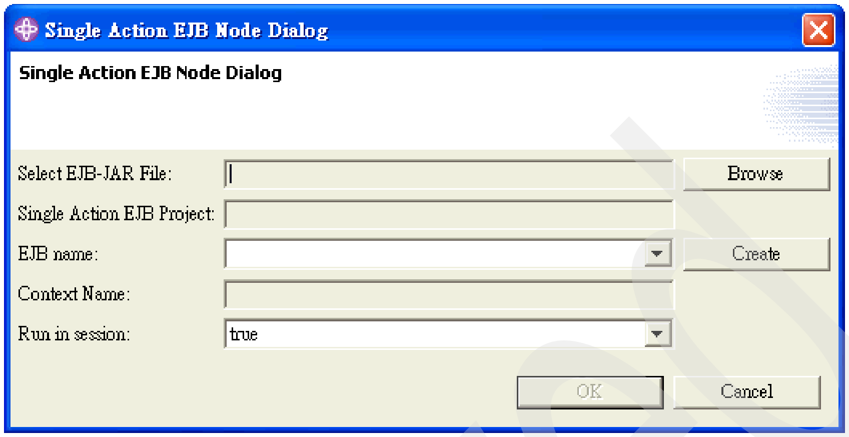
12. Click the **ejb1** node. In the Properties view, click **Property Editor** against the filename field as shown in Figure 8-123.



Property	Value
ejb name	New Ejb
ejbPath	
filename	
invokerID	
runInSession	true

Figure 8-123 Select ejb-jar.xml by clicking Property Editor

13. The Single Action EJB Node Dialog appears, as shown in Figure 8-124.


Single Action EJB Node Dialog

Single Action EJB Node Dialog

Select EJB-JAR File:

Single Action EJB Project:

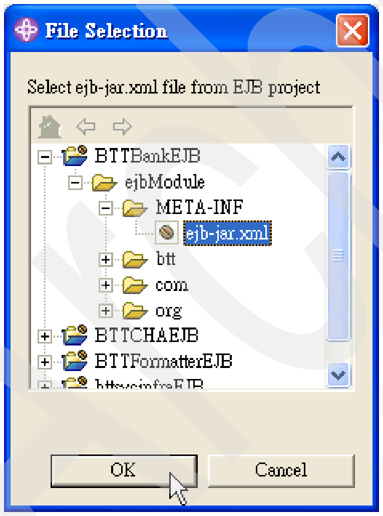
EJB name:

Context Name:

Run in session: true

Figure 8-124 The Single Action EJB Node Dialog

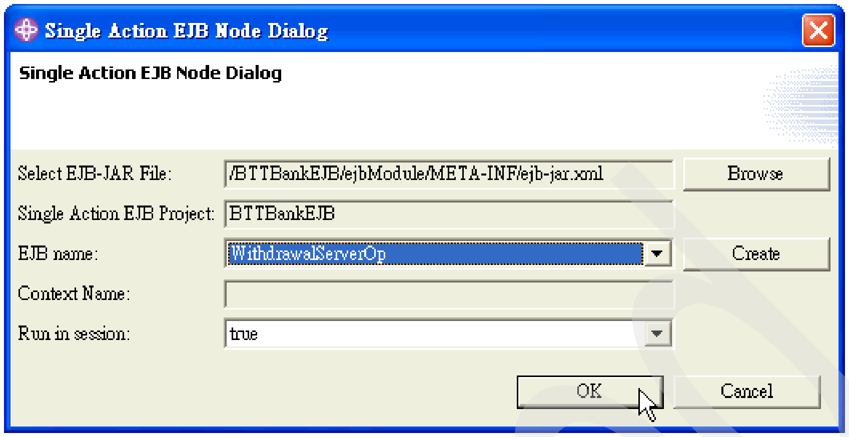
14. Click **Browse**, select the **ejb-jar.xml** file in the ejbModule/META-INF folder of the BTTBankEJB project, as shown in Figure 8-125. Click **OK**.


File Selection

Select ejb-jar.xml file from EJB project

Figure 8-125 Select ejb-jar.xml file

15. Go back to the Single Action EJB Node dialog box, select **WithdrawalServerOp** against the EJB name field. Click **OK**. See Figure 8-126 on page 393.


Single Action EJB Node Dialog

Single Action EJB Node Dialog

Select EJB-JAR File: /BTTBankEJB/ejbModule/META-INF/ejb-jar.xml

Single Action EJB Project: BTTBankEJB

EJB name: WithdrawalServerOp

Context Name:

Run in session: true

Figure 8-126 Select WithdrawalServerOp EJB

16. After the dialog box closes, you can see that the ejb1 node is renamed as `WithdrawalServerOp`, as shown in Figure 8-127.

> **Note:** At this time, there should be a withdrawalServerCtx context node associated with the EJB node. If it does not appear, you should add it manually.

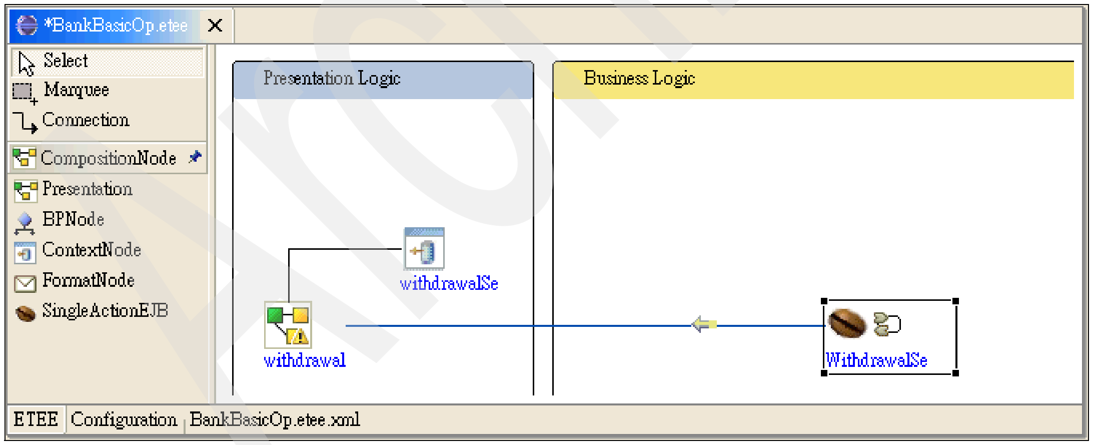


Figure 8-127 The ejb1 node renamed as WithdrawalServerOp

17. Ensure that the BTTBankBusiness.chae is opened with the Graphical Editor. Drag the **withdrawalServerCtx** context from the CHA Editor and drop it into the WithdrawalServerOp EJB node, as shown in Figure 8-128.

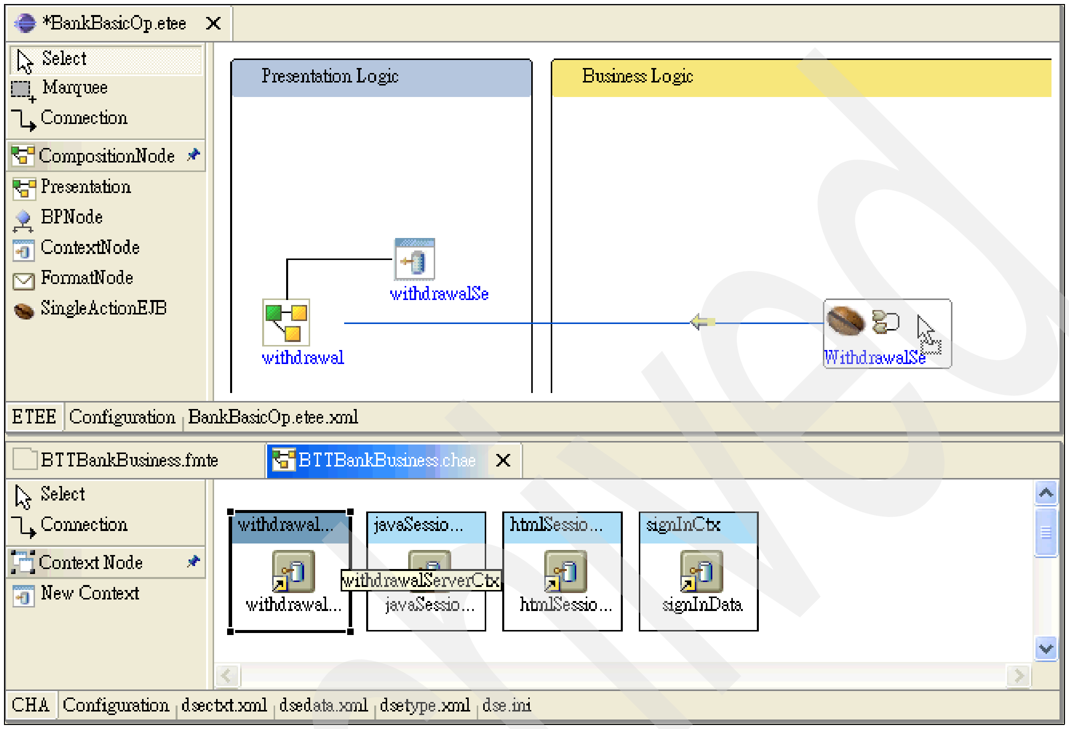


Figure 8-128 Drag and drop the withdrawalServerCtx to WithdrawalServerOp EJB node

After this operation, you can see that withdrawalServerCtx appears and associates with the WithdrawalServerOp EJB node, as shown in Figure 8-129.

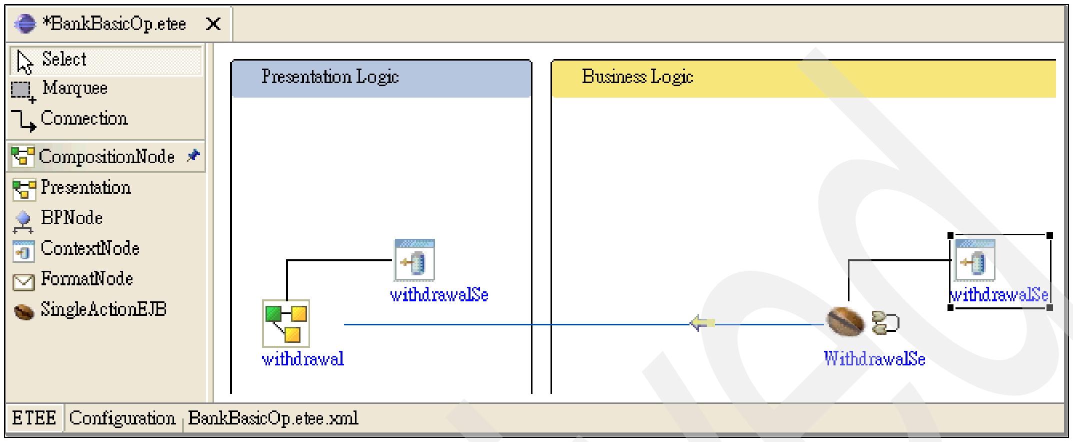
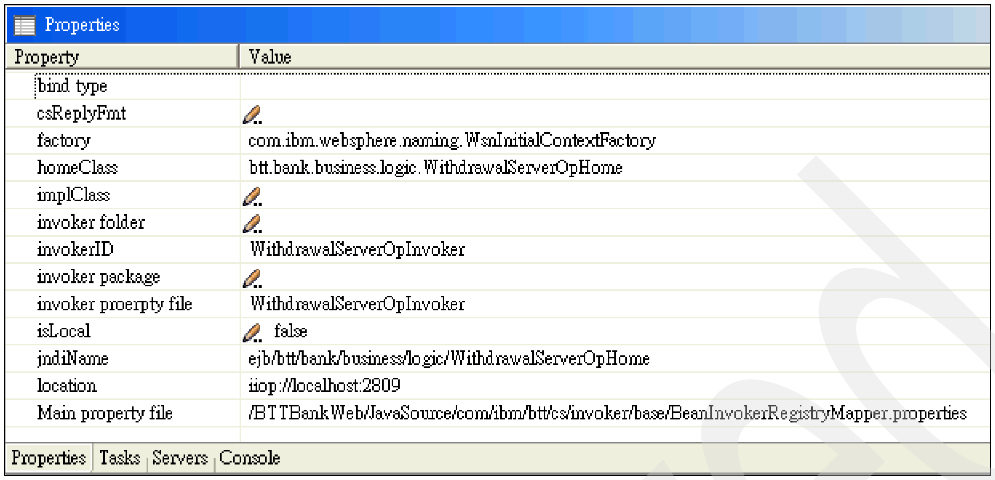


Figure 8-129 A withdrawalServerCtx context node associated with theWithdrawalServerOp EJB node

18. One more task is required to make the diagram perfect. Click the **line** between the withdrawal presentation node and the WithdrawalServerOp EJB node. If you find some settings missing in the Properties view, as shown in Figure 8-130 on page 396, use the following values to correct the problem:

   – implClass: `WithdrawalInvoker`
   – invoker folder: `/BTTBankWeb/JavaSource/btt/invoker/struts`
   – invoker package: `btt.invoker.struts`



Properties

Property	Value
bind type	
csReplyFmt	
factory	com.ibm.websphere.naming.WsnInitialContextFactory
homeClass	btt.bank.business.logic.WithdrawalServerOpHome
implClass	
invoker folder	
invokerID	WithdrawalServerOpInvoker
invoker package	
invoker proerpty file	WithdrawalServerOpInvoker
isLocal	false
jndiName	ejb/btt/bank/business/logic/WithdrawalServerOpHome
location	iiop://localhost:2809
Main property file	/BTTBankWeb/JavaSource/com/ibm/btt/cs/invoker/base/BeanInvokerRegistryMapper.properties

Figure 8-130 Some invokers' properties setting missing

After providing the values, the diagram will look as shown in Figure 8-131.

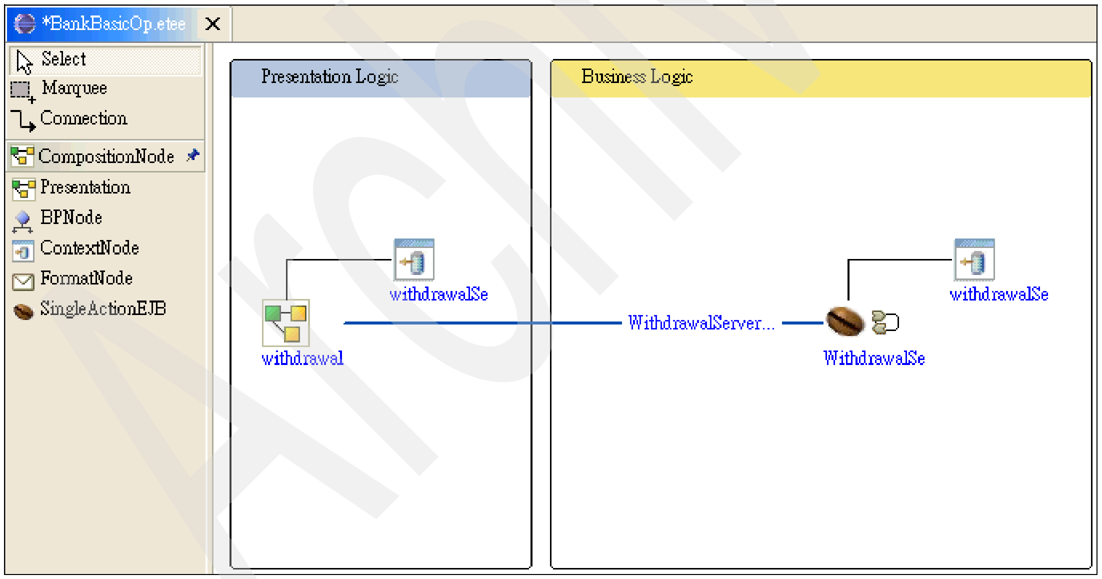


Figure 8-131 The fully reversed engineered Withdrawal operation

19. Press Ctrl+S to save your work.

Constructing the Deposit operation

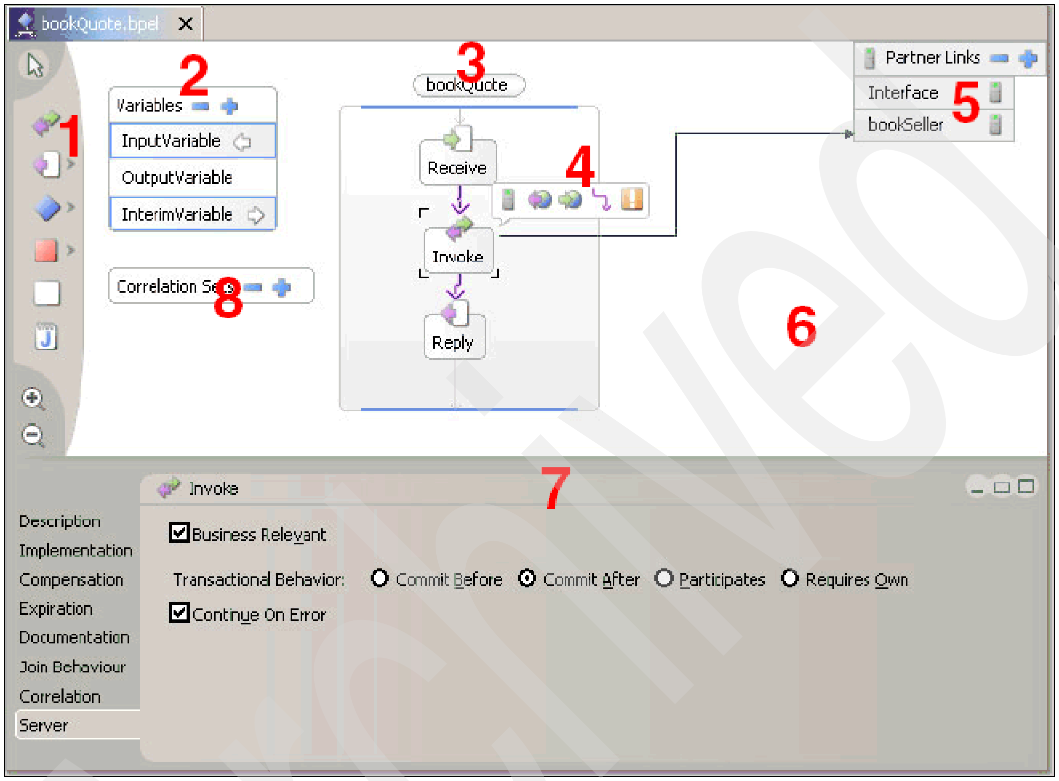
This section discusses the construction of the Deposit operation. As discussed in 8.3.1, “Development paths” on page 248, the top-down approach should be used to build the operation. The deposit operation uses BPEL, a flexible, standards-based approach for defining and executing the business processes, as its implementation.

Start from the Graphical Builder, utilize the BPEL Editor, a development tool that comes with WebSphere Studio Application Developer Integration Edition 5.1.1, and Business Process BTT Wizard, a tool that belongs to Branch Transformation Toolkit 5.1, to complete the business logic. Struts tools should be used to add the presentation logic of the deposit operation.

The tools you should use are described below.

The BPEL Editor

The BPEL Editor, also referred to as the BPEL Process Editor, is a graphical programming environment that is used to create and visually manipulate business processes. Figure 8-132 shows the BPEL Editor.



☑ Business Relevant

Transactional Behavior: ○ Commit Before [✓ ● Commit After] ○ Participates ○ Requires Own

☑ Continue On Error

Figure 8-132 The BPEL Editor

Processes are constructed in the process area (3) of the canvas (6), by dragging activities from the palette (1). Definitions of variables, partner links, and correlation sets are held in separate areas (2, 5, 8) on the canvas. Selecting any activity brings up the action bar (4), which contains a series of icons related to the activity, including adding Fault Handlers. The details area (7) below the canvas provides the means for configuring the currently selected activity.

For more information about BPEL, refer to the IBM Redbook *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318.

The Business Process BTT Wizard

The Business Process BTT Wizard provides a graphical user interface (GUI) to help you extend your business processes to take advantage of the BTT Abstract Layer. The Abstract Layer provides mapping between messages and process contexts, enables the business processes to access other toolkit components such as CHA. Refer to the online help for more information about the BTT Abstract Layer.

The Business Process BTT Wizard helps you customize the business process code or the BPEL files, from the following aspects:

- Specifying the CHA context associated with the business process.
- Specifying the process type, that is, general process, login process, or logout process.
- Specifying the mapping relationship between CHA contexts and process results.
- Specifying external snippet classes for the business process.
- Enabling conditional navigation based on snippet results.
- Adding the variables in the process to the BPEL file and adding the associated message definition of the variables to the WSDL file.

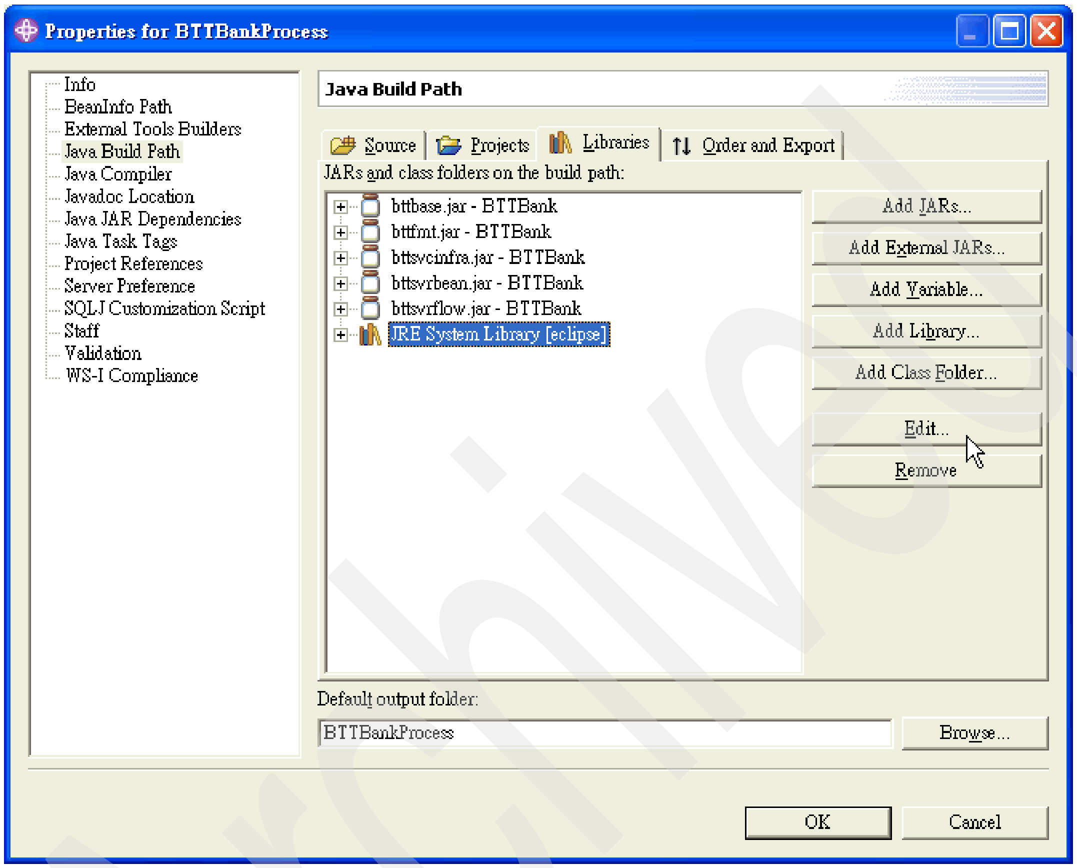
In short, the tool saves you from modifying the BPEL files directly, thus increasing your productivity while developing a Branch Transformation Toolkit application.

Setting up the BPEL project

To set up the project, follow these steps:

1. Switch to the **Project Navigator** view of the J2EE perspective, right-click **BTTBankProcess** project and select **Properties** from the context menu.
2. In the pop-up window, select **Java Build Path**, and then click the **Libraries** tab.

3. Select **JRE System Library [eclipse]**, then click **Edit**, as shown in Figure 8-133.


Properties for BTTBankProcess

Info
BeanInfo Path
External Tools Builders
Java Build Path
Java Compiler
Javadoc Location
Java JAR Dependencies
Java Task Tags
Project References
Server Preference
SQLJ Customization Script
Staff
Validation
WS-I Compliance

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

bttbase.jar - BTTBank
bttfmt.jar - BTTBank
bttsvcinfra.jar - BTTBank
bttsvrbean.jar - BTTBank
bttsvrflow.jar - BTTBank
JRE System Library [eclipse]

Default output folder:
BTTBankProcess

Figure 8-133 Change JRE System Library for BTTBankProcess project

4. In the Edit Library pop-up dialog box, select **WebSphere V5.1 EE JRE**, and click **Finish**, as shown in Figure 8-134.

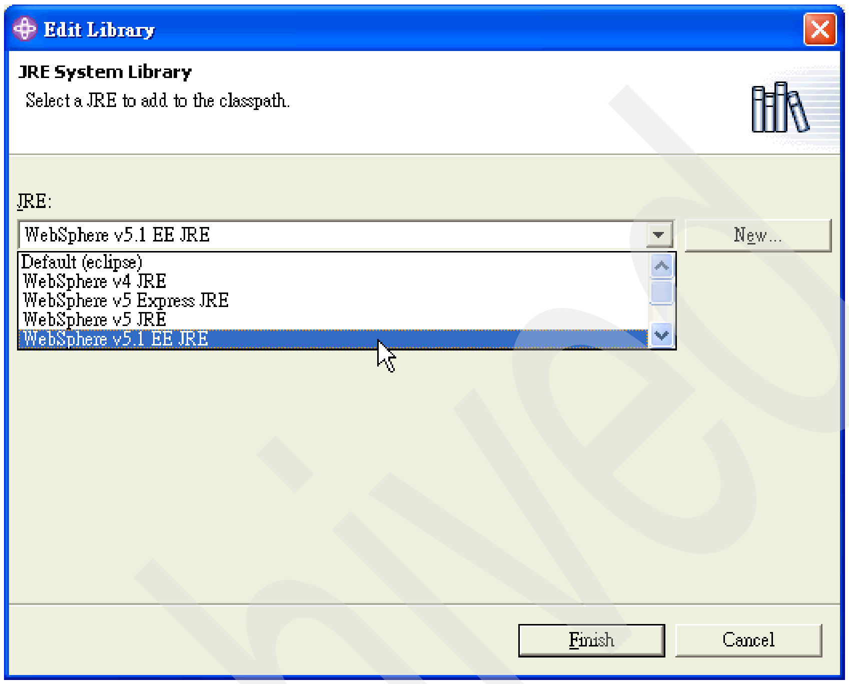


Figure 8-134 Select WebSphere V5.1 EE JRE

5. Click **OK** in the project properties window.

Using the Graphical Builder

To work with the Graphical Builder, follow these steps:

1. Switch to the **Graphical Builder** perspective, open the **BankBasicOp.etee** file in the BTTBankBTT project.

2. Select the **BPNode** icon (BPNode) from the CompositionNode drawer, and drop it into the Business Logic block, as shown in Figure 8-135.

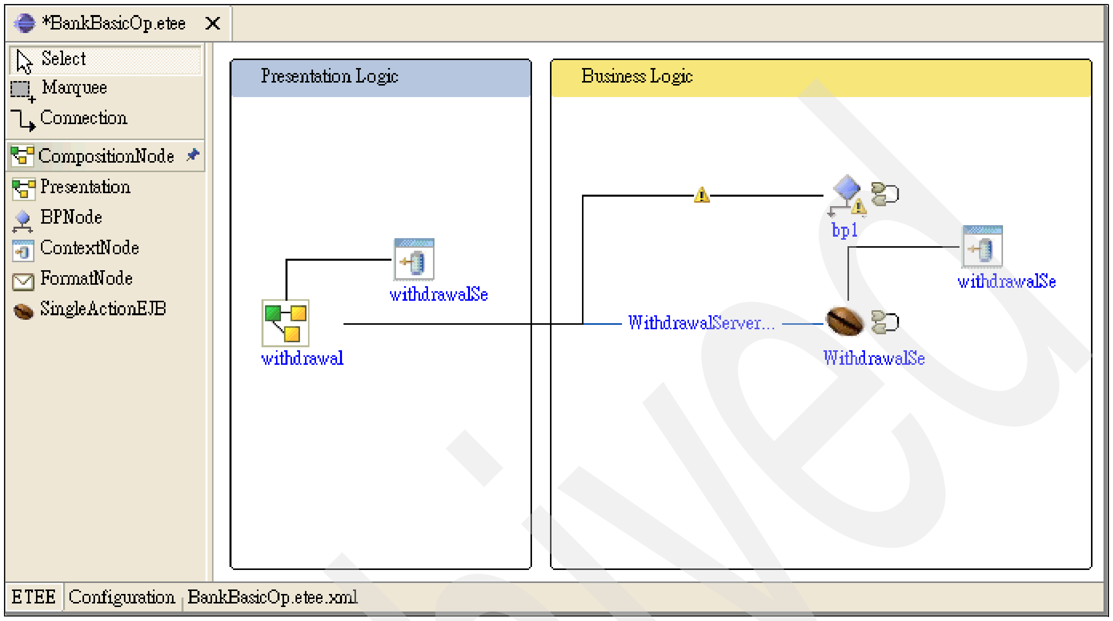
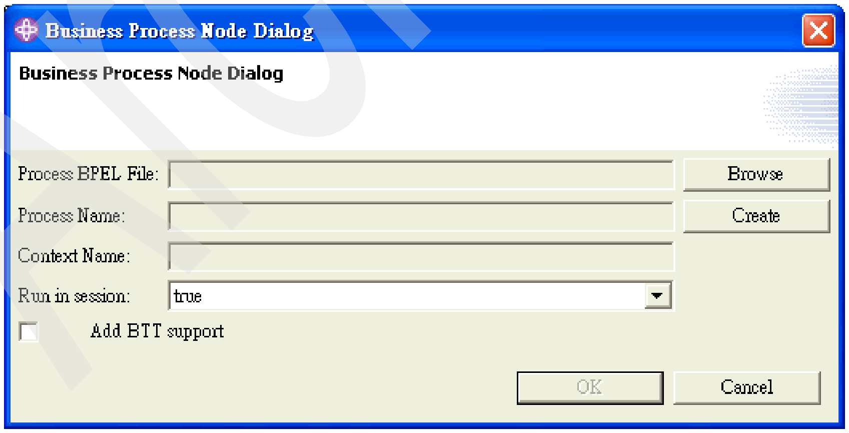


Figure 8-135 A new BP node added into the Graphical Editor

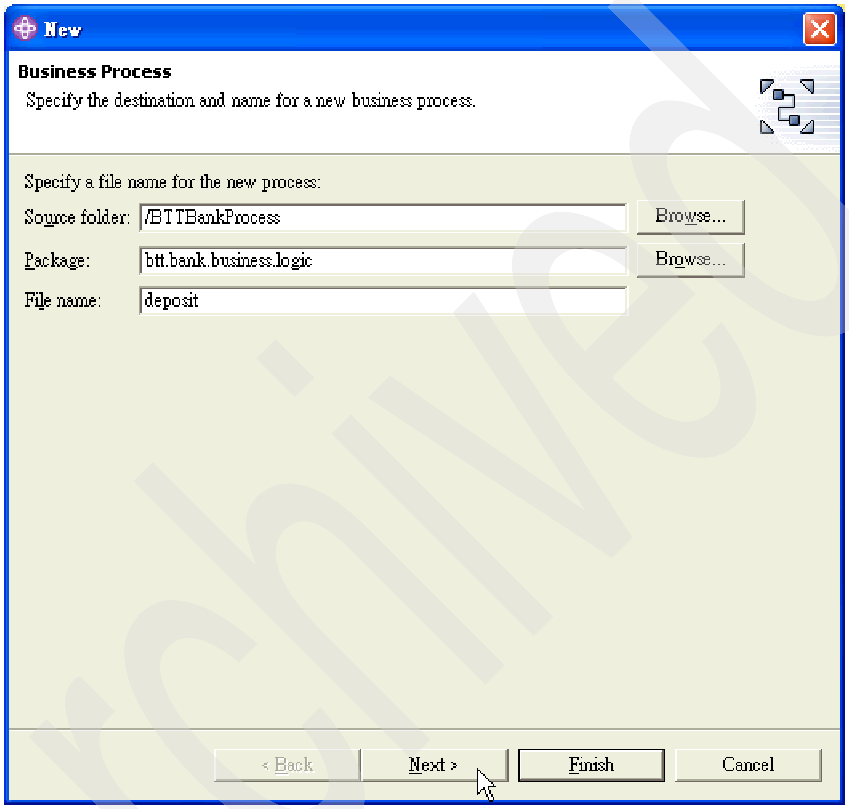
3. Click the **bp1** node, and select **Open Properties Dialog** from the context menu, as shown in Figure 8-136.



Process BPEL File:
Process Name:
Context Name:
Run in session: true
Add BTT support

Figure 8-136 The Business Process Node Dialog

4. Click **Create**, and enter the following values in the New Business Process dialog, as shown in Figure 8-137:
   – Source folder: `/BTTBankProcess`
   – Package: `btt.bank.business.logic`
   – File name: `deposit`


New

Business Process

Specify the destination and name for a new business process.

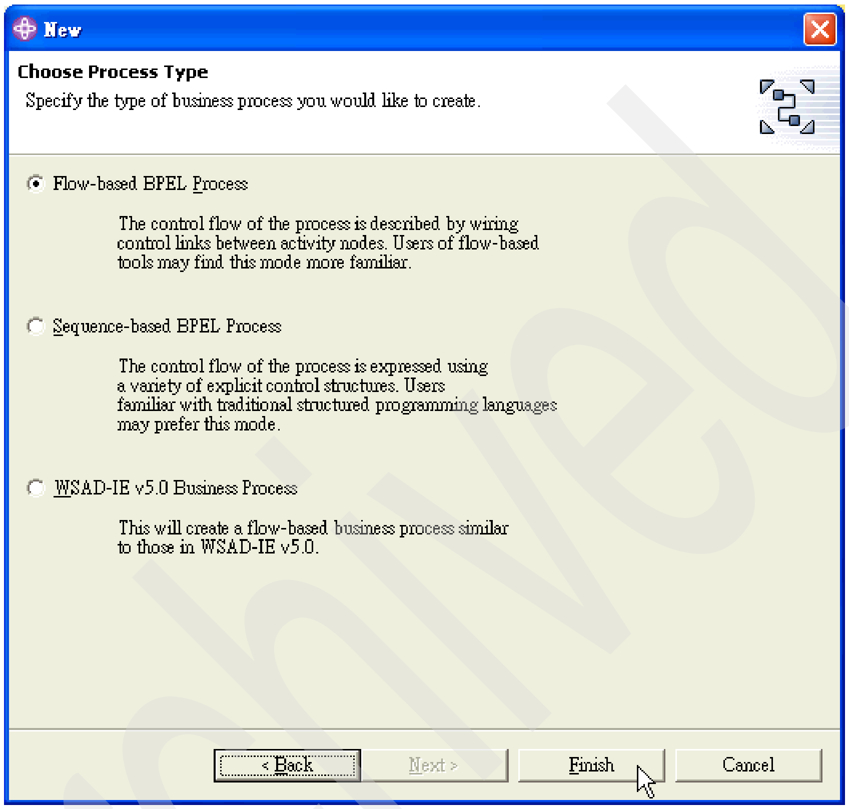
Specify a file name for the new process:

Source folder:	/BTTBankProcess
Package:	btt.bank.business.logic
File name:	deposit

Figure 8-137 Create the deposit business process

5. Click **Next**.

6. Select **Flow-based BPEL Process** as the process type, and then click **Finish**, as shown in Figure 8-138.



Specify the type of business process you would like to create.

[✓ Flow-based BPEL Process]

The control flow of the process is described by wiring control links between activity nodes. Users of flow-based tools may find this mode more familiar.

Sequence-based BPEL Process

The control flow of the process is expressed using a variety of explicit control structures. Users familiar with traditional structured programming languages may prefer this mode.

WSAD-IE v5.0 Business Process

This will create a flow-based business process similar to those in WSAD-IE v5.0.

Figure 8-138 Select flow-based BPEL Process as the process type

The deposit.bpel file will be opened in the BPEL Editor as shown in Figure 8-139 on page 405.

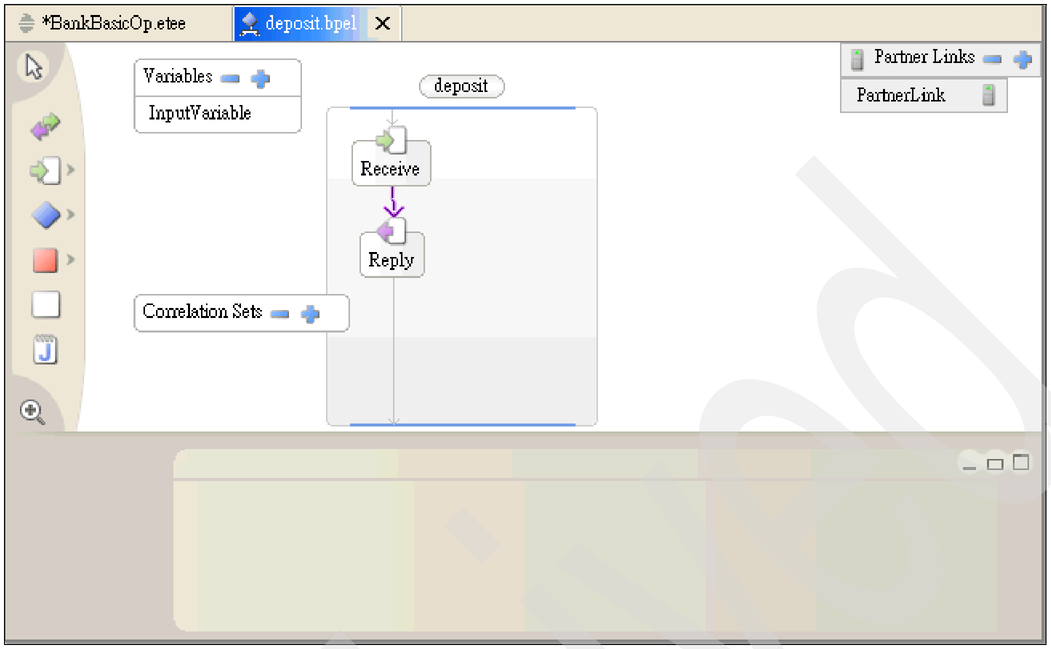


Figure 8-139 The deposit.bpel file is opened in BPEL editor

Using the BPEL Editor and the Business Process BTT Wizard

To use the BPEL Editor and the Business Projecess BTT Wizard, follow these steps:

1. In the BPEL Editor, select the **Java Snippet** icon (J) and add the snippet to the flow. Rename the snippet as `depositSnippet`. See Figure 8-140.

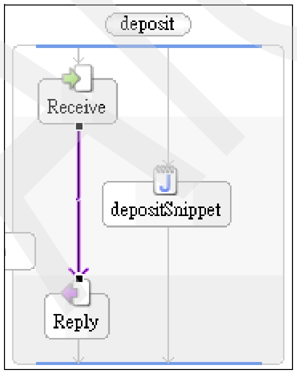


Figure 8-140 The depositSnippet icon added into the flow

2. Weave the snippet into the path between the Receive and Reply icons.
   a. Drag the **arrow** from the Reply icon to the depositSnippet icon. See Figure 8-141.

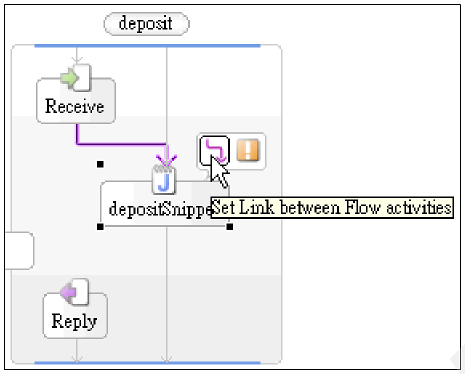


Figure 8-141 Drag the arrow from Reply icon to depositSnippet icon

   b. Click **Set Link** in the action bar of the depositSnippet, and then click the **Reply** icon. See Figure 8-142.

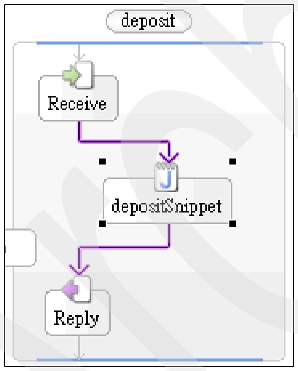
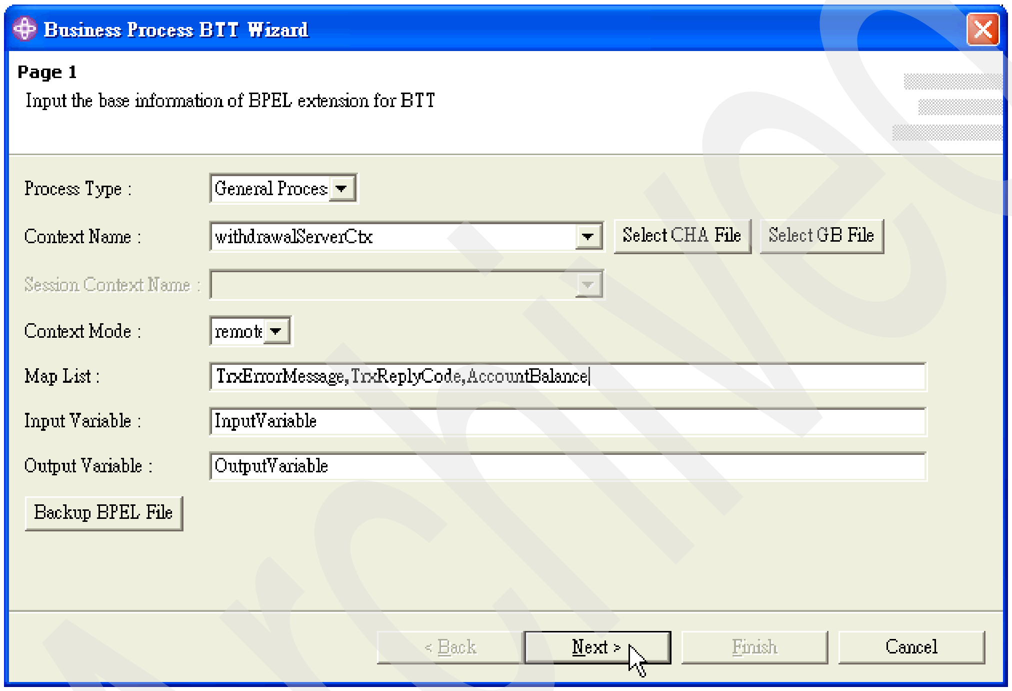


Figure 8-142 Set the link between depositSnippet and Reply node

3. Press Ctrl+S to save the changes.
4. Set up Branch Transformation Toolkit specific features for Deposit operation.
   a. Go back to **Graphical Editor** with BankBasicOp.etee, right-click the **deposit** business process icon, and select **Launch Business Process BTT Wizard** from the context menu.
   b. In the pop-up window, click **Select CHA File** to select the **dsectxt.xml** file in the business folder of the BTTBankBTT project. Select

withdrawalServerCtx in the Context Name field. Set `remote` in the Context Mode field and populate the Map List field with `TrxErrorMessage`, `TrxReplyCode`, `AccountBalance`, as shown in Figure 8-143. Click **Next**.

Note: In our sample, we chose **withdrawalServerCtx** for deposit operation since the data required by the two operations are the same. For simplicity, we reused **withdrawalServerCtx**.


Business Process BTT Wizard

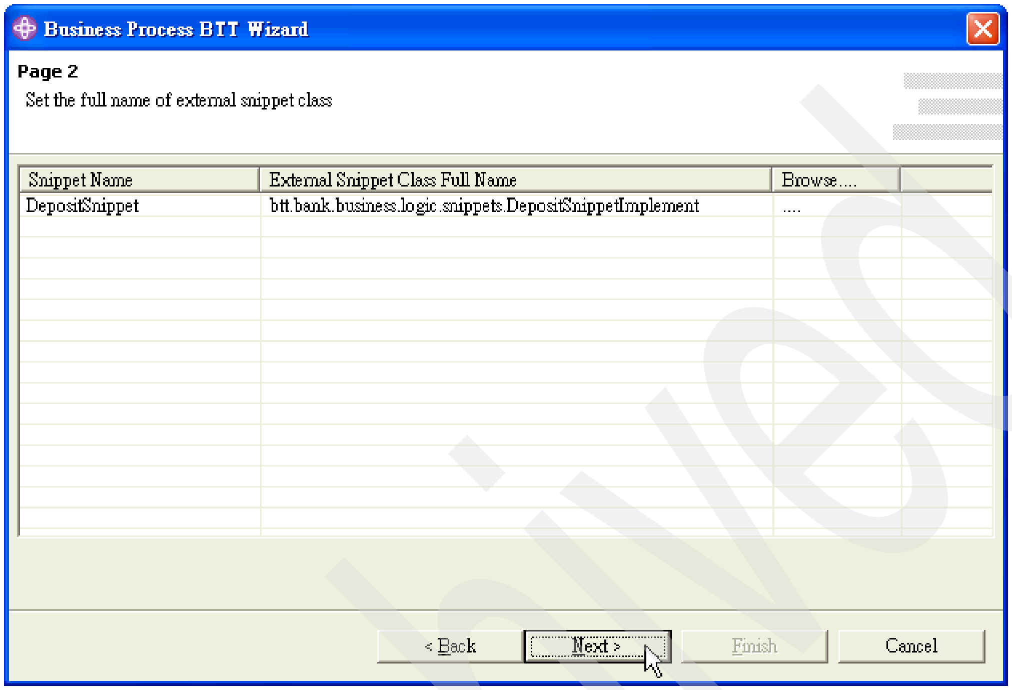
Page 1

Input the base information of BPEL extension for BTT

Process Type :	General Proces
Context Name :	withdrawalServerCtx
Session Context Name :	
Context Mode :	remote
Map List :	TrxErrorMessage,TrxReplyCode,AccountBalance
Input Variable :	InputVariable
Output Variable :	OutputVariable

Figure 8-143 Add context-related information to BPEL file

c. The implementation class of the DepositSnippet is shown in Figure 8-144. Click **Next** to continue.



Page 2

Set the full name of external snippet class

Snippet Name	External Snippet Class Full Name	Browse....
DepositSnippet	btt.bank.business.logic.snippets.DepositSnippetImplement	....

Figure 8-144 Snippet implementation file selection page

d. Set the navigation condition settings later. Click **Finish**.

5. Edit the depositInterface.wsdl file.

   a. In the BTTBankProcess project, open the **depositInterface.wsdl** file in the btt.bank.business.logic package with WSDL Editor, as shown in Figure 8-145.

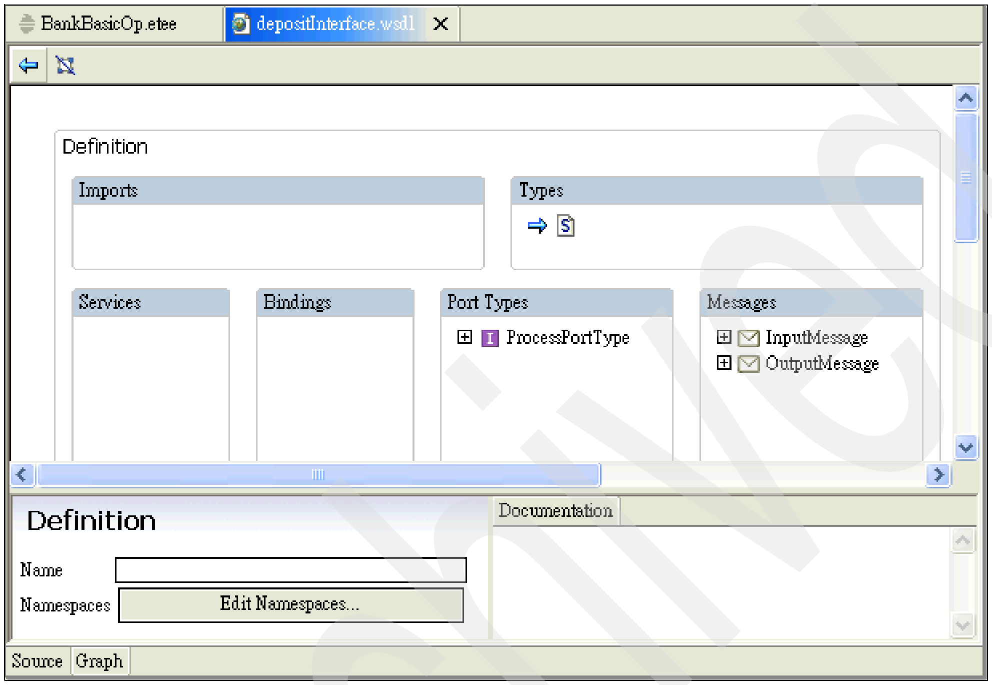


Figure 8-145 depositInterface.wsdl in WSDL Editor

   b. In the Messages area, expand the **InputMessage**, and right-click the part **contents**, and select **Delete**, as shown in Figure 8-146.

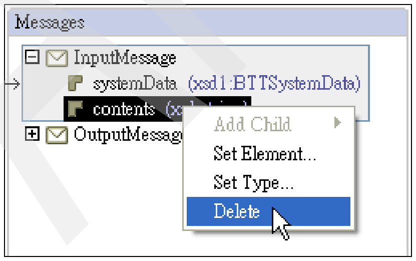
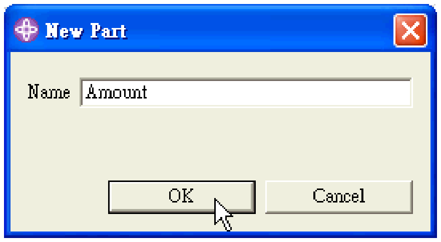


Figure 8-146 Delete contents part from InputMessage

c. Right-click **InputMessage**, select **Add Child** → **Part** to add a part. Type `Amount` in the New Part dialog, as shown in Figure 8-147 on page 410.



Name: Amount

Figure 8-147 Add Amount to InputMessage

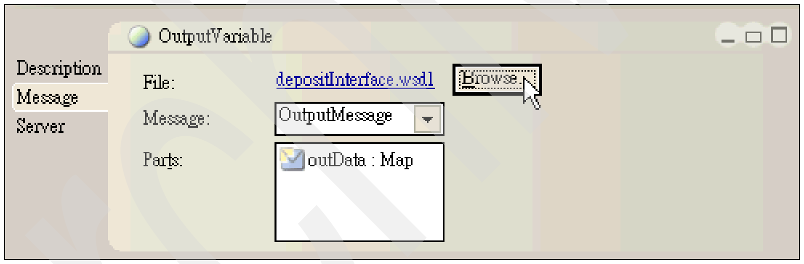
d. Likewise, add `AccountNumber` to InputMessage.

e. Save and close the **WSDL Editor**.

6. Set up variables for deposit.bpel.

   a. Open **deposit.bpel** with BPEL Editor.

   b. Select **OutputVariable** in the Variables area, and then click the **Message** tab in the details area, as shown in Figure 8-148.



File: depositInterface.wsdl
Message: OutputMessage
Parts: outData : Map

Figure 8-148 Set up the OutputVariable

   c. Click **Browse**. In the pop-up dialog, select **depositInterface.wsdl** in the btt.bank.business.logic package of the BTTBankProcess project. Select **OutputMessage** as the message. Click **OK**, as shown in Figure 8-149 on page 411.

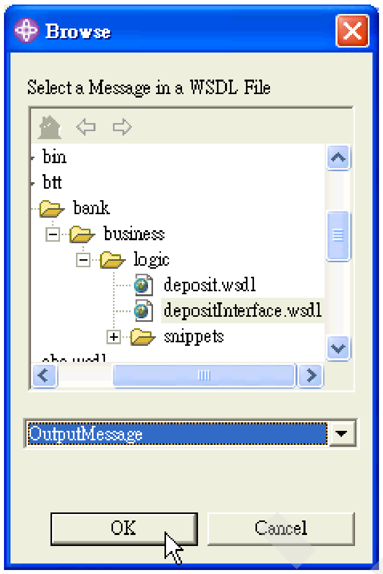


Figure 8-149 Select the OutputMessage of the depositInterface.wsdl as the deposit flow's OutputVariable

d. Select **cha** in the Variables area, then click the **Message** tab in the details area.

e. Click **Browse**. In the pop-up dialog box, then select **cha.wsdl** in the BTTBankProcess project. Select **cha** as the message. Click **OK**.

f. Save the changes and close the **BPEL Editor**.

Setting up project properties for BTTBankProcess

To set up the project properties for BTTBankProcess, follow these steps:

1. Right-click the **BTTBankProcess** project, and select **Properties** from the context menu.
2. In the pop-up window, select **Java JAR Dependencies**. Ensure that the **bttbase.jar**, **bttfmt.jar**, **bttsvrflow.jar**, **bttjdbjsvc.jar**, **bttjdbtsvc.jar**, **DummyJournal.jar**, and **bttsvcinfra.jar** check boxes have been selected in the JAR/Module list. Click **Apply**.
3. In the same pop-up window, select **Java Build Path**, and click the **Libraries** tab. Click **Add JARs** to add **bttbase.jar, bttfmt.jar**, **bttsvrflow.jar**, **bttjdbjsvc.jar**, **bttjdbtsvc.jar, and bttsvcinfra.jar** from the BTTBank project.
4. In the Java Build Path dialog box, click the **Projects** tab. Select **DummyJournal** project, and then click **OK**.

Adding code to the BTTBankProcess project

Implement the DepositSnippetImplement class by performing the following tasks:

1. Open **DepositSnippetImplement.java** from the btt.bank.business.logic.snippets package.
2. Add the import statements shown in Example 8-33.

Example 8-33 Additional import statements for DepositSnippetImplement.java

```
import com.ibm.btt.base.Context;
import com.ibm.btt.formatter.client.FormatElement;
import com.ibm.btt.services.jdbcjournalservice.Journal;
```

3. Add the code shown in Example 8-34 in the execute method.

Example 8-34 The execute method for the DepositSnippetImplement class

```
public int execute() throws BTTBPException {
   System.out.println("Executed depositServerOp.snippets.initial");
   //User code here
   try{
      // initialize context hierarchy
      Context BPContext = getContext() ;
      if(BPContext.getParent() == null){
         Context parent= Context.getContextByInstanceID(getSystemData()
                                                       .getInstanceId());
         BPContext.chainTo(parent);
      }

      // set data to deposit context
      String hostBuff = ((FormatElement)getFormat("withdrawalCSRequestFmt"))
                       .format(BPContext);
      setValueAt("HostBuff", hostBuff);
      System.out.println("deposit BP context:\n" +
                         BPContext.getKeyedCollection());

      // writes to the journal using the appropriate format
      Journal journal = (Journal)getService("JournalService");
      int recordNum = journal.addRecord(BPContext,"preSendJournalFmt");

      // Set hard coded response data to deposit context
      BPContext.setValueAt("TrxReplyCode", "00");
      BPContext.setValueAt("AccountBalance", "10000");
      BPContext.setValueAt("TrxErrorMessage", "depositOK");

      // Update the electronic journal record
      journal.updateRecord(recordNum,getContext(),"afterRecJournalFmt");
      journal.releaseServiceRequester();
   } catch(Exception e){
        e.printStackTrace();
    }
    System.out.println("deposit : executeJournalHostRequestDataStep ok!");
    return 1;
}
```

4. Save and close the **Java Editor** for DepositSnippetImplement.java.
5. Copy three files, **DummyJournal.properties**, **RemoteEJB.properties**, and **ServiceRequesterIDs.properties** from the ejbModule folder of the BTTBankEJB project to the BTTBankProcess project as shown in Figure 8-150.

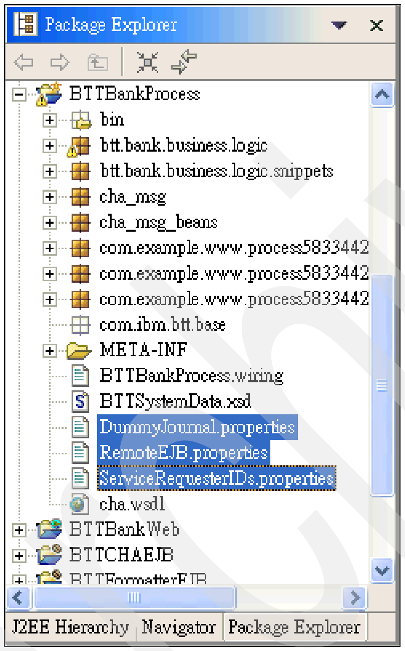


Figure 8-150 Add three properties files into BTTBankProcess project

Generating deploy code for deposit.bpel

To generate the deply code, follow these steps:

1. Right-click the **deposit.bpel** file from the btt.bank.business.logic package of the BTTBankProcess project. Select **Enterprise Services** → **Generate Deploy Code** from the context menu, as shown in Figure 8-151 on page 414.

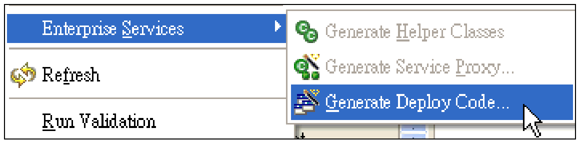
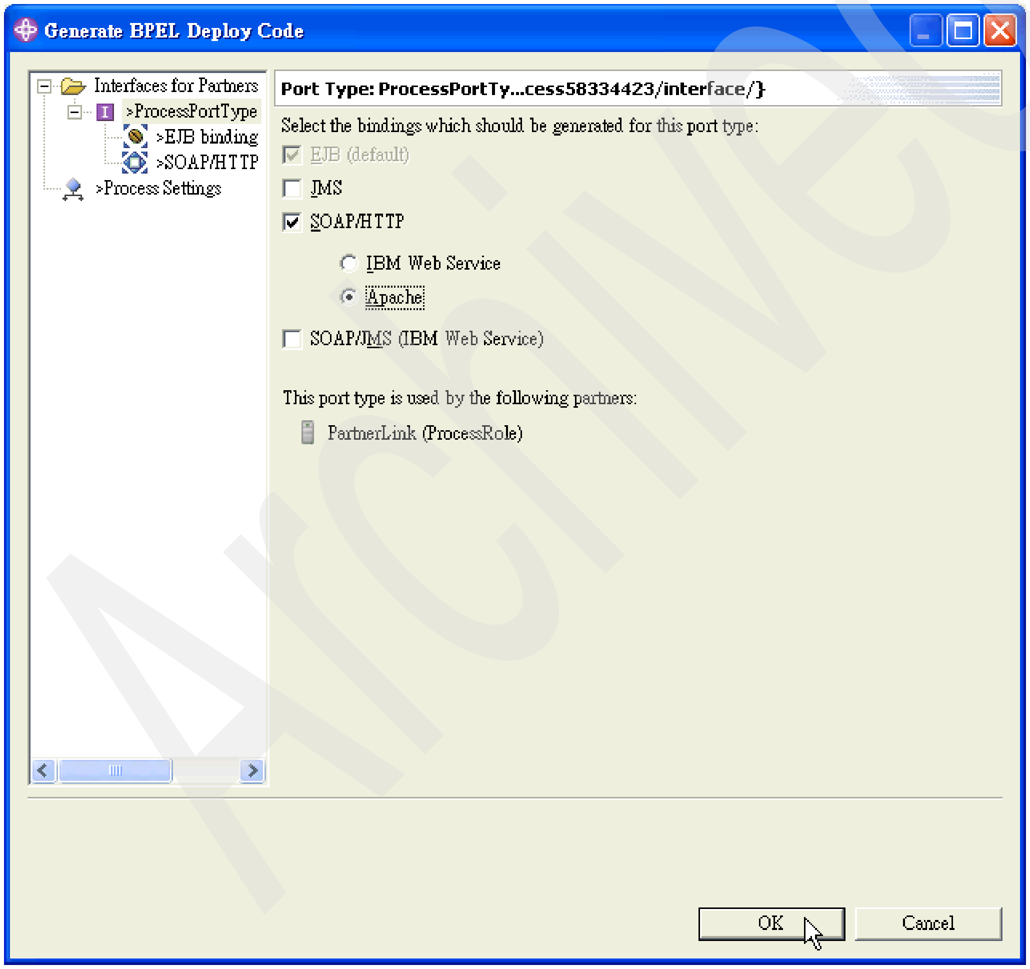


Figure 8-151 Generating deploy code

2. In the pop-up window, click **ProcessPortType** in the left navigation panel. Select the **SOAP/HTTP** check box and then the **Apache** radio button. Click **OK**, as shown in Figure 8-152.


Generate BPEL Deploy Code

Interfaces for Partners
>ProcessPortType
>EJB binding
>SOAP/HTTP
>Process Settings

Port Type: ProcessPortTy...cess58334423/interface/}

Select the bindings which should be generated for this port type:

- [✓] EJB (default)
- [ ] JMS
- [✓] SOAP/HTTP
  - ( ) IBM Web Service
  - (●) Apache
- [ ] SOAP/JMS (IBM Web Service)

This port type is used by the following partners:

PartnerLink (ProcessRole)

OK Cancel

Figure 8-152 Generate BPEL Deploy Code

The following projects are generated, containing the deploy code:

- BTTBankProcessEAR
- BTTBankProcessEJB
- BTTBankProcessWeb

You will see some errors. These can be fixed later (Can we say this? If the solution is provided in this chapter/ book, shouldn't we cross-reference it?).

Configuring generated projects

To configure **BTTBankProcessEAR**, perform the following tasks:

1. Ensure that both **BTTBankProcess** and **DummyJournal** projects are added as the Project Utility JARs.
2. Import the following JARs from <BTT_install_dir>/jars to the BTTBankProcessEAR project:
   - **bttbase.jar**
   - **bttfmt.jar**
   - **bttjdbjsvc.jar**
   - **bttjdbtsvc.jar**
   - **bttsvcinfra.jar**
   - **bttsvcbean.jar**
   - **bttsvrflow.jar**

To configure **BTTBankProcessEJB**, perform the following tasks:

1. In the Java JAR Dependencies panel, ensure that both **DummyJournal** and **BTTBankProcess** are added as the dependent JARs.
2. In the **Libraries** page of the Java Build Path panel, do the following:
   a. Ensure that the JRE System Library is WebSphere v5.1 EE JRE.
   b. Ensure that these two JARs are added into the build path. If not, add them by clicking **Add Variable**.
      - **WAS_EE_V51/lib/wsadiebp.jar**
      - **WAS_EE_V51/lib/wsatlib.jar**
   c. Add the following JARs into the build path (use those that are there in the BTTBankProcessEAR project):
      - **bttbase.jar**
      - **bttfmt.jar**
      - **bttjdbjsvc.jar**
      - **bttjdbtsvc.jar**
      - **bttsvcinfra.jar**
      - **bttsvrflow.jar**

To configure **BTTBankProcessWeb**, in the Libraries page of the Java Build Path panel, perform the following tasks:

a. Ensure that the JRE System Library is WebSphere v5.1 EE JRE.

b. Ensure that these two JARs are added into the build path. If not, add them by clicking **Add Variable**.

   - **WAS_EE_V51/lib/wsadiebp.jar**
   - **WAS_EE_V51/lib/wsatlib.jar**

c. Add the following JARs into the build path (use those in the BTTBankProcessEAR project):

   - **bttbase.jar**
   - **bttfmt.jar**
   - **bttjdbjsvc.jar**
   - **bttjdbtsvc.jar**
   - **bttsvcinfra.jar**
   - **bttsvrflow.jar**

To regenerate the deploy code for deposit.bpel, perform the following tasks:

1. Right-click the **deposit.bpel** file from the btt.bank.business.logic package of the BTTBankProcess project. Select **Enterprise Services** → **Generate Deploy Code** from the context menu, as shown in Figure 8-151 on page 414.
2. In the pop-up window, click **ProcessPortType** in the left navigation panel. Select the **SOAP/HTTP** check box and then the **Apache** radio button. Click **OK**.

Laying out the presentation logic for the deposit operation

Go back to the Graphical Editor. You will see that the canvas looks like the one shown in Figure 8-153 on page 417. All the three components use the withdrawalServerCtx. You can reuse the **withdrawal.gph** Web diagram and the **struts-btt-withdrawal.xml** Struts module here.

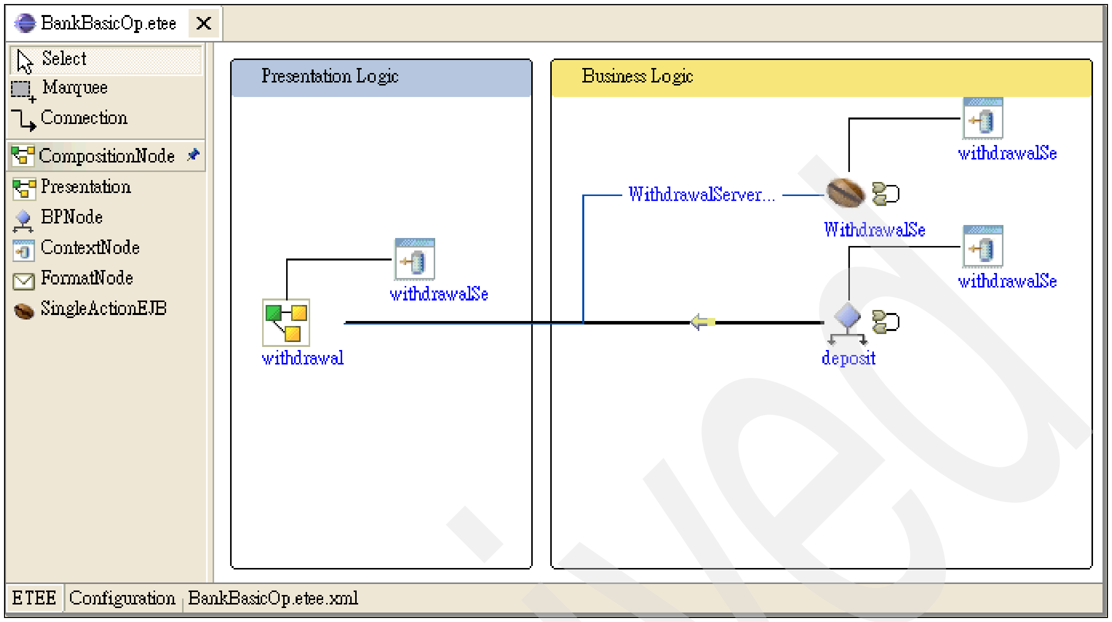


Figure 8-153 The deposit BP node with withdrawalServerCtx appears

To add more Web components, perform the following tasks:

1. Double-click the **withdrawal** icon in the Presentation Logic block to bring up the Web diagram.
2. Use the icons in the palette to add two actions and a Web page to the canvas.

   Rename the actions as shown in Figure 8-154 on page 418.

   Rename the Web page as `deposit.jsp` with `/btt/bank/ui/struts/withdrawal/` module name prepended.
3. Move the **result.jsp** icon a little lower since you will need it to display the result of a deposit operation.

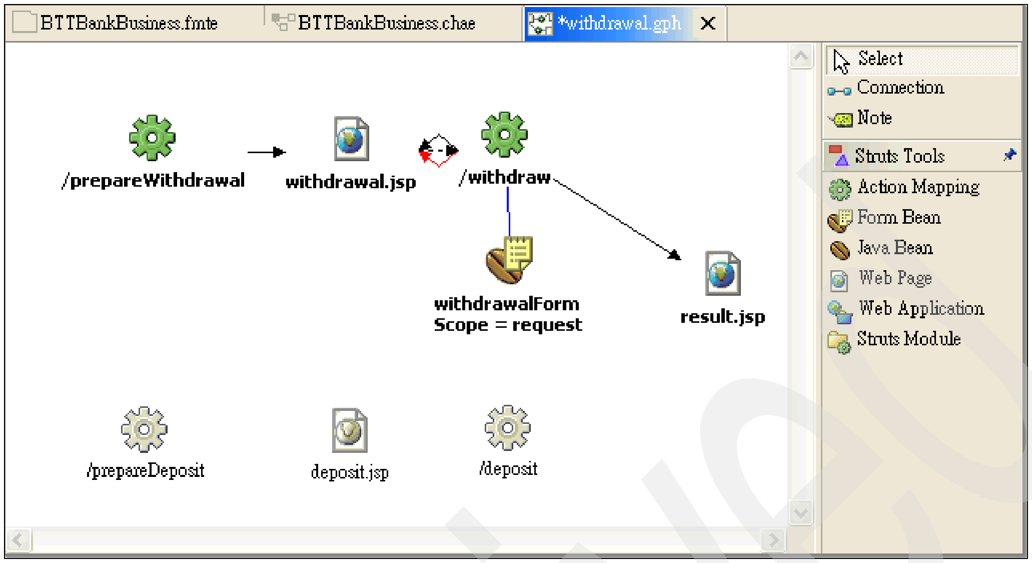


Figure 8-154 More components added to withdrawal Web diagram

To design the application flow, perform the following tasks:

1. Create a connection from the deposit.jsp to the **/deposit** action.
2. Create a local forward back from the /deposit action to the deposit.jsp. Use `failure` as the forward name.
3. Create a local forward to the result.jsp, and mark the forward as `success`.
4. Associate the /deposit action to the withdrawalForm form bean.
5. Press Ctrl+S to save the change.

After completing these steps, the Web diagram will look as shown in Figure 8-155.

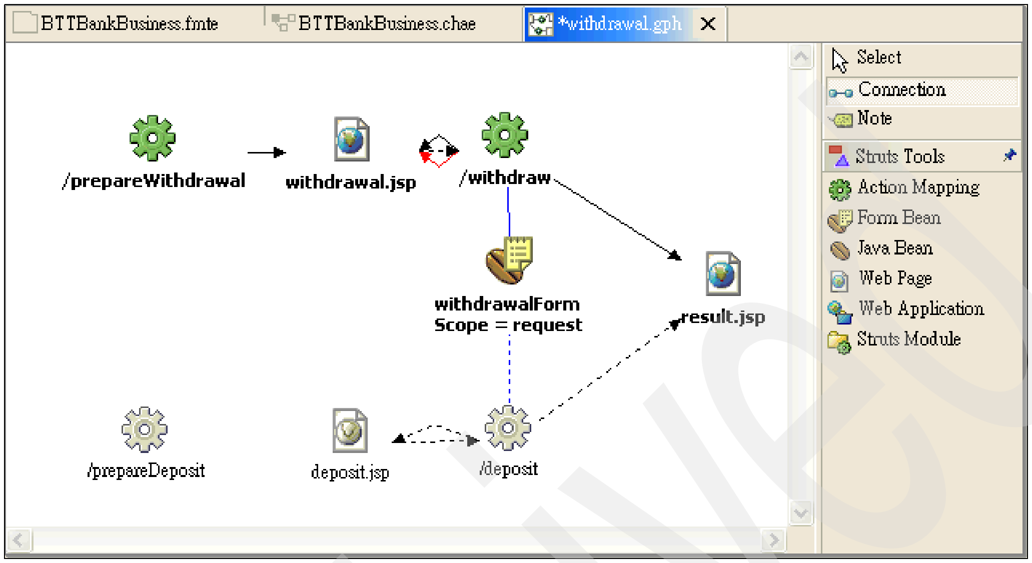


Figure 8-155 Connect Web components

To realize the Struts actions, perform the following tasks.

To realize the /prepareDeposit action:

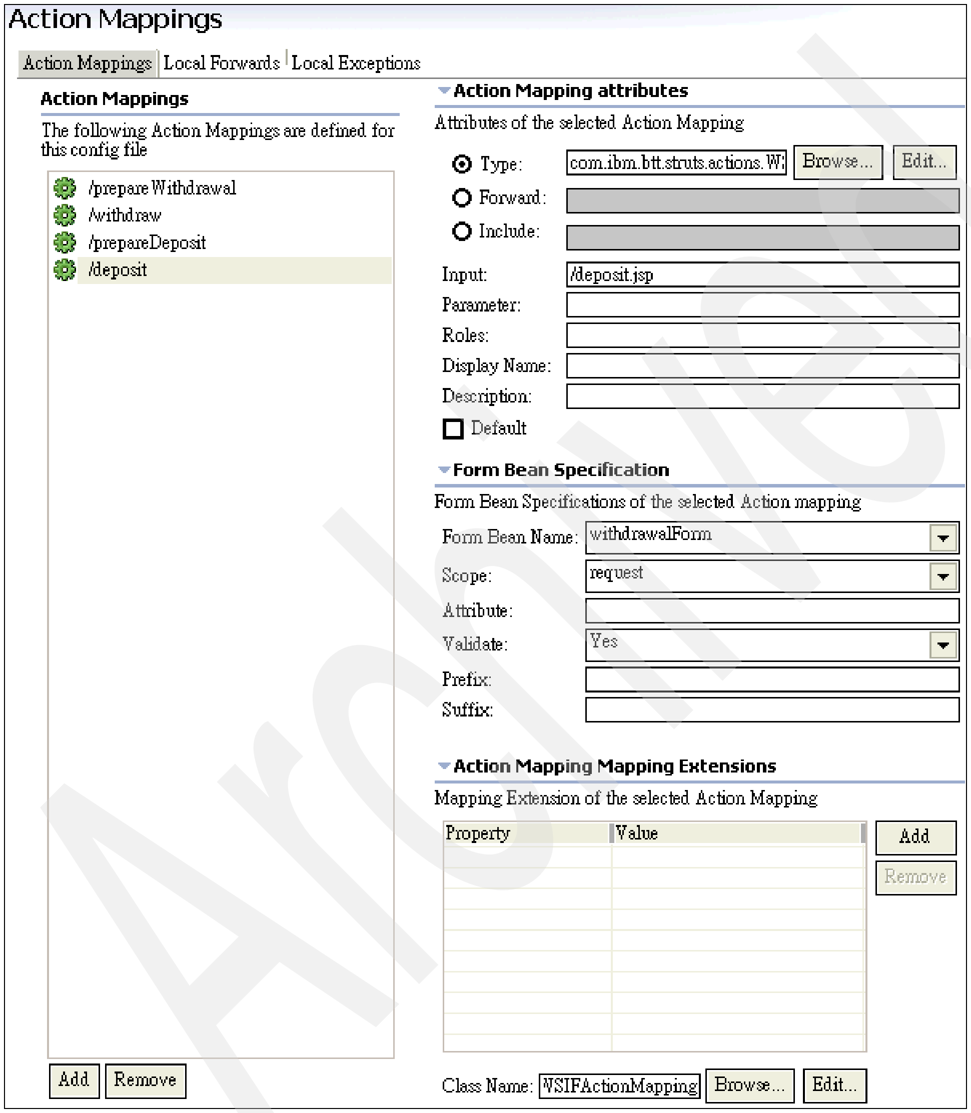
1. Double-click the **/prepareDeposit** action in the Web diagram. As you can see, the **struts-btt-withdrawal.xml** has been opened in the Struts Configuration File Editor.
2. In the Action Mapping attributes section, select the **Forward** radio button and type `/deposit.jsp` in the text box. Press **Ctrl+S** to save the change.

To realize the /deposit action, switch back to the withdrawal Web diagram:

1. Double-click the **/deposit** action in the Web diagram. The struts-btt-withdrawal.xml file opens.
2. Select the **Type** radio button in the Action Mapping attributes page, and populate the **Type** field with the value `com.ibm.btt.struts.actions.WSIFAction`. Populate the Input field with the value `/deposit.jsp`.
3. Select **withdrawalForm** for the Form Bean Name field in the Form Bean Specification section. Select **Yes** for the Validate field.

4. In the Action Mapping Extensions section, populate in the Class Name field with `com.ibm.btt.struts.config.BTTWSIFActionMapping`.

   See Figure 8-156.



Action Mappings

Action Mappings

The following Action Mappings are defined for this config file

- /prepareWithdrawal
- /withdraw
- /prepareDeposit
- /deposit

Action Mapping attributes

Attributes of the selected Action Mapping

● Type: com.ibm.btt.struts.actions.W
○ Forward:
○ Include:

Input: /deposit.jsp
Parameter:
Roles:
Display Name:
Description:
☐ Default

Form Bean Specification

Form Bean Specifications of the selected Action mapping

Form Bean Name: withdrawalForm
Scope: request
Attribute:
Validate: Yes
Prefix:
Suffix:

Action Mapping Mapping Extensions

Mapping Extension of the selected Action Mapping

Property	Value

Class Name: WSIFActionMapping

Figure 8-156 Set up action mappings for the /deposit action

5. Click the **Local Forwards** tab. Ensure that **/deposit** is selected.

   Click **Add** in the Local Forwards section, name the new forward `success`, and populate the Path field in the Forward Attributes section with `/result.jsp`.

   Click **Add** again to add the forward `failure` with the `/deposit.jsp` in the Path field.

6. Save and close the **struts-btt-withdrawal.xml** file.

To realize the deposit.jsp, perform the following tasks:

1. Double-click the **deposit.jsp** in the Web diagram.
2. When the New JSP file wizard opens, the wizard provides the default values, which should be accepted. Ensure that **Struts JSP** is selected in the Model field, and that **Configure advanced options** is checked. Click **Next**.
3. In the tag libraries page, choose **Struts html** and Branch Transformation Toolkit's **/WEB-INF/btt-html.tld** tag libraries. Click **Next**.
4. In the next page, click **Next** to accept the default, or deselect **Use workbench encoding** to use UTF-8, and then click **Next**.
5. Click **Next** when the JSP File Choose Method Stubs to generate dialog box opens.
6. In the Form Field Selection page, the withdrawalForm is selected by default. To specify which fields to use, check the **accountNumber**, and **amount** fields. Both accountNumber and amount will be the input fields. Click **Next**.
7. In the next dialog box, design the input form used in the deposit.jsp page. For the accountNumber field, input the following parameters:

   – ID: `AccountNumber`
   – Label: `AccountNumber`

   For the **amount** field, input the following parameters, and click **Finish**:

   – ID: `Amount`
   – Label: `Amount`

To complete the Web diagram, perform the following steps:

For /prepareDeposit action, do the following:

1. Close the **withdrawal.gph** Web diagram and reopen it.
2. Right-click the **/prepareDeposit** action. Select **Draw** → **Draw Selected From** from the context menu.
3. Select **Forward** → **/deposit.jsp** check box and click **OK**.

For /deposit action, do the following:

1. Right-click **/deposit** action. Select **Draw** → **Draw Selected From** in the context menu.
2. Select **<input>** → **/deposit.jsp** check box and click **OK**.
3. Press Ctrl+S to save your work.

The final Web diagram looks as shown in Figure 8-157.

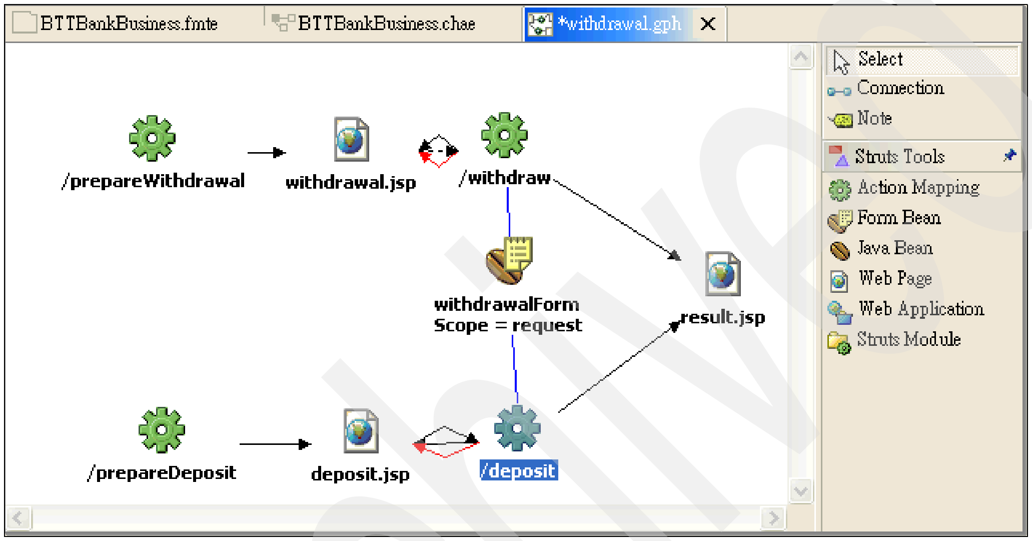


Figure 8-157 The final Web diagram

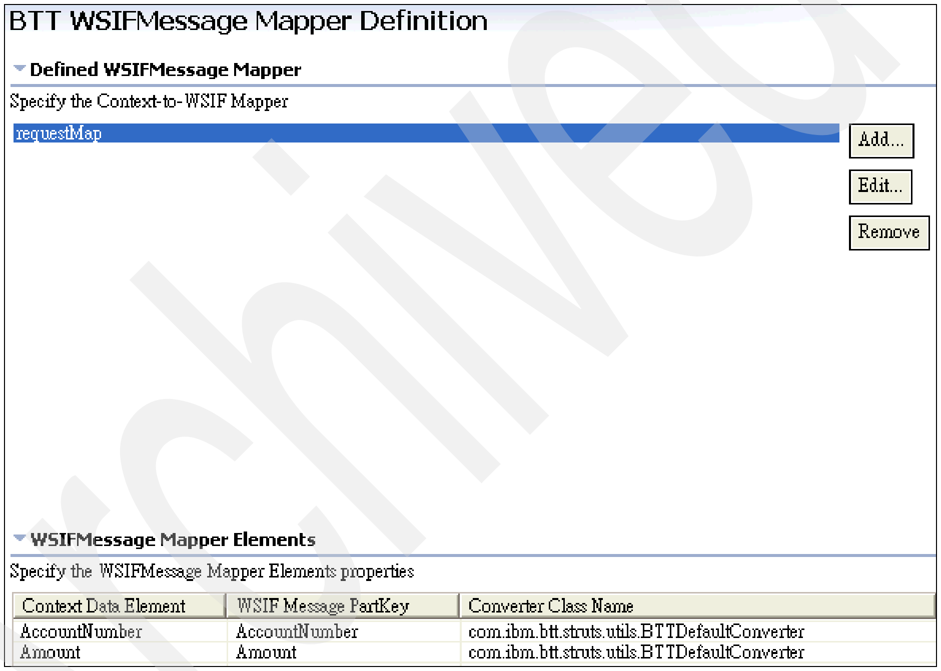
To apply Branch Transformation Toolkit-specific settings to struts-btt-withdrawal.xml, perform the following tasks:

1. Right-click the **WebContent/btt/bank** folder in the BTTBankWeb project, select **New** → **Folder** to create a new folder named `business`. Create a `logic` folder under the `business` folder.
2. Copy **BTTSystemData.xsd** and **cha.wsdl** from the BTTBankProcess project to the WebContent folder of the BTTBankWeb project.
3. Copy **deposit.wsdl**, **deposit_ProcessPortType_SOAP.wsdl**, and **depositInterface.wsdl** from the btt.bank.business.logic package of the BTTBankProcess project to the WebContent/btt/bank/business/logic folder of the BTTBankWeb project.
4. Reopen **struts-btt-withdrawal.xml** by **Open with** → **Struts Tools BTT Extensions** command.

5. Click the **WSIFMessage Mapper** tab, click **Add** in the Defined WSIFMessage Mapper section to add a mapper named `requestMap`. Click **Add** in the WSIFMessage Mapper Elements section to add the elements shown in Table 8-5 into Figure 8-158:

Table 8-5 Data elements defined in the requestMap

Context data element	WSIF message partkey	Converter class name
AccountNumber	AccountNumber	
Amount	Amount	


BTT WSIFMessage Mapper Definition

Defined WSIFMessage Mapper

Specify the Context-to-WSIF Mapper

requestMap

WSIFMessage Mapper Elements

Specify the WSIFMessage Mapper Elements properties

Context Data Element	WSIF Message PartKey	Converter Class Name
AccountNumber	AccountNumber	com.ibm.btt.struts.utils.BTTDefaultConverter
Amount	Amount	com.ibm.btt.struts.utils.BTTDefaultConverter

Figure 8-158 The requestMap and the data elements

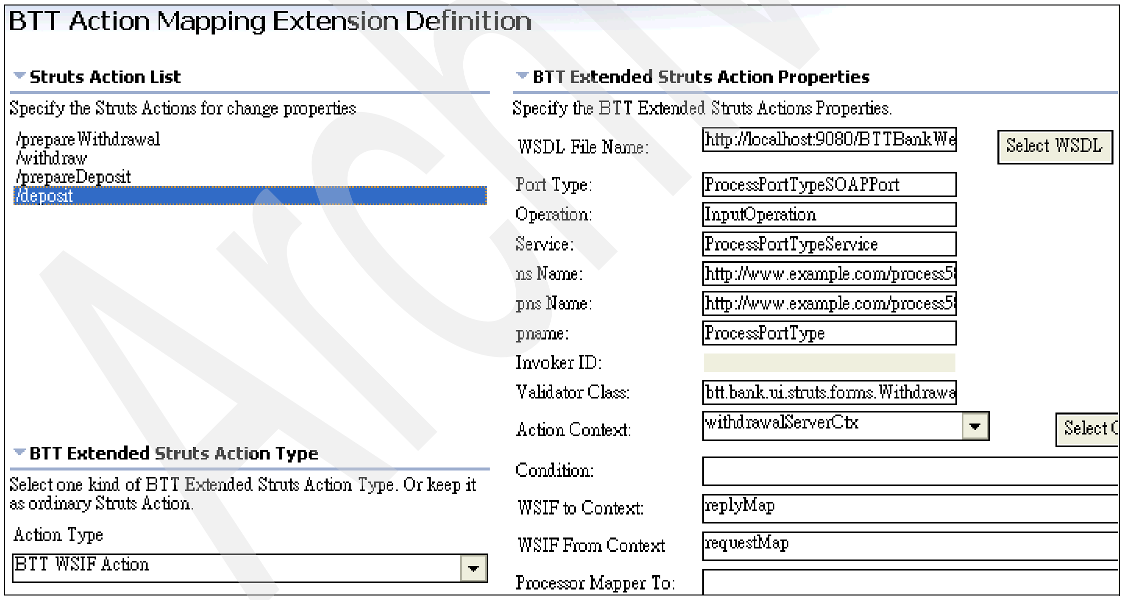
Add another mapper named `replyMap`, and define the elements shown in Table 8-6:

Table 8-6 Data elements defined in the replyMap

Context data element	WSIF message partkey	Converter class name
TrxErrorMessage	TrxErrorMessage	
TrxReplyCode	TrxReplyCode	

Context data element	WSIF message partkey	Converter class name
AccountBalance	AccountBalance	

6. Click the **Action Mapping** tab. Select the **/deposit** action in the Struts Action List. You will see that **BTT WSIF Action** is selected as the Action Type in the BTT Extended Struts Action Type section.
7. Click the **Select WSDL** button in the BTT Extended Struts Action Properties section, and select the **deposit_ProcessPortType_SOAP.wsdl** file from the /WebContent/btt/bank/business/logic directory of the BTTBankWeb project.
8. Enter the following values:
   - Validator Class: `btt.bank.ui.struts.forms.WithdrawalXVal`
   - WSIF to Context: `replyMap`
   - WSIF from Context: `requestMap`
9. Click **Select Context** next to the Action Context field, and select **dsectxt.xml** in the presentation folder of the BTTBankBTT project. Choose **withdrawalServerCtx** for the Action Context field, as shown in Figure 8-159.
10. Press Ctrl+S to save the changes.



BTT Action Mapping Extension Definition

Struts Action List

Specify the Struts Actions for change properties

/prepareWithdrawal
/withdraw
/prepareDeposit
/deposit

BTT Extended Struts Action Type

Select one kind of BTT Extended Struts Action Type. Or keep it as ordinary Struts Action.

Action Type: BTT WSIF Action

BTT Extended Struts Action Properties

Specify the BTT Extended Struts Actions Properties.

WSDL File Name:	http://localhost:9080/BTTBankWe
Port Type:	ProcessPortTypeSOAPPort
Operation:	InputOperation
Service:	ProcessPortTypeService
ns Name:	http://www.example.com/process5
pns Name:	http://www.example.com/process5
pname:	ProcessPortType
Invoker ID:	
Validator Class:	btt.bank.ui.struts.forms.Withdrawa
Action Context:	withdrawalServerCtx
Condition:	
WSIF to Context:	replyMap
WSIF From Context	requestMap
Processor Mapper To:	

Figure 8-159 Set up the WSIF Action for the deposit operation

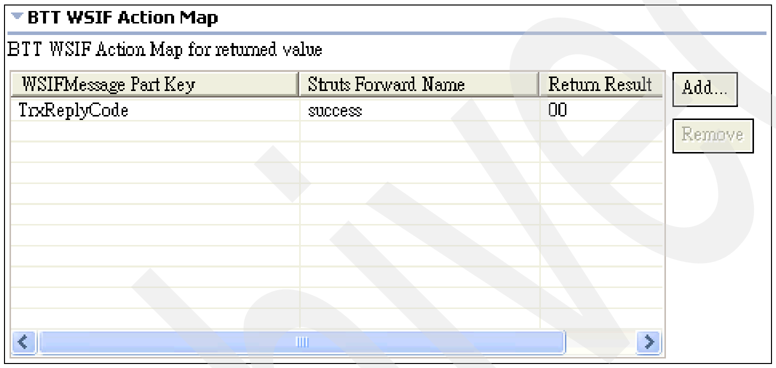
11. In the BTT WSIF Action Map section, click **Add** to map an item using the values specified in Table 8-7 on page 425. In this example, two zeros ("00") of

the TrxReplyCode means that the process ran successfully. When the Struts extension receives the successful reply code, Struts performs the success forward, /result.jsp, in this case.

Table 8-7 Properties for the success forward of the /deposit WSIF action

WSIFMessage Part Key	Struts Forward Name	Return Result
TrxReplyCode	success	00

After adding the BTT WSIF Action Map, you will see it displayed as shown in Figure 8-160:



BTT WSIF Action Map for returned value

WSIFMessage Part Key	Struts Forward Name	Return Result
TrxReplyCode	success	00

Figure 8-160 Set the success forward for /deposit action

12. Save and close the **Struts Tools BTT Extensions Editor**.

To tailoring the deposit.jsp, perform the following steps:

1. Change the **deposit.jsp** file as shown in bold in Example 8-35:

Example 8-35 The final deposit.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/btt-html.tld" prefix="btt" %>
<btt:html>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"
%>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>deposit.jsp</TITLE>
```

```
</HEAD>

<BODY>
<btt:errors/>
<btt:form action="/deposit">
   <TABLE border="0">
      <TBODY>
         <TR>
            <TD align="right"><btt:label text="AccountNumber"/></TD>
            <TD align="left"><btt:text property="AccountNumber"/></TD>
         </TR>
         <TR>
            <TD align="right"><btt:label text="Amount"/></TD>
            <TD align="left"><btt:text property="Amount"/></TD>
         </TR>
         <TR>
            <TD><html:submit property="Deposit" value="Deposit" /></TD>
            <TD><html:reset /></TD>
         </TR>
      </TBODY>
   </TABLE>
</btt:form>
</BODY>
</btt:html>
```

To change the struts-btt-signin.xml to use deposit operation, perform the following tasks:

1. Open the **struts-btt-signin.xml** file using Struts Configuration File Editor.
2. Click the **Source** tab and make the changes as shown in bold in Example 8-36:

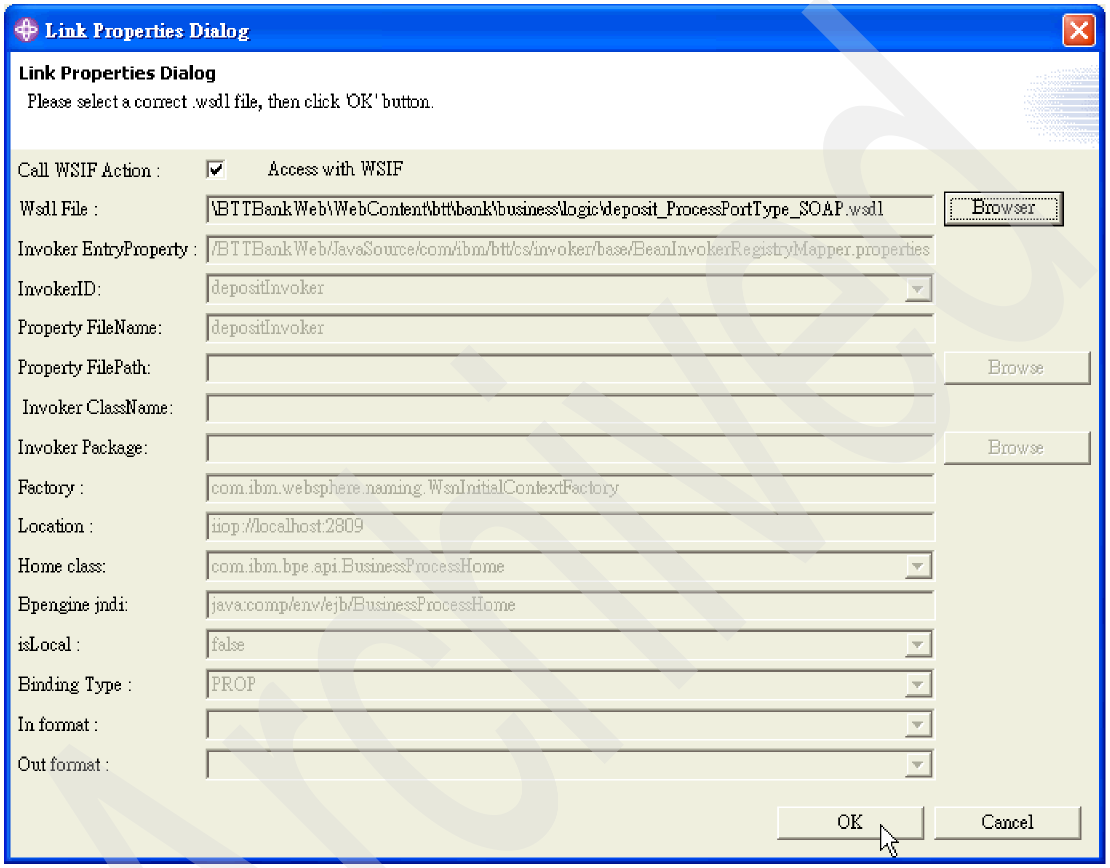
Example 8-36 Modified struts-btt-signin.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<struts-config>

   <data-sources>
   </data-sources>

   <form-beans>
     <form-bean name="signInForm"
                type="btt.bank.ui.struts.forms.SignInForm"/>
   </form-beans>

   <global-exceptions>
   </global-exceptions>
   <global-forwards>
   </global-forwards>

   <action-mappings>
      <action path="/prepareSignIn"
            forward="/signin.jsp"
            className="org.apache.struts.action.ActionMapping"
            parameter="signIn"/>

      <action name="signInForm"
          path="/signIn"
          className="com.ibm.btt.struts.config.BTTEJBActionMapping"
          type="com.ibm.btt.struts.actions.EJBSignInAction"
          input="/signin.jsp"
          invokerId="signInInvoker"
          validator="btt.bank.ui.struts.forms.SignInXVal"
          validate="true"
          parameter="signIn">

            <forward name="success"
               contextRelative="true"
               path="/btt/bank/ui/struts/withdrawal/prepareDeposit.do" />

           <forward name="signin"
                path="/signin.jsp"/>
      </action>
   </action-mappings>

   <flowcontext contextName="signInCtx" local="true" />

   <plug-in className="com.ibm.btt.struts.plugins.BTTDefaultNotifier" />

   <finals>
     <final type="forward"
            name="/btt/bank/ui/struts/withdrawal/prepareDeposit.do"/>
   </finals>

</struts-config>
```

To wire the presentation logic to the business logic for deposit operation, perform the following tasks:

1. Open **BankBasicOp.etee** with the Graphical Editor.
2. Double-click the **line** between the withdrawal presentation node and the deposit BP node. In the pop-up window, select the **Access with WSIF** check box.

3. Click **Browse** and select **deposit_ProcessPortType_SOAP.wsdl** in the folder WebContent/btt/bank/business/logic of the BTTBankWeb project.
4. Click **OK**.
5. Click **OK** again to complete the setup as shown in Figure 8-161.


Link Properties Dialog

Link Properties Dialog

Please select a correct .wsdl file, then click 'OK' button.

Call WSIF Action :	✓ Access with WSIF
Wsdl File :	\BTTBankWeb\WebContent\btt\bank\business\logic\deposit_ProcessPortType_SOAP.wsdl
Invoker EntryProperty :	/BTTBankWeb/JavaSource/com/ibm/btt/cs/invoker/base/BeanInvokerRegistryMapper.properties
InvokerID:	depositInvoker
Property FileName:	depositInvoker
Property FilePath:	
Invoker ClassName:	
Invoker Package:	
Factory :	com.ibm.websphere.naming.WsnInitialContextFactory
Location :	iiop://localhost:2809
Home class:	com.ibm.bpe.api.BusinessProcessHome
Bpengine jndi:	java:comp/env/ejb/BusinessProcessHome
isLocal :	false
Binding Type :	PROP
In format :	
Out format :	

Figure 8-161 Set up WSIF link properties

6. Press Ctrl+S to save your work.

Figure 8-162 shows the final design within the Graphical Editor.

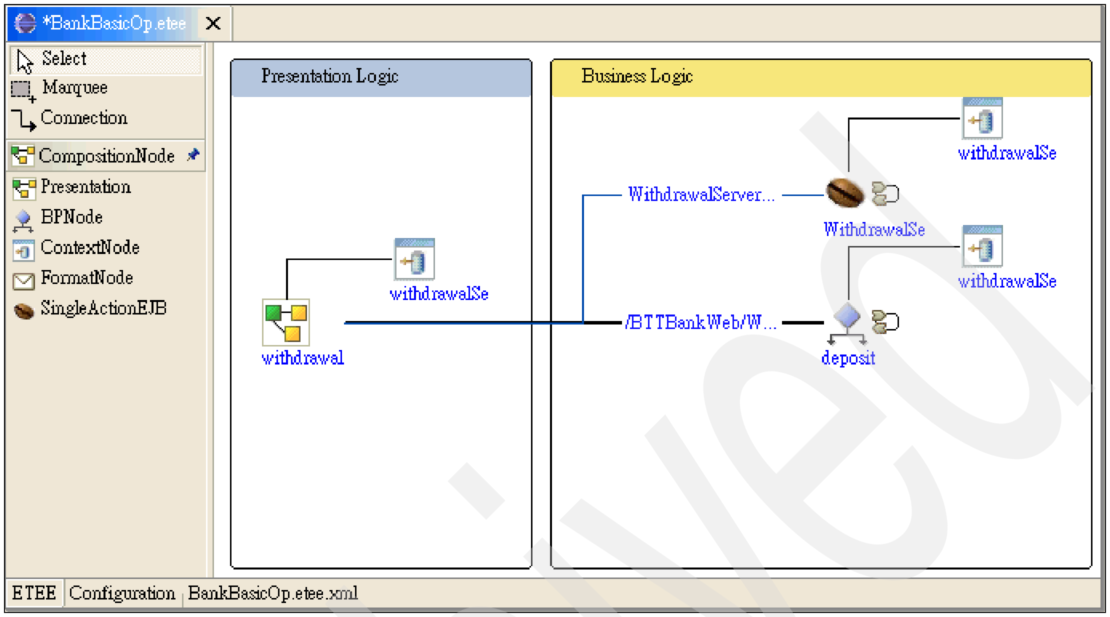


Figure 8-162 The final design in the Graphical Editor

To test the deposit business process, perform the following tasks:

Ensure that the DB2 server is ready and running:

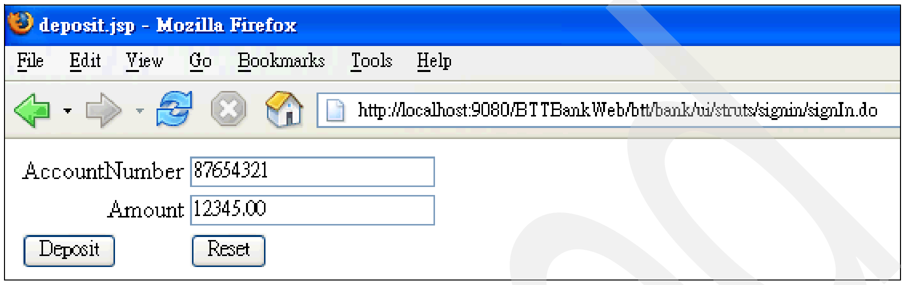
1. Add **BTTBankProcessEAR** to the WBI SF server instance.
2. In the Server perspective, right-click **WBI SF** either from the Server Configuration view or the Servers view. Select **Start** from the context menu.

   In the console view, you will see a lot of messages. This means that the server is running. When you see the "`Server server1 open for e-business`" message, it means the server is fully started.
3. Get into the sign-in page using the following URL:

   ```
   http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/signin/prepareSignIn.do
   ```

4. Input the following values and click **Submit**:
   – Use ID: `user01`
   – Password: `user01` ,

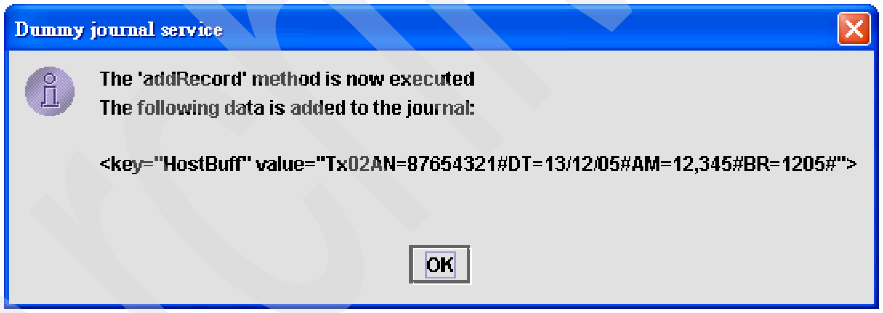
   A deposit page, as shown in Figure 8-163, opens.


deposit.jsp - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/signin/signIn.do
Deposit
Reset

AccountNumber 87654321
Amount 12345.00

Figure 8-163 The Deposit page

5. Input arbitrary Account Number and Amount, and then click **Deposit**. After the submission, you will see that the browser appears to be hanging for a while. The Dummy journal service dialog box opens as shown in Figure 8-164. Click **OK**.

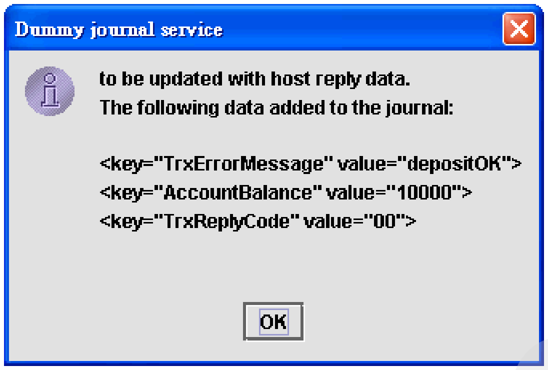


The 'addRecord' method is now executed
The following data is added to the journal:

<key="HostBuff" value="Tx02AN=87654321#DT=13/12/05#AM=12,345#BR=1205#">

Figure 8-164 The message from the addRecord method of the dummy journal service

6. In the next window shown in Figure 8-165, click **OK**.

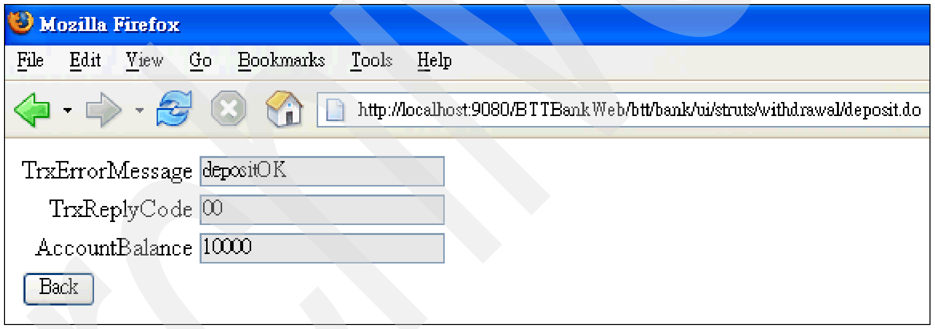


to be updated with host reply data.
The following data added to the journal:

```
<key="TrxErrorMessage" value="depositOK">
<key="AccountBalance" value="10000">
<key="TrxReplyCode" value="00">
```

Figure 8-165 The other message from the updateRecord method of the dummy journal service

The browser displays the final result as shown in Figure 8-166.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:9080/BTTBankWeb/btt/bank/ui/struts/withdrawal/deposit.do

TrxErrorMessage	depositOK
TrxReplyCode	00
AccountBalance	10000

Figure 8-166 The Deposit Result page

8.4 Developing a rich Java client

This section describes how to build the business application described in 8.3, “Developing an application using Branch Transformation Toolkit” on page 248. However, in this section, a Java client is used instead of a HTML client, to implement the presentation view. To invoke the Single Action EJB, an invoker for a Java request is used instead of invoker using Struts.

This section explores the characteristics of BTT Visual Beans to facilitate the development of operation views, besides exploring the characteristics of client operations and ithe nvoker.

This section shows how to create a view for the withdrawal operation and how the operation and view are related. It also details the creation of an invoker to implement the BeanInvokerForJavaRequest interface to invoke the Single Action EJB (SAE) created in 8.3, “Developing an application using Branch Transformation Toolkit” on page 248.

8.4.1 Rich Java client overview

Section 8.3.7, “Creating a Single Action EJB” on page 307, discussed client operation, invoker, and server SAE strategy. The client operation formated data and sent the context ID to the bean invoker, after which the invoker invoked a constant Single Action EJB to process the withdrawal action, and sent the reply data to the invoker. The invoker then sent the reply data to the client operation to display it in the Java client view.

You saw in 8.3, “Developing an application using Branch Transformation Toolkit” on page 248 a Single Action EJB named `WithdrawalServerOp` being built. Now, you have to only build the following components:

- Client operation
- Bean invoker for Java request
- Java client view using a DSE VisualBean

The architecture is shown in Figure 8-167.

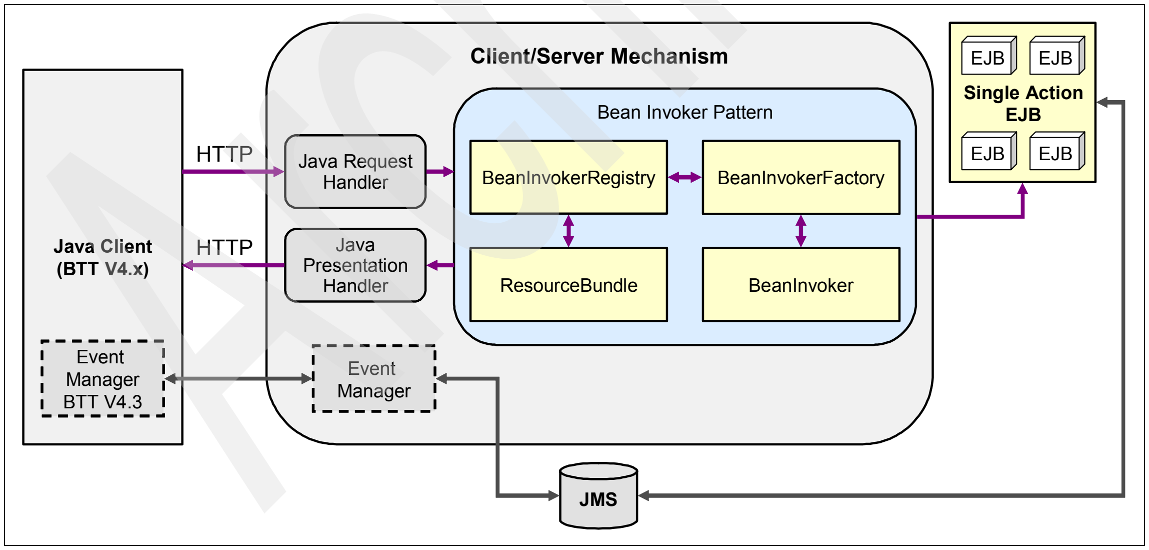


Figure 8-167 Architecture of our application

In this sample application:

1. The Java Client is launched and data is gathered from the Java Client. On clicking **OK**, the view launches the client operation defined in the DSE Operation properties of the View panel.
2. The client operation then sends the operation data to the server in serverOperation properties in the ClientOperation Externalize file dseoper.xml.
3. In the server, based on the definitions in com.ibm.btt.cs.invoker.base.BeanInvokerRegistryMapper.properties, use the key identified by the client operation's serverOperation attribute to find the corresponding property file and then instantiate the invoker defined in the property file.
4. In the server, use the parseRequestData(String requestData) method in the invoker to unformat the request data on to the SAE, which should have the corresponding method to get the request data.
5. Run the invoker to execute the SAE.
6. In the server, use the processRespondData(Object ejbResult) method in the invoker to format the reply data.
7. In the client, use the format identified by csReplyFormat to unformat the reply data on to the client operation contex, and then fire an OperationRepliedEvent.
8. When the event is fired, the operation panel refreshes the components inside the panel with the operation context.

Figure 8-168 shows the steps discussed.

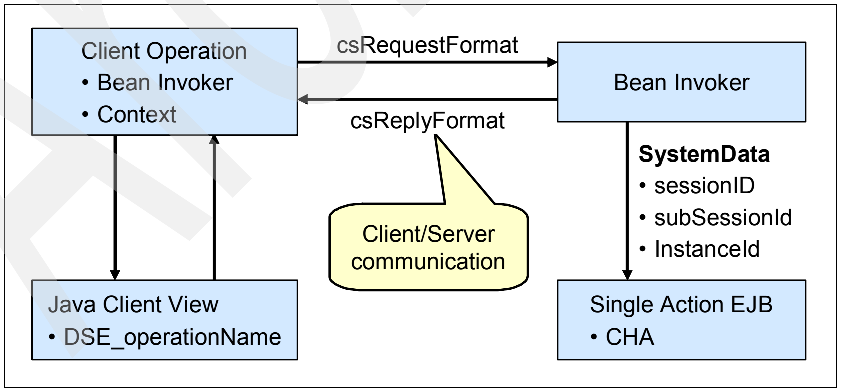


Figure 8-168 The application's architecture

8.4.2 Creating the client operation

The client operation is responsible for performing cross validation of the data received from the operation view, accessing local devices and sending the operation to the server. In our sample, the client withdrawal operation does not perform any extra processes in the client. It merely sends operation data to the server. The Java view, described in 8.4.3, “Creating Java client using VisualBeans” on page 442, launches this client operation.

To create client operation, perform the following tasks:

1. Open the **Java** perspective.
2. Highlight the **BTTBankApplicationClient** project in the **appClientModule** folder.
3. Select **File** → **Import** → **zip file.**
4. Import the **BTTBankAppClient.zip** file located in c:\7160code\chap8.

   If the file import is successful, the BTTBankApplicationClient project hierarchy looks similar to Figure 8-169.

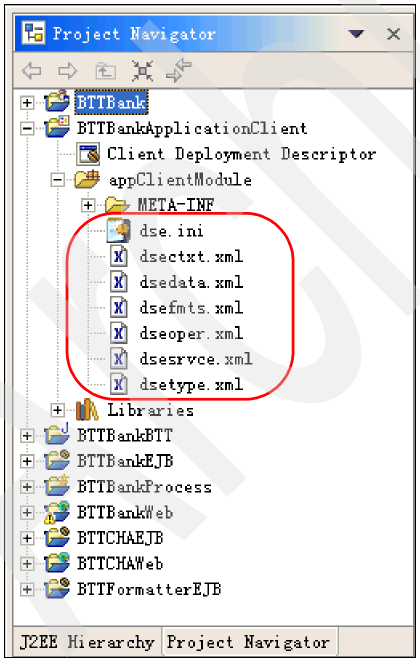
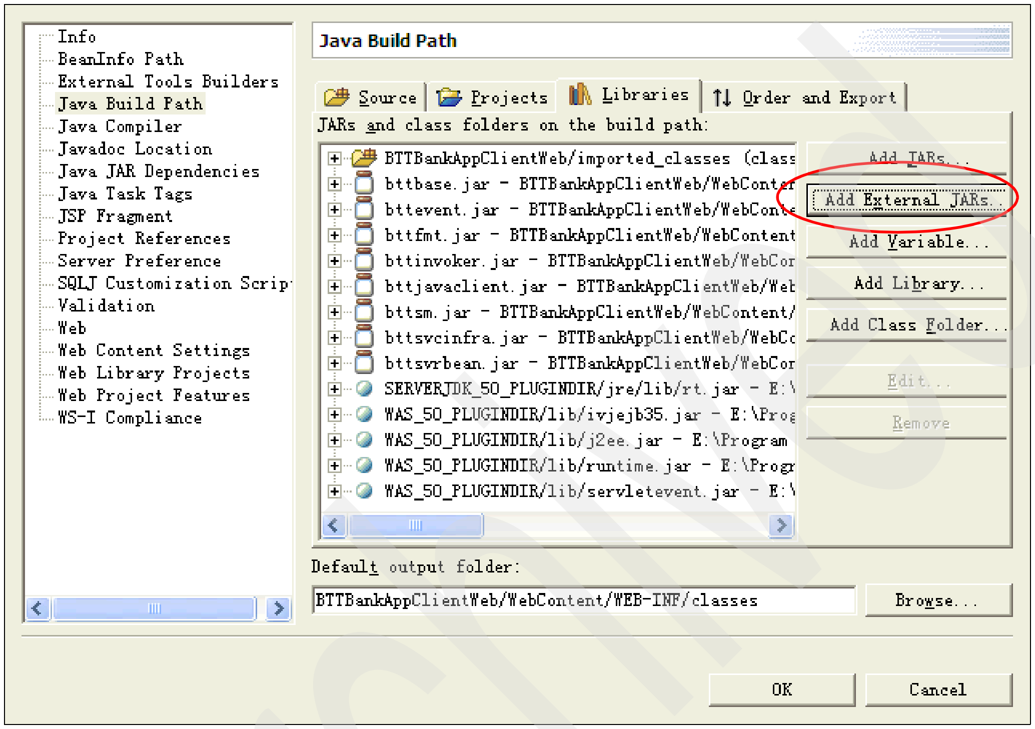


Figure 8-169 BTTBankApplicationClient After importing BTTBankAppClient.zip

5. Right-click the **BTTBankApplicationClient** project and click **Properties**.
   a. Select the **Java Build Path** list item in the Libraries tab.
   b. Click **Add External Jars**, as shown in Figure 8-170.



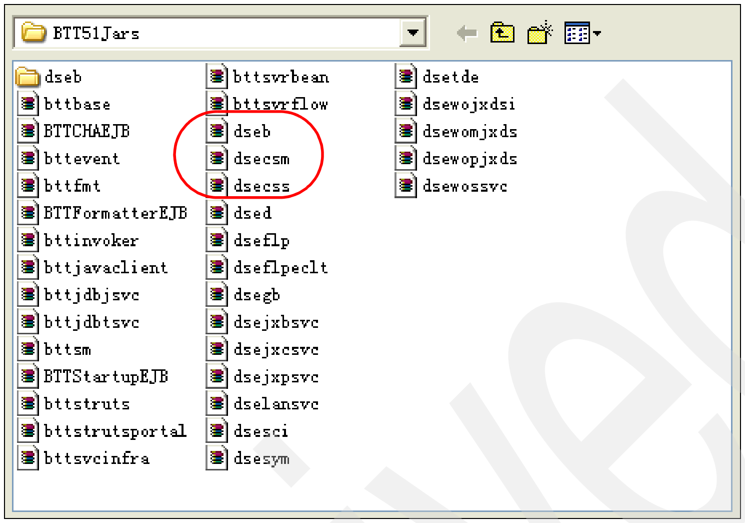
JARs and class folders on the build path:

```
BTTBankAppClientWeb/imported_classes (class
bttbase.jar - BTTBankAppClientWeb/WebConte
bttevent.jar - BTTBankAppClientWeb/WebConte
bttfmt.jar - BTTBankAppClientWeb/WebContent
bttinvoker.jar - BTTBankAppClientWeb/WebCor
bttjavaclient.jar - BTTBankAppClientWeb/Web
bttsm.jar - BTTBankAppClientWeb/WebContent/
bttsvcinfra.jar - BTTBankAppClientWeb/WebCo
bttsvrbean.jar - BTTBankAppClientWeb/WebCor
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - E:\
WAS_50_PLUGINDIR/lib/ivjejb35.jar - E:\Prog
WAS_50_PLUGINDIR/lib/j2ee.jar - E:\Program
WAS_50_PLUGINDIR/lib/runtime.jar - E:\Progr
WAS_50_PLUGINDIR/lib/servletevent.jar - E:\
```

Default output folder: BTTBankAppClientWeb/WebContent/WEB-INF/classes

Figure 8-170 Import external jars

c. Add the **dseb.jar**, **dsecss.jar**, and **dsecsm.jar** files from c:\7160code\chap8\BTT51Jars, as shown in Figure 8-171.



BTT51Jars

dseb	bttsvrbean	dsetde
bttbase	bttsvrflow	dsewojxdsi
BTTCHAEJB	dseb	dsewomjxds
bttevent	dsecsm	dsewopjxds
bttfmt	dsecss	dsewossvc
BTTFormatterEJB	dsed	
bttinvoker	dseflp	
bttjavaclient	dseflpeclt	
bttjdbjsvc	dsegb	
bttjdbtsvc	dsejxbsvc	
bttsm	dsejxcsvc	
BTTStartupEJB	dsejxpsvc	
bttstruts	dselansvc	
bttstrutsportal	dsesci	
bttsvcinfra	dsesym	

Figure 8-171 Import external jars

6. Click the **Create a Java Package** wizard in the toolbar, as shown in Figure 8-172.

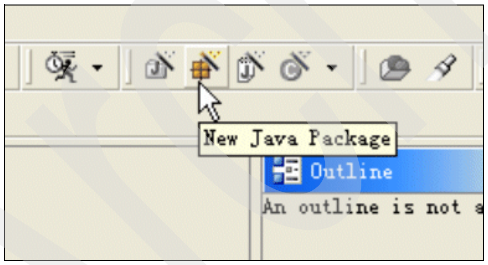
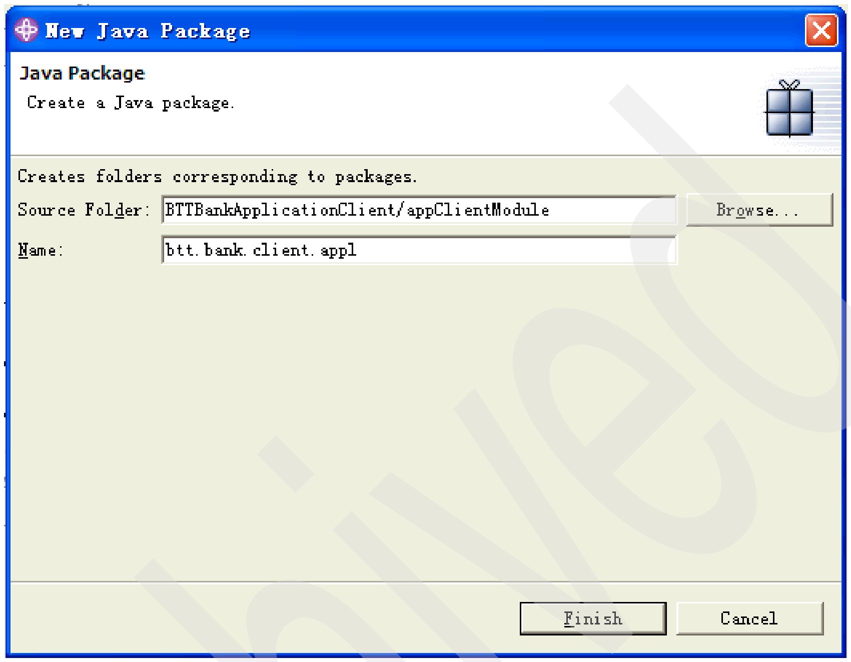


Figure 8-172 Create a new Java package named btt.bank.clientopertion

7. Name the package `btt.bank.client.appl` and click **Finish** as shown in Figure 8-173.



Java Package

Create a Java package.

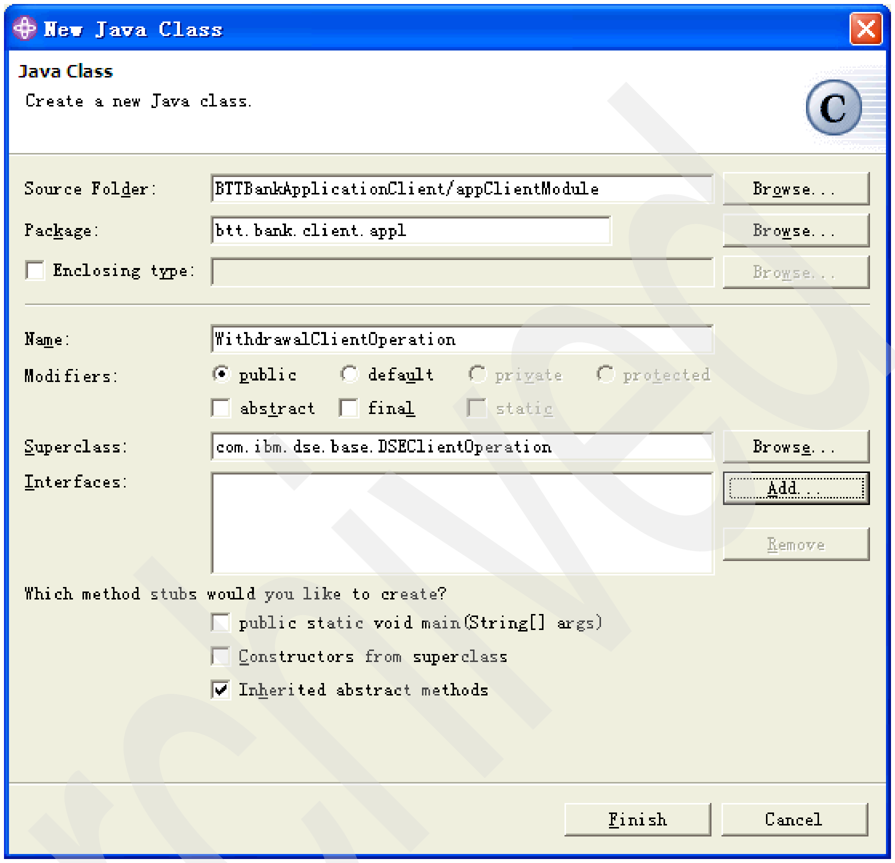
Creates folders corresponding to packages.

Source Folder: BTTBankApplicationClient/appClientModule

Name: btt.bank.client.appl

Figure 8-173 New package wizard

8. Create a new class named `WithdrawalClientOperation` extending com.ibm.dse.base.DSEClientOperation, as shown in Figure 8-174.



Source Folder: BTTBankApplicationClient/appClientModule

Package: btt.bank.client.appl

[ ] Enclosing type:

Name: WithdrawalClientOperation

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: com.ibm.dse.base.DSEClientOperation

Interfaces:

Which method stubs would you like to create?

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

[ ] Constructors from superclass

[✓] Inherited abstract methods

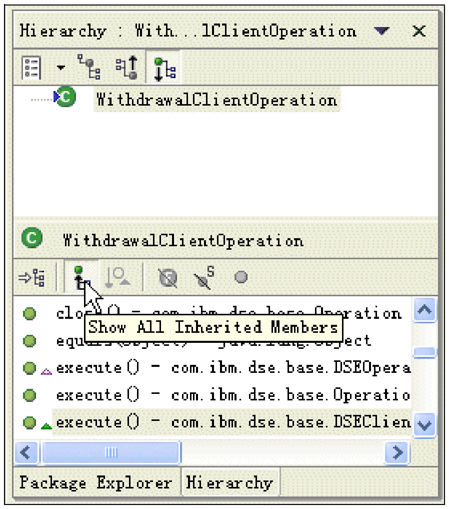
Figure 8-174 Create WithdrawalClientOperation

9. In the class definition, import the contents of com.ibm.dse.base.*, com.ibm.dse.clientserver, and com.ibm.dse.cs.servlet packages. These packages contain the base Branch Transformation Toolkit classes and the client/server support.

10. The WithdrawalClientOperation requires an execute() method. Perform the following tasks:

    a. Open **WithdrawalClientOperation** to its type hierarchy.

    b. In the Hierarchy view, click **Show All Inherited Members**, as shown in Figure 8-175 on page 439.


Hierarchy : With...lClientOperation
WithdrawalClientOperation
WithdrawalClientOperation
Show All Inherited Members
Package Explorer
Hierarchy

```
clo...() - com.ibm.dse.base.Operation
equ...(Object) - java.lang.Object
execute() - com.ibm.dse.base.DSEOpera
execute() - com.ibm.dse.base.Operatio
execute() - com.ibm.dse.base.DSEClien
```

Figure 8-175 Show all inherited members of WithdrawalClientOperation hierarchy

c. From the list of inherited members, right-click the **execute** method of DSEClientOperation and select **Override in 'WithdrawalClientOperation'**, as shown in Figure 8-176.

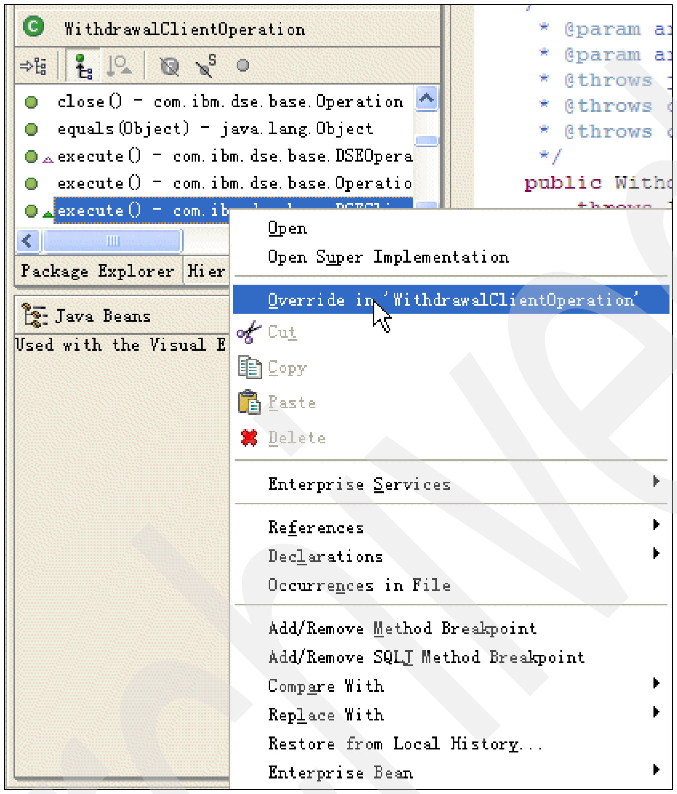

WithdrawalClientOperation
close() - com.ibm.dse.base.Operation
equals(Object) - java.lang.Object
execute() - com.ibm.dse.base.DSEOpera
execute() - com.ibm.dse.base.Operatio
execute() - com.ib
Package Explorer
Hier
Java Beans
Used with the Visual E
Open
Open Super Implementation
Override in 'WithdrawalClientOperation'
Cut
Copy
Paste
Delete
Enterprise Services
References
Declarations
Occurrences in File
Add/Remove Method Breakpoint
Add/Remove SQLJ Method Breakpoint
Compare With
Replace With
Restore from Local History...
Enterprise Bean

Figure 8-176 Override execute method in WithdrawalClientOperation

11. Write the **WithdrawalClientOperation.execute()** method, by performing the following tasks:

   a. Get the service named **CSClient**. This gets the client instance of client/server service from the operation's context hierarchy.

   b. Cast this service to CSClientService.

   c. Use the service to invoke the synchronous sendAndWait(ClientOperation, long) method. The first parameter is the client operation, "`this`". The second parameter is a time-out in milliseconds. An appropriate value can be 60000, or 60 seconds.

The entire WithdrawalClientOperation.java is shown in Example 8-37.

Example 8-37 WithdrawalClientOperation.java

```
/*
 * Created on 2005-12-6
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package btt.bank.client.appl;

import com.ibm.dse.base.*;
import com.ibm.dse.clientserver.*;
import com.ibm.dse.cs.servlet.*;
import com.ibm.dse.base.DSEClientOperation;

/**
 * @author add
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class WithdrawalClientOperation extends DSEClientOperation {

   /* (non-Javadoc)
    * @see com.ibm.dse.base.Operation#execute()
    */
   public void execute() throws Exception {
      // TODO Auto-generated method stub
      super.execute();
      ((CSClientService)getService("CSClient")).sendAndWait(this,6000);
   }

}
```

12. Save and close **WithdrawalClientOperation.java**.
13. Create a new class named `StartupClientOp` extending com.ibm.dse.base.DSEClientOperation in the package btt.bank.client.appl. As described in Step 8, override the execute method of DSEClientOperation in StartupClientop. Import the **com.ibm.dse.base.*** package and the **com.ibm.dse.clientserver.*** package. Implement the execute method of StartupClientOp as shown in Example 8-38.

Example 8-38 Execute method of StartupClientOp

```
public void execute() throws Exception {
   CSClientService csClientService=null;
   setValueAt("TID",Settings.getTID());//$NON-NLS-1$
   csClientService=((CSClientService)getService("CSClient"));//$NON-NLS-1$
   setValueAt("permanentConnectionForEvents",(new
Boolean(csClientService.getPermanentConnectionForEvents())).toString());
   setValueAt("ipAddress",Settings.getIpAddress());
   setValueAt("eventsPort",new Integer(csClientService.getEventsPort()));
   csClientService.sendAndWait(this,40000);
}
```

14. Externalize the client operation data by performing the following tasks:
    a. Open the **dseoper.xml** file.
    b. Create your client operation with the values shown in Example 8-39.

Example 8-39 withdrawalClientOperation

```
<operation context=”withdrawalClinetCtx”
id=”withdrawalClientOp”
impClass=”btt.bank.client.appl.WithdrawalClientOperation”
serrverOperation=”withdrawalServerOp”>
<refFormat name=”csRequestFormat” refId=”withdrawalCSRequestFmt”/>
</operation>
```

8.4.3 Creating Java client using VisualBeans

This section describes how to create a Java client view to launch the client operation.

To construct the Java client view, use DSE VisualBeans and some standard WebSphere Studio Application Developer visual parts such as Swing components. Using Branch Transformation Toolkit VisualBeans in the WebSphere Studio Application Developer environment requires a simple setup to be in place beforehand.

This section first looks at setting up the visual composition environment, and then at creating a Java View named `WithdrawalView`.

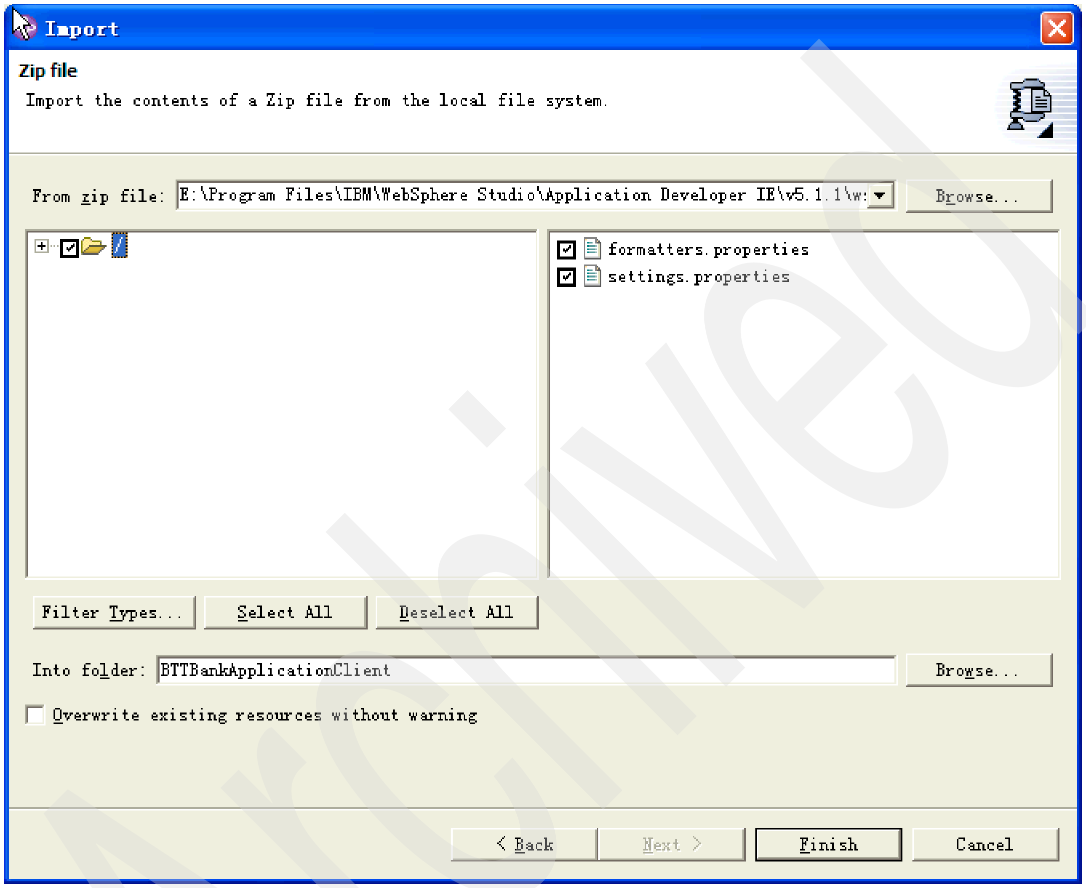
Setting up the WebSphere Studio environment

.To set up DSE VisualBeans environment, perform these tasks:

1. Open WebSphere Studio Application Developer.
2. Open the **Java** perspective and select **BTTBankApplicationClient** → **appClientModule**.
    a. Select **File** → **Import** → **ZIP file** → **Next**.
    b. Import the **DseGuiBeans.zip** file located in the src directory of the visual beans plug-in, that is, [drive]:\[WebSphere Studio Application

Developer_install_dir]\wstools\eclipse\plugins\com.ibm.dse.guibeans_5.1.0\src).

c. Click **Select All** and then **Finish**, as shown in Figure 8-177.


Import

Zip file

Import the contents of a Zip file from the local file system.

From zip file: E:\Program Files\IBM\WebSphere Studio\Application Developer IE\v5.1.1\w:

☑ /

☑ formatters.properties

☑ settings.properties

Into folder: BTTBankApplicationClient

☐ Overwrite existing resources without warning

Figure 8-177 Import DSEGuiBeans into BTTBankApplicationClient

d. Select the **SPColorEditor.Java** class in the com.ibm.dse.gui package. Right-click **SPColorEditor.Java** and select **Source** → **Organize Imports**. Save and close.

3. The GUI beans have to access information from your client's initialization file. Therefore, make sure that dse.ini is visible:

a. Open the **BTTBankApplicationClient->appClinetModule** project in the settings.properties file.

b. Check the **DSEINIDevelopmentPath** property. It must point to the dse.ini file in your client project workspace, for example:

```
DSEINIDevelopmentPath =
[WORKSAPCE_DIR]\\BTTBankApplicationClient\\appClientModule\\dse.ini
```

c. Open the **BTTBankApplicationClient** project in the appClientModule folder in the dse.ini file. Modify the **entities** property in your dse.ini file so that your visual beans can access the proper externalized XML files, for example:

```
<field id="entities" value=".\"/>
```

4. In the **Java** perspective, right-click the **BTTBankApplicationClient** project and select **Properties** → **Java Build Path** → **Libraries** tab.

   a. Click **Add Library**. This calls the Add library dialog box.

   b. Select **DSE VisualBeans**.

   See Figure 8-178.

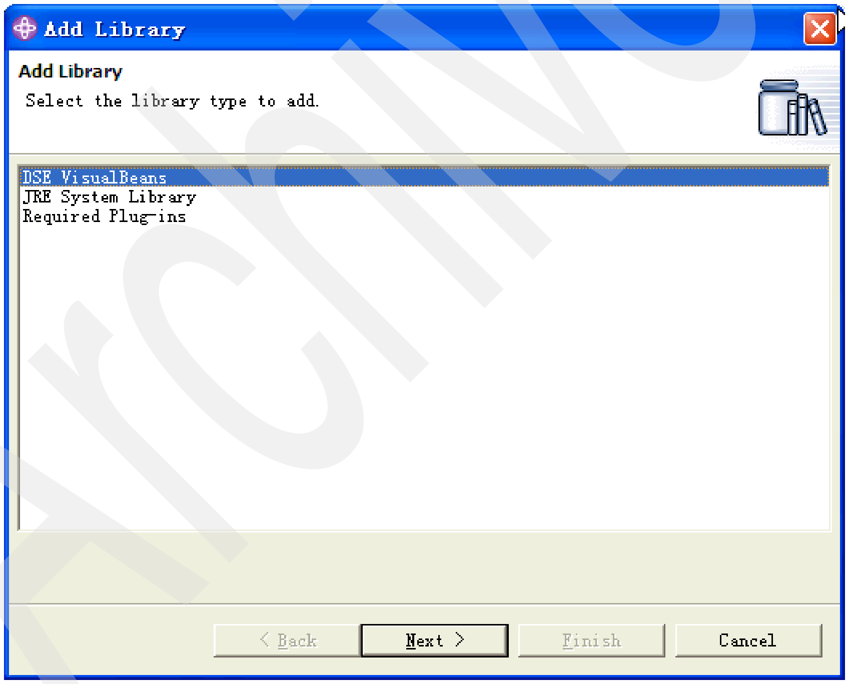
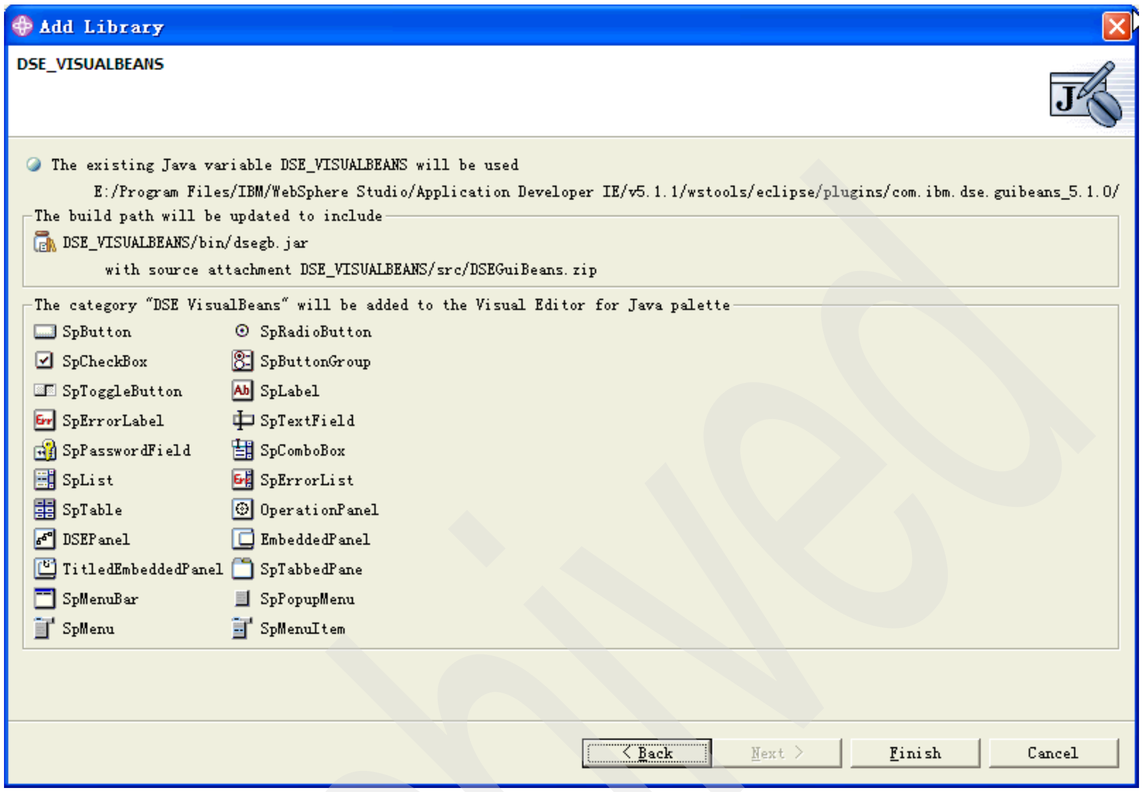


Figure 8-178 Add DSE VisualBeans to application

   c. Click **Next**. The new library of Visual Components is displayed as shown in Figure 8-179 on page 445.

   d. Click **Finish**.



The existing Java variable DSE_VISUALBEANS will be used
E:/Program Files/IBM/WebSphere Studio/Application Developer IE/v5.1.1/wstools/eclipse/plugins/com.ibm.dse.guibeans_5.1.0/

The build path will be updated to include
DSE_VISUALBEANS/bin/dsegb.jar
with source attachment DSE_VISUALBEANS/src/DSEGuiBeans.zip

The category "DSE VisualBeans" will be added to the Visual Editor for Java palette

SpButton	SpRadioButton
SpCheckBox	SpButtonGroup
SpToggleButton	SpLabel
SpErrorLabel	SpTextField
SpPasswordField	SpComboBox
SpList	SpErrorList
SpTable	OperationPanel
DSEPanel	EmbeddedPanel
TitledEmbeddedPanel	SpTabbedPane
SpMenuBar	SpPopupMenu
SpMenu	SpMenuItem

Figure 8-179 Add Library

e. Click **OK**. The Add Library Wizard adds a container to the build path. When the you open the **Java Visual Editor**, the categories defined in the XMI file will be added at the top of the palette. You can then drop **JavaBeans** into the visual canvas without having to use the VisualEditor's Choose Bean dialog.

f. If there are any problems with your plug-in, such as not appearing in the list of containers for the project, it may mean that errors have occurred in the plug-in's XML file. WebSphere Studio Application Developer writes these errors in the [WebSphere Studio Application Developer_install_dir]\workspace\.metadata\.log file.

Creating the operation view

This section describes how to create a Java view to launch the operation. To perform the operation, you should provide some input data such as the branch identifier, the account number, the operation date, and the amount. You should expect the new balance and a transaction error message or just the transaction error message to be returned.

The view you create now will appear as shown in the Figure 8-180. It can be personalized.

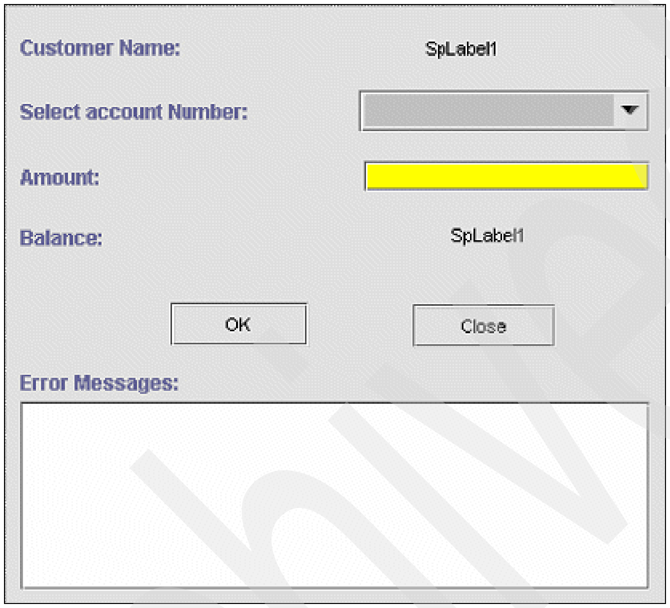
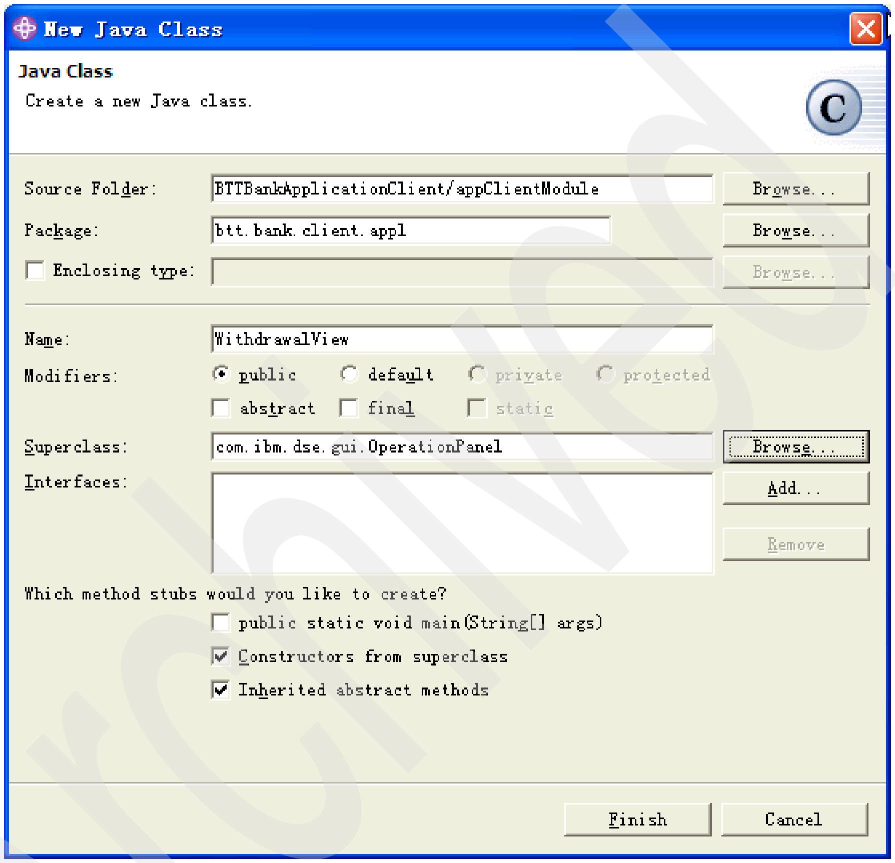


Figure 8-180   WithdrawalView

To build the view that can manage the data, perform the following tasks:

1. In BTTBanckApplicationClient in appClientModule in btt.bank.client.appl package, create a new class named `WithdrawalView`, as shown in Figure 8-181, and which extends com.ibm.dse.gui.OperationPanel.



New Java Class

Java Class

Create a new Java class.

Source Folder:	BTTBankApplicationClient/appClientModule
Package:	btt.bank.client.appl
☐ Enclosing type:	
Name:	WithdrawalView
Modifiers:	● public ○ default ○ private ○ protected
	☐ abstract ☐ final ☐ static
Superclass:	com.ibm.dse.gui.OperationPanel
Interfaces:	

Which method stubs would you like to create?

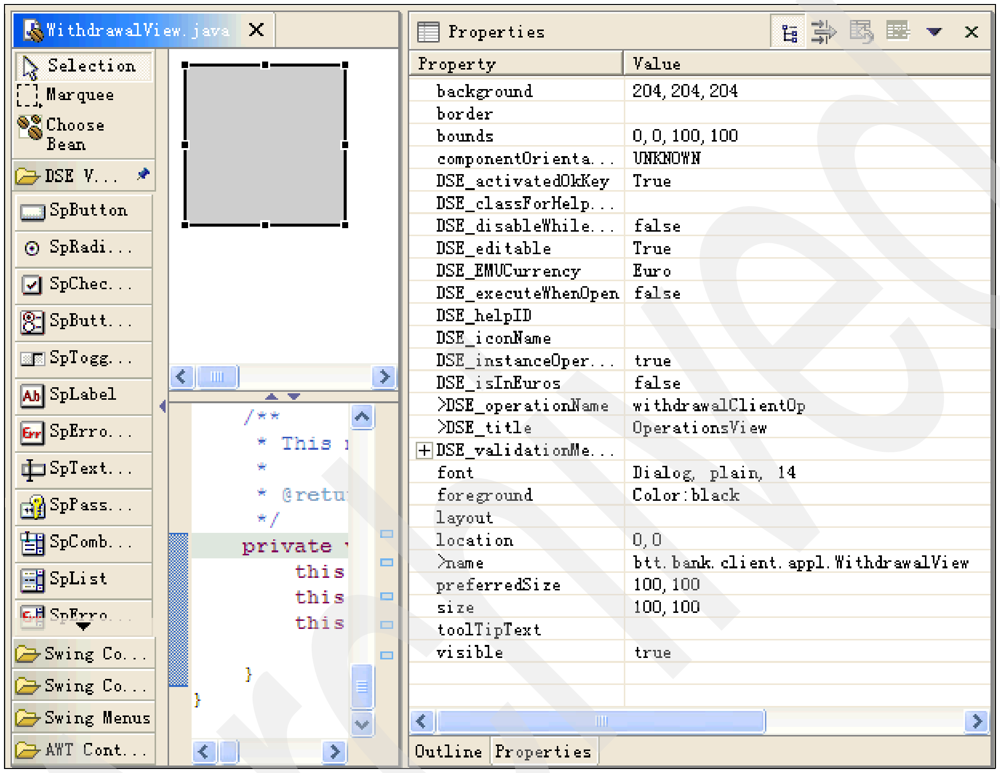
- ☐ public static void main(String[] args)
- ✓ Constructors from superclass
- ✓ Inherited abstract methods

Figure 8-181 Create WithdrawalView

2. Open the class to visual composition by right-clicking **BTTBankApplicationClient** and selecting **appClientModule** → **btt.bank.client.appl** → **WithdrawalView**, and selecting **Open With** → **Visual Editor**. Make sure this class is not open in any other Java editor. Close that editor, if needed. Once opened, all the properties belonging to a visual element selected is displayed in the right-hand side of the Properties view. Select the operation panel, which is now a plain, gray box, and set the

following properties, as shown in Figure 8-182 on page 448 and Figure 8-183 on page 449:

– Name: `btt.bank.client.appl.WithdrawalView`.
– DSE_operationName: `withdrawalClientOp`
– DSE_title: `OperationsView`



```
/**
 * This i
 *
 * @retu
 */
private
    this
    this
    this
}
}
```

Property	Value
background	204, 204, 204
border	
bounds	0, 0, 100, 100
componentOrienta...	UNKNOWN
DSE_activatedOkKey	True
DSE_classForHelp...	
DSE_disableWhile...	false
DSE_editable	True
DSE_EMUCurrency	Euro
DSE_executeWhenOpen	false
DSE_helpID	
DSE_iconName	
DSE_instanceOper...	true
DSE_isInEuros	false
>DSE_operationName	withdrawalClientOp
>DSE_title	OperationsView
+ DSE_validationMe...	
font	Dialog, plain, 14
foreground	Color:black
layout	
location	0, 0
>name	btt.bank.client.appl.WithdrawalView
preferredSize	100, 100
size	100, 100
toolTipText	
visible	true

Figure 8-182 Set name to btt.bank.client.appl.WithdrawalView

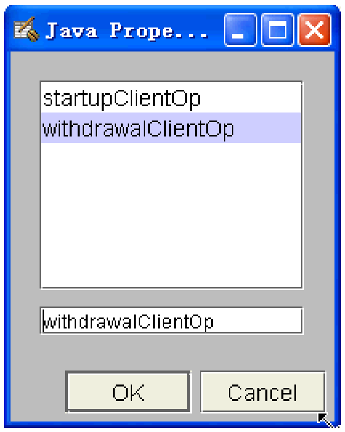


Figure 8-183   Set DSE_operationName

3. You can see a palette of DSE Visual Beans on the left side of the Visual Composition Editor, as shown in Figure 8-184.

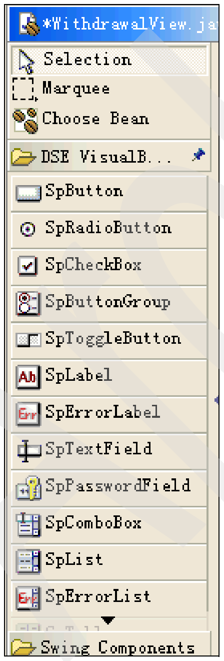
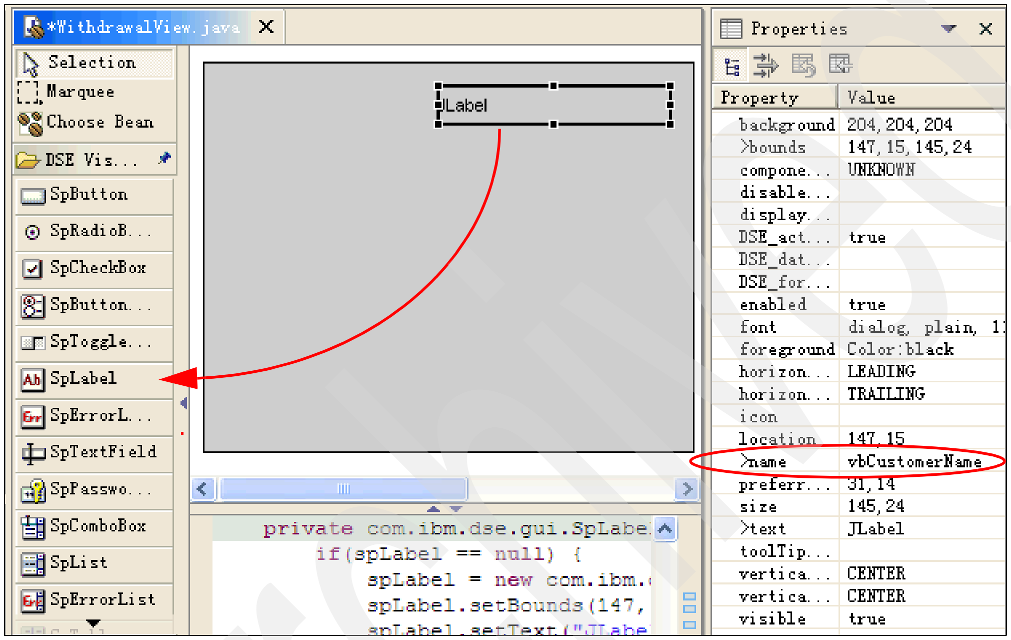


Figure 8-184   DSE VisualBeans

> **Note:** If you do not have or cannot find this palette of beans, construct your view using the Choose Bean item.

4. Add an **SpLabel** to represent the Customer Name field as shown in Figure 8-185.

   – Change the name to `vbCustomerName`.

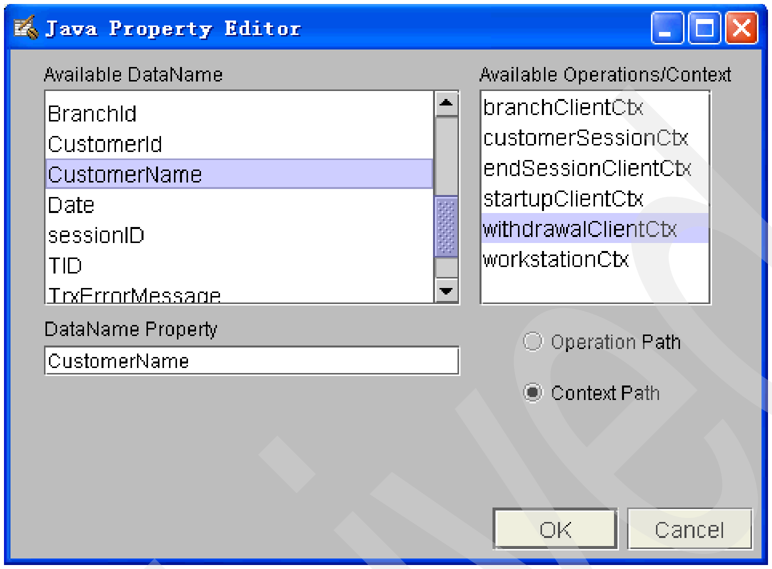


Property	Value
background	204, 204, 204
>bounds	147, 15, 145, 24
compone...	UNKNOWN
disable...	
display...	
DSE_act...	true
DSE_dat...	
DSE_for...	
enabled	true
font	dialog, plain, 1
foreground	Color:black
horizon...	LEADING
horizon...	TRAILING
icon	
location	147, 15
>name	vbCustomerName
preferr...	31, 14
size	145, 24
>text	JLabel
toolTip...	
vertica...	CENTER
vertica...	CENTER
visible	true

```
    private com.ibm.dse.gui.SpLabe
        if(spLabel == null) {
            spLabel = new com.ibm.
            spLabel.setBounds(147,
            spLabel.setText("JLabe
```

Figure 8-185 Add a SpLabel to operationPanel and change name

   – Set the DSE_dataName property from the Context Path to `CustomerName`. In the Properties view, click **DSE_dataName Item**. A dialog box as shown in Figure 8-186 opens. Select the **Context Path** radio button, **withdrawalClientCtx** item, and **CustomerName**.



Available DataName

BranchId
CustomerId
CustomerName
Date
sessionID
TID
TrxErrorMessage

Available Operations/Context

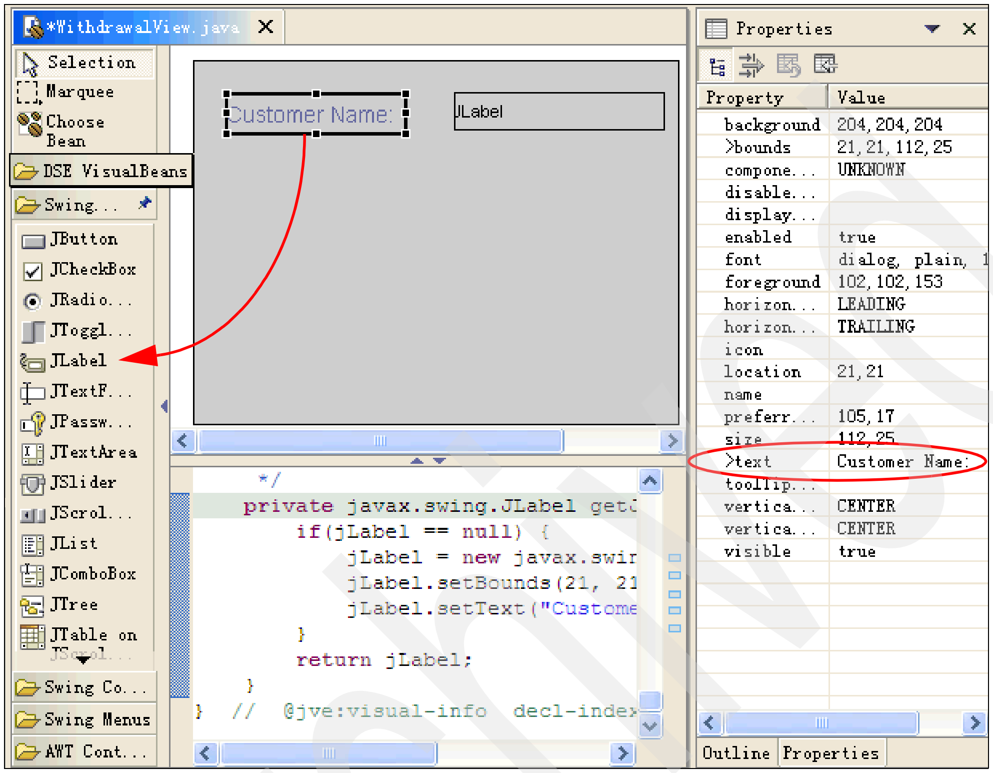
branchClientCtx
customerSessionCtx
endSessionClientCtx
startupClientCtx
withdrawalClientCtx
workstationCtx

DataName Property: CustomerName

Operation Path [✓ Context Path]

Figure 8-186 Set DSE_DataName

– Add a **javax.swing.JLabel** for the previous SpLabel. Change the text property to `Customer Name`, as shown in Figure 8-187.



```
        */
    private javax.swing.JLabel getJ
        if(jLabel == null) {
            jLabel = new javax.swir
            jLabel.setBounds(21, 21
            jLabel.setText("Custome
        }
        return jLabel;
    }
}  //  @jve:visual-info  decl-index
```

Property	Value
background	204, 204, 204
>bounds	21, 21, 112, 25
compone...	UNKNOWN
disable...	
display...	
enabled	true
font	dialog, plain, 1
foreground	102, 102, 153
horizon...	LEADING
horizon...	TRAILING
icon	
location	21, 21
name	
preferr...	105, 17
size	112, 25
>text	Customer Name:
tooltip...	
vertica...	CENTER
vertica...	CENTER
visible	true

Figure 8-187 Add JLabel to OperationPanel and change text

5. Add **SpComboBox** to represent the selection of Account Numbers.
   – Change the name to `vbAccountName`.
   – Set the DSE_dataName property from the context path to `AccountNumber`.
   – Set the DSE_dataNameForList property to `accountListData`.
6. Add a **javax.swing.JLabel** for the previous SpComboBox. Change the text property to `Select Account Number:`.
7. Add a **SpTextField** to represent the Amount field:
   – Change the name to `vbAmount`.
   – Set the DSE_mandatory attribute to `true`.
   – Set the DSE_autoTab attribute to `true`.
   – Set the DSE_maxChars attribute to `5`.
   – Set the DSE_dataName to `Amount`.

- Set the DSE_formatter to `FloatFormatter`. Inside the DataFormatter attribute specifications, set the appropriate error message you want the system to display when a validation error occurs. You can also set the default thousands/decimal separators.

8. Add a **javax.swing.JLabel** for the previous SpTextField. Change the text property to `Amount:`.
9. Add a **SpLabel** to represent the new balance after performing the operation.
   - Change the name to `vbAccountBalance`.
   - Set the DSE_dataName attribute to `AccountBalance`.
10. Add a **javax.swing.JLabel** for the previous SpLabel. Change the text property to `Balance:`.
11. Add a **com.ibm.dse.gui.SpErrorList** to the OperationPanel. Change the beanName to `vbErrorList`. This bean will display validation errors during the data input process that occurs in any of the OperationPanel components, including withdrawal validation class messages. Make sure your error list is large enough to display long messages. Multiple error messages can be displayed.
12. Add a **javax.swing.JLabel** for the previous SpErrorList. Change the text property to `Error messages:`.
13. Add an **SpButton**.
   - Change the beanName name to `vbWithdrawal`.
   - Set the text attribute to `OK`.
   - Set the DSE_type to `OK`. When you click **OK**, it executes the withdrawal operation.
14. Save the bean and run it. Although the view does not really do anything at this point, notice that when the mandatory fields are missing or when the formats are not correct, **OK** is disabled.
15. Add another **SpButton**.
   - Change the beanName to `vbClose`.
   - Set the text attribute to `Close`.
   - Set the DSE_type to `Close`. When you click **Close**, it will close the operation view.
16. A summary of the VisualBeans to be added to your view is shown in Table 8-8 and Table 8-9.

Table 8-8 DSE components

DSE visual beans	Properties
SpLabel	beanName:`vbCustomerName` DSE_dataName:`CustomerName`

DSE visual beans	Properties
SpComboBox	beanName:vbAccountNumber DSE_dataName:AccountNubmer DSE_dataNameForList:accountListData
SpTextField	beanName:vbAmount DSE_dataName:Amount DSE_mandatory:true DSE_autoTab:true DSE_maxChar:5 DSE_formatter:FloatFormatter
SpLabel	beanName:vbAccountBalance DSE_dataName:AccountBalance
SpErrorList	beanName:vbErrorList
SpButton	beanName:vbWithdrawal text:OK DSE_type:ok
SpButton	beanName:vbClose text:Close DSE_type:close

Table 8-9 Swing components

Swing beans	Properties
JLabel	text:Customer Name:
JLabel	text:Select Account Nubmer:
JLabel	text:Amount:
JLabel	text:Available Balance:
JLabel	text:Error Messages:

8.4.4 Connecting the view to the operation

To link the client operation to the view, the following tasks need to be complete:

- Defining the fields in view, that will interact with the operation. This task was completed when you set the DSE_dataName properties of the components in the OperationPanel.
- Setting the operation attribute of the view with the client operation instance.This too was done when you specified the DSE_operationName property of the OperationPanel.

- Notifying the view when the operation execution is finished. You may have noticed that when carrying out visual composition, you were setting the attribute DSE_dataName of the input and output components as the ID of the fields from the operation context. This is all that you require to define which fields will interact with the operation context.

In order to create a client operation instance, initialize the Branch Transformation Toolkit environment. In this sample, since you are working in a client/server environment, initialize the complete Branch Transformation Toolkit client environment. To do this, use the initialize() method of the view that has been automatically created.

1. Navigate to the class declaration of the **WithdrawalView** (right-click it and select **Open With** → **Java Editor**), add a statement to import the contents of the com.ibm.dse.base package, as shown in Example 8-40.

Example 8-40 Add import class

```
...
import java.io.BufferedInputStream;
import com.ibm.btt.dse.base.*;
import com.ibm.dse.clientserver.CSClientService;
...
```

2. Create a new method named `initEnv()` in the class WithdrawalView that throws Exception. This method initializes the entire Branch Transformation Toolkit client environment.The source code is shown in Example 8-41.

Example 8-41 Add method initEnv

```
...
   private void initEnv() throws Exception{

      String iniPath =
"http://127.0.0.1:9080/BTTBankAppClientWeb/dse/dse.ini";//$NON-NLS-1$
      Context.reset();
      Settings.reset(iniPath);
      Settings.initializeExternalizers(Settings.MEMORY);
      Context context= (Context)Context.readObject("workstationCtx");
      System.out.println(context);
      System.out.println(context.getService("CSClient"));
      ((CSClientService) context.getService("CSClient")).establishSession();
      ClientOperation anOp =
(ClientOperation)DSEOperation.readObject("startupClientOp");
      anOp.execute();
   }
...
```

3. Add a call to this method in initialize() at the top of the method. See Example 8-42.

Example 8-42   Add method initEnv() into initialize()

```
//user code begin {1}
try{
   initEnv();
}catch(Exception e){
   System.out.println(e.toString());
}
//user code end
```

4. Use the refresh method to update the visual components at the end of the method. See Example 8-43.

Example 8-43   Add refresh method into initialize method

```
// user code begin {2}
refresh();
// user code end
```

This step is necessary if you require interaction between the view components and the operation when the operation execution is completed. In this sample, the view should get the AccountBalance and the TrxErrorMessage values when the operation finishes.

After you have completed step 3 and step 4, the initialize() method will look as shown in Example 8-44.

Example 8-44   Initialize method

```
   private void initialize() {
      //user code begin {1}
      try {
         initEnv();
      } catch (Exception e) {
         // TODO Auto-generated catch block
         System.out.println(e.toString());
      }
      //user code end
      this.add(getSpLabel(), null);
      this.add(getJLabel(), null);
      this.add(getJLabel1(), null);
      this.add(getSpComboBox(), null);
      this.add(getJLabel2(), null);
      this.add(getJLabel3(), null);
      this.add(getJLabel4(), null);
      this.add(getSpTextField(), null);
      this.add(getSpButton(), null);
```

```
        this.add(getSpButton1(), null);
        this.add(getSpErrorList(), null);
        this.add(getSpLabel1(), null);
        this.setSize(342, 276);
        this.setName("labs.WithdrawalView");
        this.setOperationName("withdrawalClientOp");
        this.setTitle("OperationsView");

        //user code begin {2}
        refresh();
        //user code end

    }
```

5. Make the operation fire an event when the execution is complete. To accomplish this, add the following line at the end of the client operation (btt.bank.client.appl.WithdrawalClientOperation) execute method:

   ```
   fireHandleOperationRepliedEvent(new OperationRepliedEvent(this));
   ```

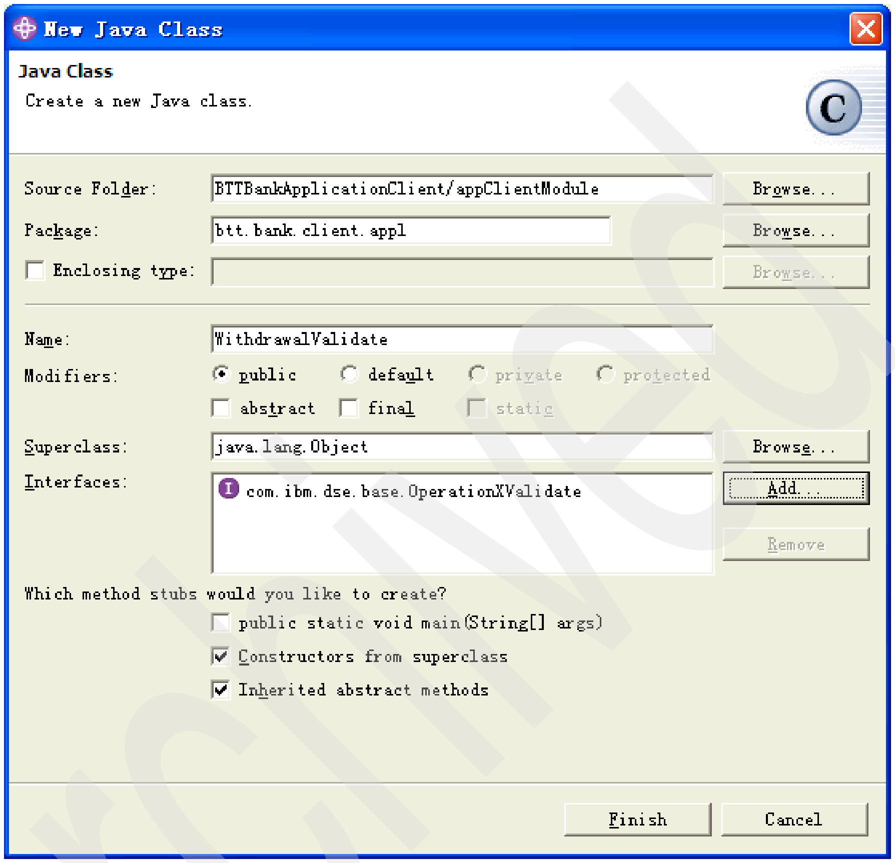
   The operation panel knows that when it executes the operation, that is, when you click **OK**, it has to wait for this event. When the event is fired, the operation panel refreshes the components inside the panel with the operation context.

8.4.5 Displaying the operation messages

This step is useful if you want to display the error messages of the business operation. The messages written in the WithdrawalValidate class will be displayed in the bean SpErrorList.

1. Select **Java** perspective → **BTTBankApplicationClient** → **appClientModule** → **btt.bank.client.appl** package. In this package, create a

class named `WithdrawalValidate implementing com.ibm.dse.base.OperationXValidate`, as shown in Figure 8-188.


New Java Class

Java Class
Create a new Java class.

Source Folder: BTTBankApplicationClient/appClientModule
Package: btt.bank.client.appl
☐ Enclosing type:

Name: WithdrawalValidate
Modifiers: [✓ public] default private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces: com.ibm.dse.base.OperationXValidate

Which method stubs would you like to create?
☐ public static void main(String[] args)
☑ Constructors from superclass
☑ Inherited abstract methods

Figure 8-188 Create WithdrawalVlidate class

2. To implement this interface, use the xValidate(Context) method. Your xValidate(Context arg0) code should:
   a. Declare a string array that contains the error messages and declare a float variable, as shown in Example 8-45.

Example 8-45 Declare a string array to contain error messages

```
String [] errorMessages = new String[5];
java.util.Date date = new java.util.Date();
Float amount = new Float("0");
```

   b. Set the data value in the context as shown in Example 8-46 on page 459.

Example 8-46   Set data value in the context

```
try {
   arg0.setValueAt("Date", date);
}catch(Exception e){
   errorMessages[0] = "Invalid date";
   errorMessages[1] = "Date must be a valid dd/mm/yy date";
   return errorMessages;
}
```

   c. Get the Amount value from the context and if the value is true, return null. However, if there is an exception, return the error messages list. See Example 8-47.

Example 8-47   Get amount value

```
try{
   String strAmount=((Object) arg0.getValueAt("Amount")).toString();
   amount = new Float(strAmount);
   if (amount.intValue() > 5000) {
      errorMessages[0] = "Field amount greater than 5000";
      errorMessages[1] = "Input a new amount";
      return errorMessages;
   }
}catch(Exception e){
   errorMessages[0] = "Invalid field";
   return errorMessages;
}

return null;
```

3. The WithdrawalValidation class will look as shown in Example 8-48.

Example 8-48   WithdrawalValidation.java

```
/*
 * Created on 2005-12-7
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package btt.bank.client.appl;

import com.ibm.dse.base.Context;
import com.ibm.dse.base.DataField;
import com.ibm.dse.base.OperationXValidate;
import com.ibm.dse.base.types.DSETypeException;

/**
```

```
 * @author addd
 *
 * To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class WithdrawalValidate implements OperationXValidate {

   /**
    *
    */
   public WithdrawalValidate() {
      super();
      // TODO Auto-generated constructor stub
   }

   /* (non-Javadoc)
    * @see
com.ibm.dse.base.OperationXValidate#xValidate(com.ibm.dse.base.Context)
    */
   public String[] xValidate(Context arg0) {
      String [] errorMessages = new String[5];
      java.util.Date date = new java.util.Date();
      Float amount = new Float("0");
      try {
         arg0.setValueAt("Date", date);
      }catch(Exception e){
         errorMessages[0] = "Invalid date";
         errorMessages[1] = "Date must be a valid dd/mm/yy date";
         return errorMessages;
      }
      try{
         String strAmount=((Object) arg0.getValueAt("Amount")).toString();
         amount = new Float(strAmount);
         if (amount.intValue() > 5000) {
            errorMessages[0] = "Field amount greater than 5000";
            errorMessages[1] = "Input a new amount";
            return errorMessages;
         }
      }catch(Exception e){
         errorMessages[0] = "Invalid field";
         return errorMessages;
      }

      // TODO Auto-generated method stub
      return null;
   }

   /* (non-Javadoc)
```

```
     * @see com.ibm.dse.base.OperationXValidate#validate(java.lang.String,
com.ibm.dse.base.DataField, com.ibm.dse.base.Context)
     */
    public void validate(String arg0, DataField arg1, Context arg2)
       throws DSETypeException {
       // TODO Auto-generated method stub

    }

}
```

4. Modify **withdrawalClientOperation** defined in **Java Perspective** → **BTTBankApplicationClient** → **appClientModule** → **dseoper.xml** to include the reference to this validation class. See Example 8-49.

Example 8-49 withdrawalClientOperation

```
<operation
   context="withdrawalClientCtx"
   id="withdrawalClientOp"
   serverOperation="withdrawalServerOp"
   impClass="btt.bank.client.appl.WithdrawalClientOperation"
   xVal="labs.WithdrawalValidate">
   ......
```

8.4.6 Creating the invoker

Instantiated by the Bean Invoker Factory, invokers enables Struts actions or Java request handlers to access business processes and Single Action EJBs through an EJB call.

When a request comes from a requester, which could be a request handler or a Struts action, the request brings a request ID and a session ID to the Bean Invoker Factory. The request ID indicates what kind of transaction the client is requesting, and the session ID identifies the session of this transaction request. The Bean Invoker Factory generates or allocates an invoker with the request ID and session ID. The Bean Invoker Factory then returns the invoker to requester so that the requester can send the request to the application logic layer.

Bean invoker pattern

This section describes some common ways that you can work with invoker beans.

- Invoker bean overview
  - EJB methods are strongly typed and each EJB has its own corresponding access information. In order to provide a generic way to access EJB, invocation architecture is required.
  - The logic for creating Bean Invoker and bean invocation logic are encapsulated within the Bean Invoker. Developers may invoke EJB through a Bean Invoker with a generic interface.
  - The composition of Bean Invoker Pattern includes Bean Invoker Factory, Basic Invoker Class (super class), end-user extended Bean Invoker class and Bean Invoker Registry.

  The Bean Invoker Pattern is shown in Figure 8-189.

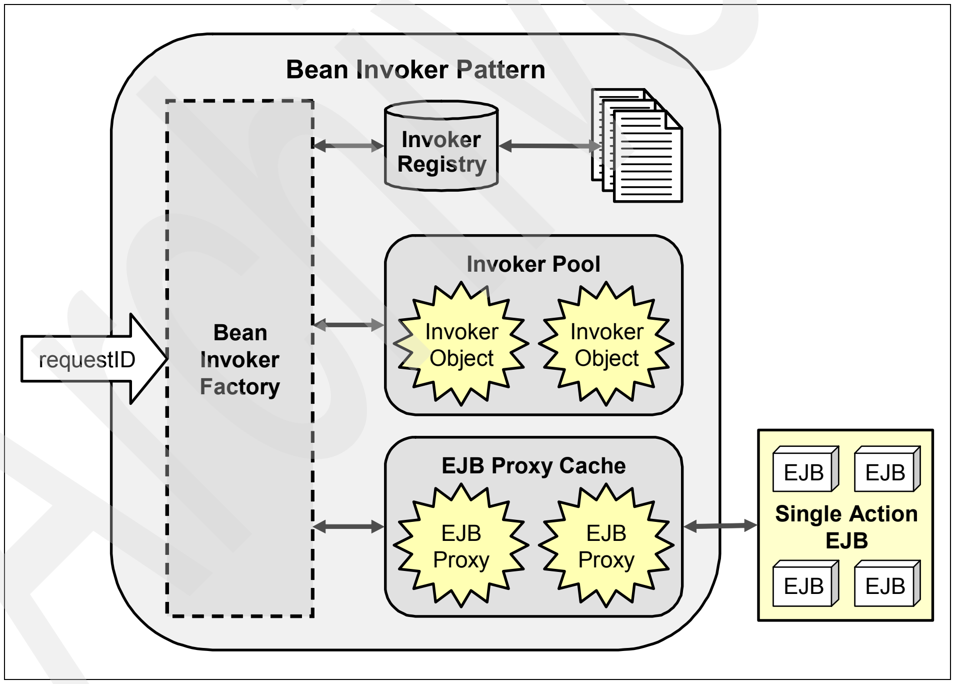
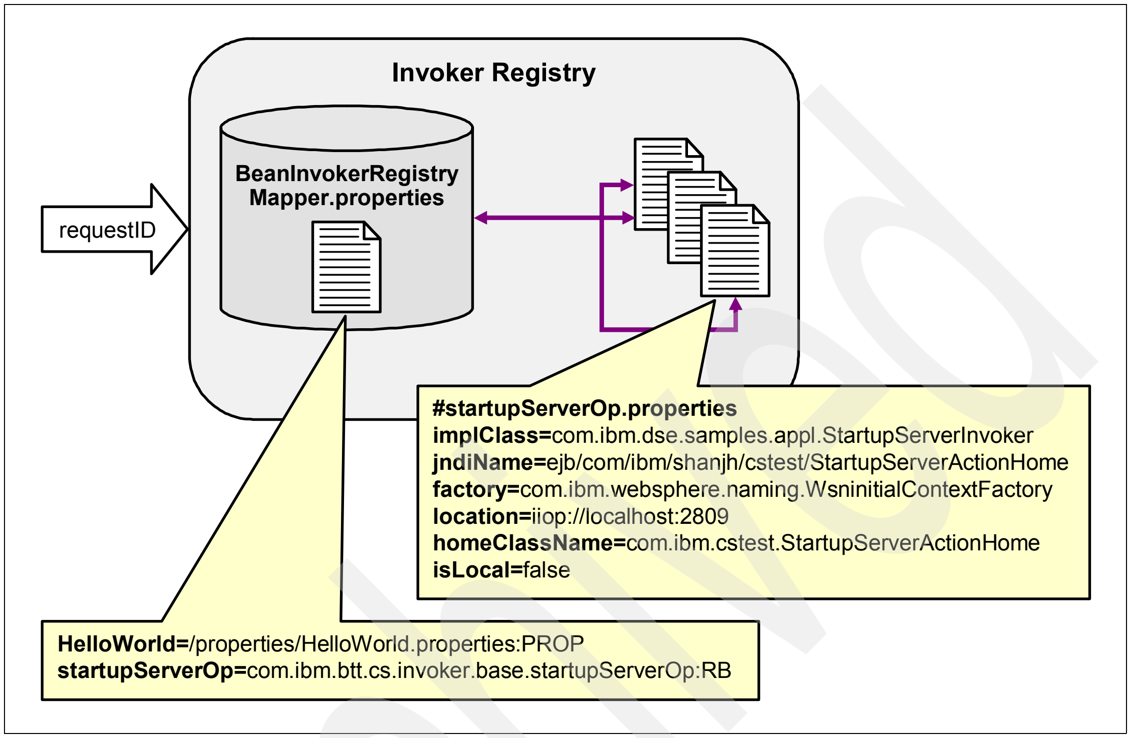


Figure 8-189 Invoker pattern architecture

- Base invoker class
  - This is the abstract class that encapsulates some basic functions about EJB accessing and cache mechanism.

  - It also provides all the format/unformat functions, for example, StringFormatter, XMLFormatter, FloatFormatter, and so on.
  - Each bean invoker should be extended from the base invoker class. You should manually modify it for each EJB method accessed.
- Interface for channels
  - Different channels should implement different invoker interfaces. The defaults are:
    - BeanInvokerForJavaRequest:`parseRequestData(String requestData)`
    - BeanInvokerForStrutsAction:`parseRequestData(Context requestData)`
- End-user extended invoker
  - For each EJB method in Single Action EJB, the end-user must extend the super invoker class. The logic for accessing the EJB method is defined in this extended class.
  - executeEJB() should be overridden by the end-user.
  - End-user extended invoker implements different interfaces according to the channels that use the Bean Invoker.
- Bean Invoker Factory
  - This is responsible for the Bean Invoker life cycle.
  - Only one instance of Bean Invoker Factory should be available in a JVM (Factory Pattern).
  - Bean Invoker Factory maintains the invoker object cache and EJB proxy object cache.
  - Exception handling.
- Bean Invoker Pool
  - It is used to store Bean Invoker Object.
  - It improve performance with caching.
- Bean Proxy Cache
  - It stores EJB Proxy Object.
  - It improves performance with EJB interaction.
  - It is session-relevant.
  - Session timeout handling
- Bean Invoker Registry (see Figure 8-190 on page 464)
  - This is a resource bundle that stores the information that is required for creating a Bean Invoker.
  - Each type of Bean Invoker should have its corresponding registry file.
  - It links a request ID to a given resource bundle name.

– There are two ways of accessing resource bundle file: ResourceBundle mode and Properties file mode.



```
#startupServerOp.properties
implClass=com.ibm.dse.samples.appl.StartupServerInvoker
jndiName=ejb/com/ibm/shanjh/cstest/StartupServerActionHome
factory=com.ibm.websphere.naming.WsninitialContextFactory
location=iiop://localhost:2809
homeClassName=com.ibm.cstest.StartupServerActionHome
isLocal=false
```

```
HelloWorld=/properties/HelloWorld.properties:PROP
startupServerOp=com.ibm.btt.cs.invoker.base.startupServerOp:RB
```

Figure 8-190 Invoker Registry

Creating the Withdrawal invoker

To create the Withdrawal invoker, perform the following tasks:

1. Import a Web project for building the Invoker.

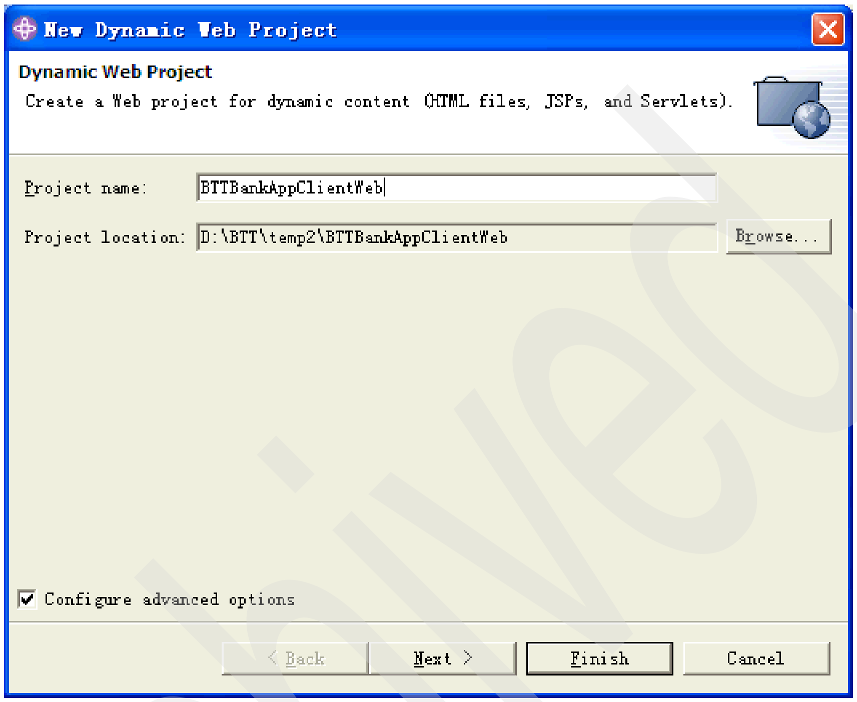
   a. From the WebSphere Studio Application Developer menu, select **File** → **Import**.

   b. In the dialog box that opens, select **WAR file** and click **Next**.

   c. From WAR file field, browse to the directory **c:\7160code\cha8\BTTBankAppClientWeb.war**.

   d. For the Project field, click **New...** to create a New Dynamic Web Project named `BTTBankAppClientWeb`, whose EAR project should be set as

`BTTBank` by selecting the **Configure advanced options** check box and clicking **Next**, as shown in Figure 8-191.



Dynamic Web Project

Create a Web project for dynamic content (HTML files, JSPs, and Servlets).

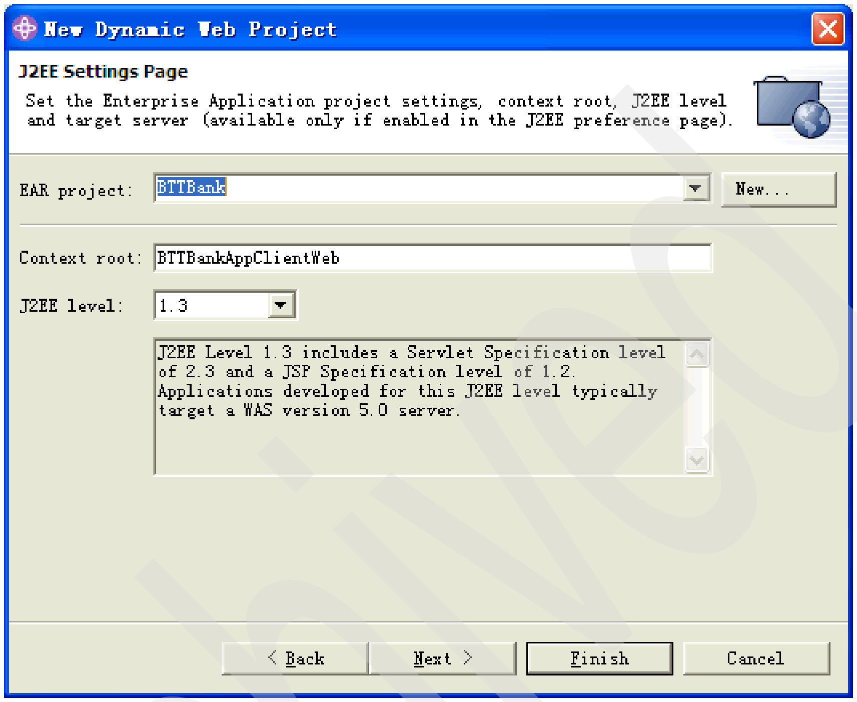
Project name: BTTBankAppClientWeb

Project location: D:\BTT\temp2\BTTBankAppClientWeb

☑ Configure advanced options

Figure 8-191 Import EAR and new BTTBankAppClientWeb

e. Set the **EAR project** as `BTTBank`, and click **Finish**. Click **OK** in the pop-up window, as shown in Figure 8-192.


New Dynamic Web Project

J2EE Settings Page

Set the Enterprise Application project settings, context root, J2EE level and target server (available only if enabled in the J2EE preference page).

EAR project: BTTBank

Context root: BTTBankAppClientWeb

J2EE level: 1.3

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

Figure 8-192 New BTTBankAppClientWeb

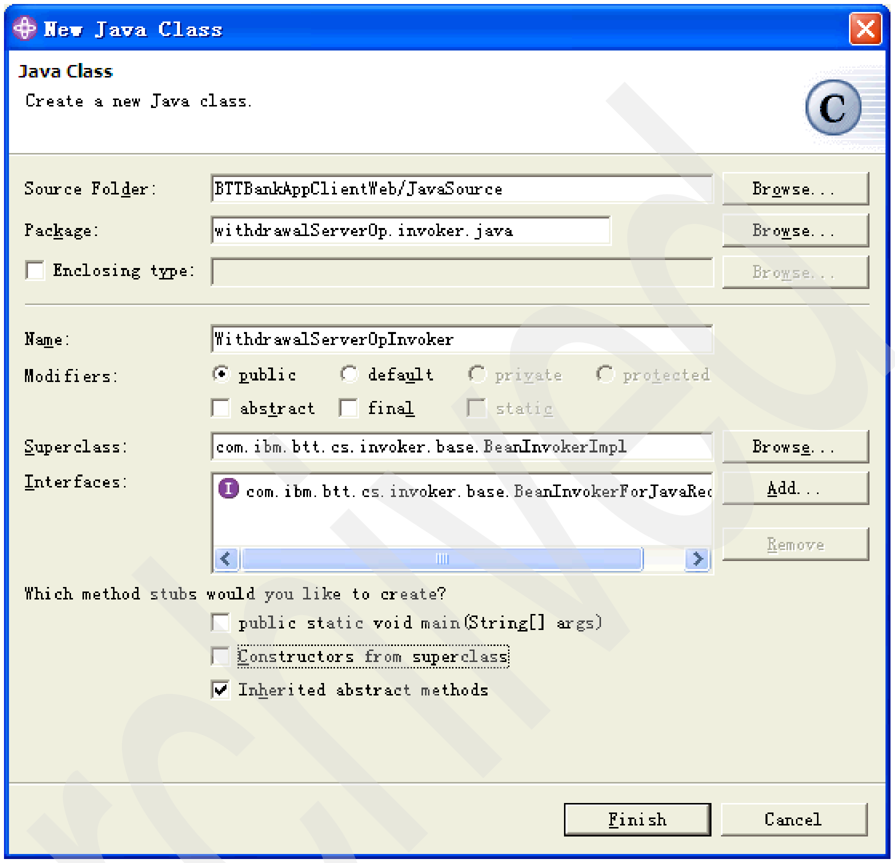
f. Click **Finish** in the Import Dialog boxto complete the Web Project Import.

g. Right-click the Web project **BTTBankAppClientWeb**, and select **Properties**. In the pop-up window, select **Java Build Path** in the left navigation panel, and the **Projects** tab in the right panel. Check **BTTBankEJB** project. Click **OK**. This step can fix the errors in the BTTBankAppClientWeb Web project.

2. In J2EE perspective, right-click **BTTBankAppClientWeb** project, and select **Java Resources**. Click **New** → **Package** to create a package named `withdrawalServerOp.invoker.java`.

3. Right-click **BTTBankAppClientWeb** project and select **Java Resources** → **withdrawalServerOp.invoker.java**. Select **New** → **Class** to create a Java class named `WithdrawalServerOpInvoker` which extends the super class com.ibm.btt.cs.invoker.base.BeanInvokerImpl and implements the interface

com.ibm.btt.cs.invoker.base.BeanInvokerForJavaRequest, as shown in Figure 8-193.



Source Folder: BTTBankAppClientWeb/JavaSource

Package: withdrawalServerOp.invoker.java

[ ] Enclosing type:

Name: WithdrawalServerOpInvoker

Modifiers: [● public] ○ default ○ private ○ protected

[ ] abstract [ ] final [ ] static

Superclass: com.ibm.btt.cs.invoker.base.BeanInvokerImpl

Interfaces: com.ibm.btt.cs.invoker.base.BeanInvokerForJavaRec

Which method stubs would you like to create?

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

[ ] Constructors from superclass

[✓] Inherited abstract methods

Figure 8-193 New class WithdrawalServerOpInvoker

4. Add code to class WithdrawalServerOpInvoker.java.

   a. Add import class as shown in Example 8-50.

Example 8-50 Import class of WithdrawalServerOpInvoker

```
import com.ibm.btt.base.Context;
import java.util.Hashtable;
import javax.ejb.EJBHome;
import com.ibm.btt.base.BTTSystemData;
import com.ibm.btt.base.Constants;
import com.ibm.btt.base.DSEInvalidArgumentException;
import com.ibm.btt.base.DSEInvalidRequestException;
import com.ibm.btt.base.DSEObjectNotFoundException;
import com.ibm.btt.cs.invoker.base.*;
import com.ibm.btt.cs.servlet.CSConstants;
import javax.naming.InitialContext;
import btt.bank.business.logic.*;
```

b. Add code to the method parseRequestData(String arg0), which uses BeanInvokerFormatter instance to unformat the request data and put the result into a hashtable, as shown in Example 8-51.

Example 8-51 WithdrawalServerOpInvoker.parseRequestData()

```
try {
   //** Prepare session data in ejb parameters
   getEjbParameters().put(Constants.SESSION_ID,
getSystemData().getSessionId());
   getEjbParameters().put(CSConstants.DATAAPPLICATIONIDKEY,
getSystemData().getSubsessionId());

   //** Get BranchId value from request data
   Tokenizer tokens = getDelimitedTokenizer(arg0); //The arg0 is request data
   BeanInvokerFormatter formatter = getFormatter();

   String s = formatter.unformatString((String)tokens.nextToken("#"),null);
   getEjbParameters().put("BranchId", s);

   //** Get AccountNumber value
   s = formatter.unformatString((String)tokens.nextToken("#"),null);
   getEjbParameters().put("AccountNumber", s);

   //** Get Date value
   java.util.Date aDate =
formatter.unformatDate((String)tokens.nextToken("#"),true,"ymd",true,"/");
   getEjbParameters().put("Date", aDate);

   //** Get Amount value
   Float amt = (Float) formatter.unformatFloat((String)tokens.nextToken("#"));
   getEjbParameters().put("Amount", amt);
} catch (Exception e) {
   throw new DSEInvalidRequestException(Constants.COMPID, "", e.getMessage());
}
```

c. Add code to the method processRespondData(Object arg0), which uses BeanInvokerFormatter instance to format the SAE execution result to a string for client withdrawal operation, as shown in Example 8-52.

Example 8-52 WithdrawalServerOpInvoker.processRespondData()

```
Hashtable haResult = (Hashtable)arg0;//The arg0 is SAE execution result.
BeanInvokerFormatter formatter = getFormatter();
```

```
//Build a string
String responseString = "";
try {
   responseString =
formatter.formatString((String)haResult.get("TrxReplyCode"), null);
   responseString = formatter.addDelimiter(responseString, "#");

   responseString +=
formatter.formatNumericString((String)haResult.get("AccountBalance"));
   responseString = formatter.addDelimiter(responseString, "#");

   responseString +=
formatter.formatString((String)haResult.get("TrxErrorMessage"), null);
   responseString = formatter.addDelimiter(responseString, "#");
} catch (Exception e) {
   throw new DSEInvalidRequestException(Constants.COMPID, "", e.getMessage());
}
return responseString;
```

d. Add code to the method createBeanInvokerProxy(), which creates an SAE instance, as shown in Example 8-53.

Example 8-53 WithdrawalServerOpInvoker.createBeanInvokerProxy()

```
WithdrawalServerOp bean = null;
WithdrawalServerOpHome home = null;
try {
   home = (WithdrawalServerOpHome)this.getHomeObject();
   bean = (WithdrawalServerOp) home.create();
} catch (Exception e) {
   throw new DSEInvalidRequestException(Constants.COMPID, "", e.getMessage());
}

return bean;
```

e. Add code to the method executeEJB(), which uses the created SAE instance to execute the method in SAE and get results, as shown in Example 8-54.

Example 8-54 WithdrawalServerOpInvoker.executeEJB()

```
WithdrawalServerOp bean = (WithdrawalServerOp) getBeanInvokerProxy();
Hashtable result = (Hashtable)bean.execute(getSystemData(),getEjbParameters());
return result;
```

f. Save and close the **Java Editor** for withdrawalServerOpInvoker.

5. Create a property file named `withdrawalServerOpOP.properties` in the package withdrawalServerOp.invoker.java.

   a. Right-click **BTTBankAppClient Web** project, and select **Java Resources** → **withdrawalServerOp.invoker.java** package. Click **New** → **Other**.

   b. In the dialog box, select **Simple** in the left navigation panel, and **File** in the right panel, as shown in Figure 8-194, and click **Next**.

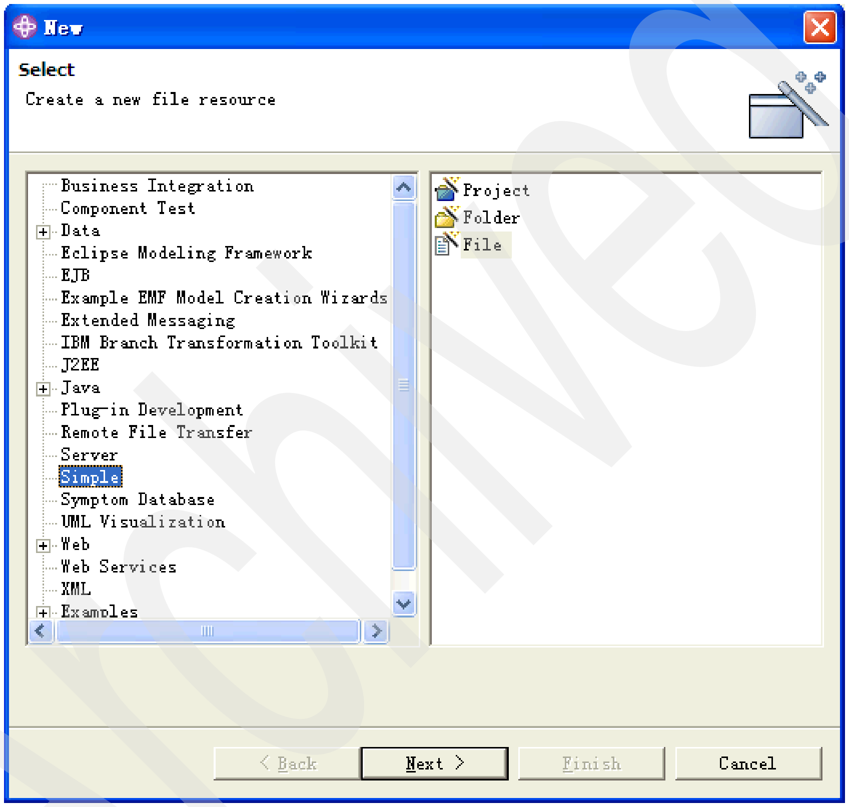
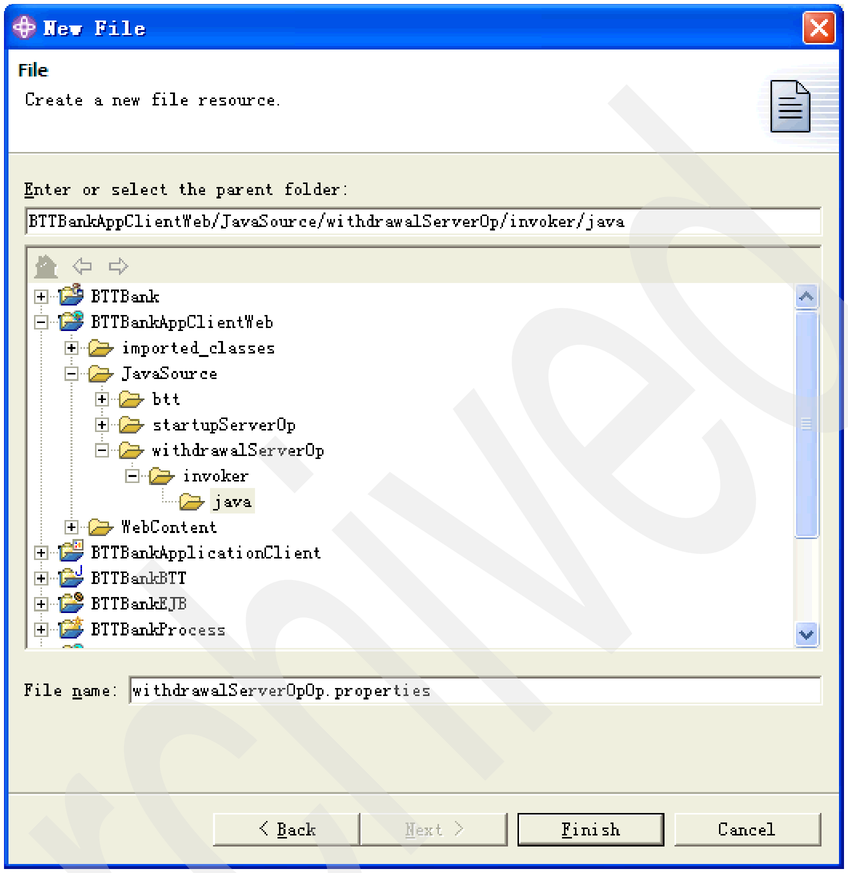


Figure 8-194 Create a simple file

   c. The value in the parent folder field should be `BTTBankAppClientWeb/JavaSource/withdrawalServerOp/invoker/java`, then type `withdrawalServerOpOP.properties` in the File name field. Click **Finish**, as shown in Figure 8-195.



Enter or select the parent folder:
BTTBankAppClientWeb/JavaSource/withdrawalServerOp/invoker/java

```
+ BTTBank
- BTTBankAppClientWeb
    + imported_classes
    - JavaSource
        + btt
        + startupServerOp
        - withdrawalServerOp
            - invoker
                java
    + WebContent
+ BTTBankApplicationClient
+ BTTBankBTT
+ BTTBankEJB
+ BTTBankProcess
```

File name: withdrawalServerOpOp.properties

Figure 8-195 New withdrawalServerOpOp.properties file

d. Add some definitions to withdrawalServerOpOP.properties as shown in Example 8-55.

Example 8-55 withdrawalServerOpOp.properties

```
implClass=withdrawalServerOp.invoker.java.WithdrawalServerOpInvoker
jndiName=ejb/btt/bank/business/logic/WithdrawalServerOpHome
factory=com.ibm.websphere.naming.WsnInitialContextFactory
location=iiop://localhost:2809
homeClassName=btt.bank.business.logic.WithdrawalServerOpHome
isLocal=false
csReplyFormat=withdrawalRepFmt
```

e. Based on the above definitions in the property file, the Invoker component can find and instance withdrawalServerOp.invoker.java.WithdrawalServerOpInvoker. Save, and close the **Java Editor**.

6. Open **BTTBankAppClientWeb** project and select **Java Resources** → **com.ibm.btt.cs.invoker.base** package → **BeanInvokerRegistryMapper.properties** with Properties File Editor. Add the code shown in Example 8-56 in a new line.

Example 8-56 BeanInvokerRegistryMapper.properties

```
withdrawalServerOp=withdrawalServerOp.invoker.java.withdrawalServerOpOp:RB
```

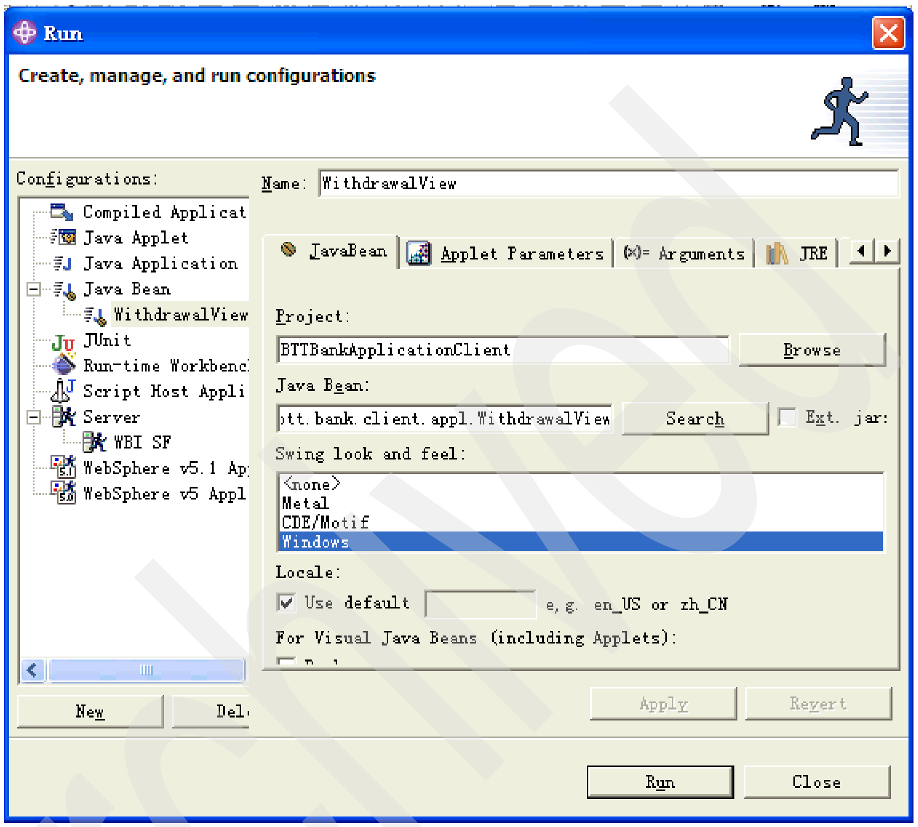
7. Save, and close the **Properties File Editor**.

8.4.7 Running this application

To run this application, perform the following tasks:

1. From WebSphere Studio Application Developer menu select **Project** → **Rebuild All**.
2. Open the **Server** perspective.
3. In the Server Configuration view, right-click **Servers** and select **WBI SF**. Select **Start**.
4. Once the WBI SF server is started, switch to the **Java** perspective and select **WithdrawalView.java** from the BTTBankApplicationClient project. From the WebSphere Studio Application Developer menu, select **Run** → **Run**.

5. In the dialog box, select **Java Bean** and click **New** in the left panel, as shown in Figure 8-196.



Name: WithdrawalView

Project: BTTBankApplicationClient

Java Bean: ›tt.bank.client.appl.WithdrawalView

Ext. jar:

Swing look and feel:

<none>
Metal
CDE/Motif
Windows

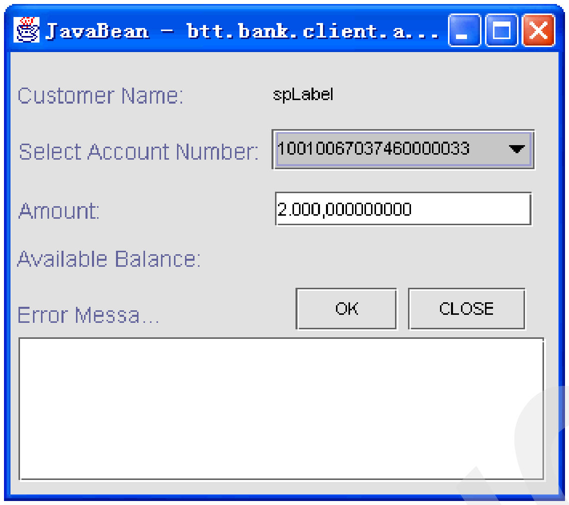
Locale:

✓ Use default e.g. en_US or zh_CN

For Visual Java Beans (including Applets):

Figure 8-196 Run template

6. Click **Run**. After a couple of moments, a Java client window is displayed as shown in Figure 8-197.


JavaBean - btt.bank.client.a...

Customer Name: spLabel

Select Account Number: 10010067037460000033

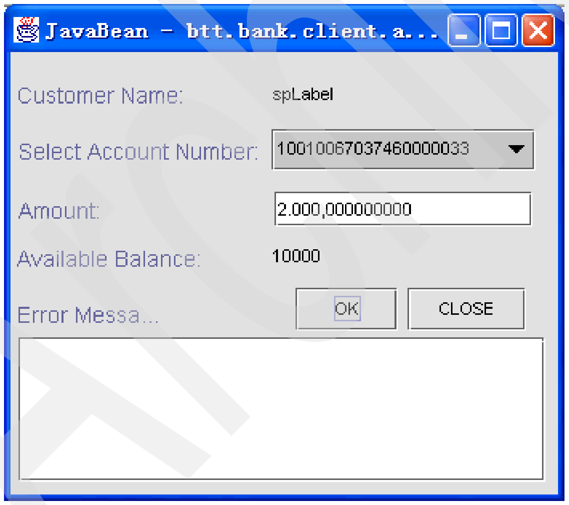
Amount: 2.000,000000000

Available Balance:

Error Messa...

Figure 8-197 The input view

7. You can modify the Account Number or Amount. If you click **OK**, a window opens, as shown in Figure 8-198.


JavaBean - btt.bank.client.a...

Customer Name: spLabel

Select Account Number: 10010067037460000033

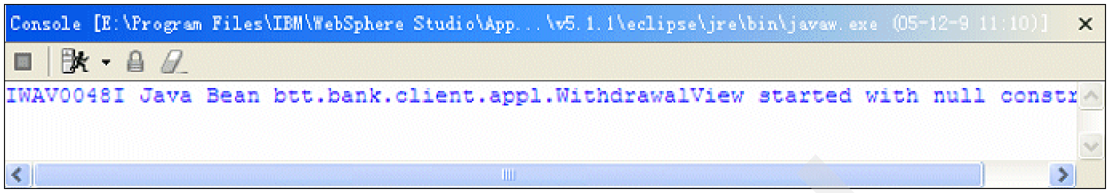
Amount: 2.000,000000000

Available Balance: 10000

Error Messa...

Figure 8-198 The result view

8. During the execution, a messageis displayed in the console shown in Figure 8-199 on page 475.


Console [E:\Program Files\IBM\WebSphere Studio\App...\v5.1.1\eclipse\jre\bin\javaw.exe (05-12-9 11:10)]

```
IWAV0048I Java Bean btt.bank.client.appl.WithdrawalView started with null constr
```

Figure 8-199 Console showing message

.

A

Branch Transformation Toolkit development and runtime requirements

When using Branch Transformation Toolkit to develop a new application or migrate an existing one, prepare the runtime and development environments to meet the requirements. This appendix lists the environment requirements for the runtime and development environments of Branch Transformation Toolkit 5.1.

Branch Transformation Toolkit 5.1 supports the following operating systems:

- Microsoft Windows 2000
  - Microsoft Windows 2000 Professional with Service Pack 4
  - Microsoft Windows 2000 Server with Service Pack 4
  - Microsoft Windows 2000 Advanced Server with Service Pack 4
- Microsoft Windows XP
  - Microsoft Windows XP Professional with Service Pack 1
- Microsoft Windows Server 2003
  - Microsoft Windows Server 2003, Standard
  - Microsoft Windows Server 2003, Enterprise

- Sun Solaris
  - Sun Solaris 8 with the Recommended Patch Cluster of November 2003
  - Sun Solaris 9 with the Recommended Patch Cluster of November 2003
- Red Hat Linux on Intel
  - RedHat Linux v7.2
  - RedHat Linux v8.0
  - RedHat Linux v9.0
  - Red Hat Enterprise Linux WS 3.0 Update 1 or 3
  - Red Hat Enterprise Linux AS 3.0 Update 1 or 3
  - Red Hat Enterprise Linux ES 3.0 Update 1 or 3
- IBM AIX
  - AIX Version 5.1 with the 5100-05
  - AIX Version 5.2 with the 5200-01 recommended maintenance package and APAR iY44183, and PTF U484272
  - AIX Version 5.2 with 5200-03 recommended maintenance package
  - AIX 5L^{TM} version 5.3 with WebSphere Application Server APAR PK01428
- IBM z/OS
  - IBM z/OS 1.2 or later

Some differences exist between development environments and runtime environments.

A.1 Microsoft Windows 2000

For Windows 2000 client side, the minimum requirements of runtime environment are listed in Table A-1.

Table A-1 Windows 2000 client requirements

Windows 2000 client side	**Requirement description**
Processor	PII 266 MHz or higher
Memory	48 MB minimum /64 MB recommended
Hard drive	50 M
Display	800 x 600 minimum / 1024 x 768 recommended
Intranet test LAN base	16 MB token ring / 100 MB BaseTx Ethernet recommended
Operating system	Microsoft Windows 2000 Professional with Service Pack 4
Browser	One of the following is required: ► Netscape Communicator 7.02 or later with Java^{TM} Plug-in 1.3.1 or later ► Internet Explorer 6.0 SP1
Communication protocol	TCP/IP
JDK	Java^{TM} 2 SDK supplied by WebSphere Application Server

The minimum requirements of server side are listed in Table A-2.

Table A-2 Windows 2000 server requirements

Windows 2000 server side	**Requirement description**
Processor	PIII 500 MHz or higher
Memory	512 MB minimum / 768 MB recommended
Hard drive	100 MB minimum
Intranet test LAN base	16 MB token ring / 100 MB BaseTx Ethernet recommended

Windows 2000 server side	**Requirement description**
Operating System	One of the following is required: ► Microsoft Windows 2000 Advanced Server with Service Pack 4 ► Microsoft Windows 2000 Server with Service Pack 4
Application server	One of the following is required: ► IBM(R) WebSphere Application Server - Express V5.1.1 ► IBM WebSphere Application Server V5.1.1 ► IBM WebSphere Application Server Network Deployment V5.1.1 ► IBM WebSphere Application Server For Developers V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1 for Multiplatforms
Communication protocol	TCP/IP
Communication services (LU0, LU6.2)	IBM Communications Server V6.1.2
Database manager	One of the following is required: ► DB2 UDB Enterprise Server Edition v8.1 FP5 or FP7 ► DB2 UDB Enterprise Server Edition v8.2 ► Oracle 8i Standard/Enterprise Release 3 V8.1.7.4 ► Oracle 9i Standard/Enterprise Release 2 V9.2.0.5 ► Microsoft SQL Server 2000 Enterprise SP3 ► Microsoft SQL Server 2000 Standard Edition
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

For the Windows 2000 development environments, the minimum requirements are listed in Table A-3.

Table A-3 Windows 2000 development requirements

Windows 2000 development	Requirement description
Processor	PIII 500 MHz or higher recommended
Memory	1024 MB recommended
Hard drive	2 GB minimum for WebSphere Studio Application Developer; 4.5 GB minimum for WebSphere Studio Application Developer Integration Edition
Display	TCP/IP
Operating system	100 MB Ethernet recommended
Integrated development environment	One of the following is required: ► IBM WebSphere Studio Application Developer V5.1.1 ► IBM WebSphere Studio Application Developer Integration Edition V5.1.1
Browser	► Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later ► Internet Explorer 6.0 SP1

A.2 Microsoft Windows XP requirements

For Windows XP, Table A-4 lists the requirements of client side of runtime environment.

Table A-4 Windows XP client requirements

Windows XP client side	Requirement description
Processor	PII 266 MHz or higher
Memory	48 MB minimum /64 MB recommended
Hard drive	50 MB
Display	800 x 600 minimum / 1024 x 768 recommended
Intranet test LAN base	16 MB token ring / 100 MB BaseTx Ethernet recommended

Windows XP client side	Requirement description
Operating system	Windows XP Professional with SP1
Browser	Any of the following: ► Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later ► Internet Explorer 6.0 SP1
Communication protocol	TCP/IP
JDK	Java 2 SDK supplied by WebSphere Application Server

Table A-5 lists the requirements for development environment.

Table A-5 Windows XP development requirements

Windows XP development	Requirement description
Processor	PII 500 MHz processor. PIII 500 MHz or higher recommended
Memory	1024 MB recommended
Hard drive	2 GB minimum for WebSphere Studio Application Developer; 4.5 GB minimum for WebSphere Studio Application Developer Integration Edition
Display	800 x 600 minimum / 1024 x 768 recommended at a color setting of High Color (16 bit)
Operating system	Windows XP Professional with SP1
Integrated development environment	One of the following is required: ► IBM WebSphere Studio Application Developer V5.1.1 ► IBM WebSphere Studio Application Developer Integration Edition V5.1.1
Browser	► Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later ► Internet Explorer 6.0 SP1

A.3 Microsoft Windows Server 2003 requirements

Table A-6 lists the requirements of server side of runtime environment for Windows Server 2003.

Table A-6 Windows 2003 server runtime requirements

Windows 2003 server side	**Requirement description**
Processor	PII 500 MHz or higher
Hard drive	100 MB minimum
Memory	512 MB minimum / 768 MB recommended
Operating system	One of the following is required: ► Windows Server 2003 Standard ► Windows Server 2003 Enterprise
Application server	One of the following is required: ► IBM WebSphere Application Server - Express V5.1.1 ► IBM WebSphere Application Server V5.1.1 ► IBM WebSphere Application Server Network Deployment V5.1.1 ► IBM WebSphere Application Server For Developers V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1 for Multiplatforms
Communication protocol	TCP/IP
Communication services (LU0, LU6.2)	IBM Communications Server V6.1.2

Windows 2003 server side	Requirement description
Database manager	One of the following is required: ► DB2 UDB Enterprise Server Edition v8.1 FP5 or FP7 ► DB2 UDB Enterprise Server Edition v8.2 v Oracle 8i Standard/Enterprise Release 3 V8.1.7.4 ► Oracle 9i Standard/Enterprise Release 2 V9.2.0.5 ► Microsoft SQL Server 2000 Enterprise SP3 ► Microsoft SQL Server 2000 Standard Edition
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

A.4 Linux on Intel requirements

Table A-7 lists the requirements of client side for Linux on Intel platforms.

Table A-7 Linux on Intel client requirements

Linux client side	Requirement description
Processor	PII 233 MHz or higher
Memory	48 MB minimum / 64 MB recommended
Hard drive	50 MB
Display	800 x 600 minimum / 1024 x 768 recommended
Intranet test LAN base	16 MB token ring / 100 MB BaseTx Ethernet recommended
Operating system	One of the following is required: ► Red Hat 8.0 ► Red Hat 9.0 ► Red Hat Enterprise Linux WS 3.0 Update 1 or 3
Browser	Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later

Linux client side	Requirement description
Communication protocol	TCP/IP
JDK	Java 2 SDK supplied by WebSphere Application Server

Table A-8 lists the requirements of server side for Linux on Intel platforms.

Table A-8 inux on Intel server requirements

Linux server side	Requirement description
Processor	PII 500 MHz or higher
Hard drive	100 MB minimum
Memory	512 MB minimum / 768 MB recommended
Operating system	One of the following is required ► Red Hat Enterprise Linux AS 3.0 Update 1 or 3 ► Red Hat Enterprise Linux ES 3.0 Update 1 or 3
Application server	One of the following is required: ► IBM WebSphere Application Server - Express V5.1.1 v ► IBM WebSphere Application Server V5.1.1 ► IBM WebSphere Application Server Network Deployment V5.1.1 ► IBM WebSphere Application Server For Developers V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1 for Multiplatforms
Communication protocol	TCP/IP
Communication services (LU0, LU6.2)	Communications Server for Linux V6.1.2

Linux server side	Requirement description
Database manager	One of the following is required: ▶ DB2 UDB Enterprise Server Edition v8.1 FP5 or FP7 ▶ DB2 UDB Enterprise Server Edition v8.2 ▶ Oracle 8i Standard/Enterprise Release 3 V8.1.7.4 ▶ Oracle 9i Standard/Enterprise Release 2 V9.2.0.5
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

Table A-9 lists the requirements of development environment for Linux on Intel platform.

Table A-9 inux on Intel development requirements

Linux development	Requirement description
Processor	PII processor 500 MHz. PIII 500 MHz or higher recommended
Memory	1024 MB
Hard drive	2 GB minimum for WebSphere Studio Application Developer; 4.5 GB minimum for WebSphere Studio Application Developer Integration Edition
Display	1024 x 768 minimum at a color setting of 16 bit for workstation
Operating system	Red Hat Linux v7.2 or later
Integrated development environment	One of the following is required for development of workstations: ▶ IBM WebSphere Studio Application Developer V5.1.1 ▶ IBM WebSphere Studio Application Developer Integration Edition V5.1.1
Browser	Netscape Communicator 7.02 or later with Java Plug-in 1.3.1 or later

A.5 Sun Solaris requirements

Table A-10 lists the requirements for Solaris on server side.

Table A-10 Sun Solaris server runtime requirements

Solaris server side	Requirement description
Processor	Sparc workstation at 440 MHz, or faster
Hard drive	100 MB minimum
Memory	768 MB minimum / 1024 MB recommended
Operating system	One of the following is required: ► Solaris 8 with the Recommended Patch Cluster of November 2003 ► Solaris 9 with the Recommended Patch Cluster of November 2003
Application server	One of the following is required: ► IBM WebSphere Application Server - Express V5.1.1 ► IBM WebSphere Application Server V5.1.1 ► IBM WebSphere Application Server Network Deployment V5.1.1 ► IBM WebSphere Application Server For Developers V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1 for Multiplatforms
Communication protocol	TCP/IP

Solaris server side	Requirement description
Database manager	One of the following is required: ► DB2 UDB Enterprise Server Edition v8.1 FP5 or FP7 ► DB2 UDB Enterprise Server Edition v8.2 ► Oracle 8i Standard/Enterprise Release 3 V8.1.7.4 ► Oracle 9i Standard/Enterprise Release 2 V9.2.0.5
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

A.6 IBM AIX requirements

Table A-11 lists the requirements of server side environment for AIX.

Table A-11 AIX server runtime requirements

AIX server side	Requirement description
Processor	604e RS/6000Æ workstation at 375 MHz or higher frequency
Hard drive	100 MB minimum
Memory	512 MB minimum / 768 MB recommended
Operating system	One of the following is required: ► AIX Version 5.1 with the 5100-05 l ► AIX Version 5.2 with the 5200-01 recommended maintenance package and APAR iY44183, and PTF U484272 l ► AIX Version 5.2 with 5200-03 recommended maintenance package l ► AIX 5L™ version 5.3 with WebSphere Application Server APAR PK01428

AIX server side	Requirement description
Application server	One of the following is required: ► IBM WebSphere Application Server - Express V5.1.1 ► IBM WebSphere Application Server V5.1.1 ► IBM WebSphere Application Server Network Deployment V5.1.1 ► IBM WebSphere Application Server For Developers V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1.1 ► IBM WebSphere Business Integration Server Foundation V5.1 for Multiplatforms
Communication protocol	TCP/IP
Communication services (LU0, LU6.2)	IBM Communications Server V6.1.2
Database manager	One of the following is required: ► DB2 UDB Enterprise Server Edition v8.1 FP5 or FP7 ► DB2 UDB Enterprise Server Edition v8.2 ► Oracle 8i Standard/Enterprise Release 3 V8.1.7.4 ► Oracle 9i Standard/Enterprise Release 2 V9.2.0.5
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

A.7 IBM z/OS requirements

Table A-12 lists the requirements for z/OS.

Table A-12 z/OS runtime requirments

zSeries® eServer™	Requirement description
Processor	Any hardware that supports z/OS 1.2 or z/OSe 1.3, or even later versions. S/390 Parallel Enterprise Server. Generation 5 or 6, or ever later systems is recommended
Hard drive	100 MB minimum
Memory	512 MB minimum /768 MB recommended
Operating system	z/OS (5694-A01) 1.2 or later
Application server	One of the following is required: ▶ IBM WebSphere Application Server for z/OS V5.1 ▶ IBM WebSphere Business Integration Server Foundation for z/OS V5.1
Communication protocol	TCP/IP
Communication services (LU0, LU6.2)	IBM Communications Server V6.1.2
Database manager	One of the following is required: ▶ IBM DB2 Universal Database. V7.0 for z/OS ▶ IBM DB2 Universal Database V8.0 for z/OS
Portal Connector	IBM WebSphere Portal for Multiplatforms V5.1

A.8 Additional requirements

Depending on the framework services you use, you may require other hardware and software to support financial devices. The following additional requirements apply to the type of workstation, that is, client, server, or development, that accesses the financial device as shown in Table A-13.

Table A-13 Additional requirements

Framework component	**Additional requirements**
J/eXtensions for Financial Services	Any financial printer, magnetic stripe reader/encoder, or check reader with a device service that is compliant with the J/XFS specification.
eXtensions for Financial Services	Any financial printer, magnetic stripe reader/encoder, or check reader with a device service that is compliant with the J/XFS specification.
LANDP MSR/E Device Service	Any magnetic stripe reader/encoder supported by the LANDP MSRE47## server.

B

Setting up a Branch Transformation Toolkit sample application

The toolkit application can be set up to run in the WebSphere test environment within WebSphere Studio Application Developer Integration Edition 5.1.1 or on WebSphere Studio Application Developer V5.1.1.

The Java client sample application includes a J2EE enterprise application (EAR) file containing everything you need to run this sample. The procedures contained in the first section describe how to set up or configure the EAR file and deploy it.

The HTML sample application can be set up to run in the WebSphere test environment within WebSphere Studio Application Developer Integration Edition 5.1.1 . The HTML sample application includes a EAR file containing everything you need to run this sample. The procedures contained in the second section describes how to set up or configure the EAR file and deploy it.

All the resources needed by sample applications are provided by the Branch Transformation Toolkit after it is installed successfully.

B.1 Setting up the Java sample application

The following procedure describes how to install the Java sample application in WebSphere Studio Application Developer 5.1.1 and run the sample in one of the test environment configurations. The procedure described here applies if you are running WebSphere Studio Application Developer 5.1.1 on Microsoft Windows.

To set up the application in WebSphere Studio Application Developer 5.1.1, perform the following steps:

1. Copy external files.
   a. Locate the EAR file BTTJavaSample.ear from <toolkit_root>\samples\JavaSampleApplication\StandAlone\BTTJavaSample.ear
   b. Extract the **BTTJavaSampleWeb.war** file from BTTJavaSample.ear, then extract the **\dse\server** directory from the BTTJavaSampleWeb.war.
   c. On your OS system, create a directory, `c:\dse`.
   d. Copy all the files in the \dse\server directory you extracted from BTTJavaSampleWeb.war to the c:\dse directory you created.
2. Create database and tables:
   a. Run the following command in the DB2 command window to create a database named `sample`:

      ```
      DB2 CREATE DATABASE SAMPLE
      ```

      Set a user and password for access to the database. For example, set both user and password as `db2admin`.
   b. Create three tables:
      i. Create a directory called `c:\temp`.
      ii. Copy ***<toolkit_root>*\dbtools\Windows\DB2\tableDefinition\cha\createCHATables.ddl** to the c:\temp directory.

         Note that *<toolkit_root>* refers to the root directory where you have the toolkit installed.
      iii. Change your current directory to `c:\temp`, and open a DB2 command window.
      iv. In the DB2 command window, run:

         ```
         DB2 CONNECT TO SAMPLE USER db2admin USING db2admin.
         ```
      v. In the DB2 Command Window, run:

         ```
         db2 -tvf createCHATables.ddl
         ```

You will see messages indicating that CHAChildren, CHAInstance, and CHAControl tables have been created successfully.

3. Import BTTJavaSample.ear.

   a. Start WebSphere Studio Application Developer **5.1.1**. From the menu bar, open **J2EE** perspective. Select **File** → **Import** → **EAR file**. Click **Next**.

   b. In the Import wizard, set the following parameters, and click **Finish**:

      - EAR File: `<toolkit_root>\samples\JavaSampleApplication\StandAlone \BTTJavaSample.ear`
      - Enterprise Application project name: `BTTJavaSample`

4. Import dummysnalu0.rar.

   a. Open **J2EE** perspective and from the menu bar, select **File** → **Import** → **RAR file**. Click **Next**.

   b. In the Import wizard, set the following values, and click **Finish**:

      - Connector File: `<toolkit_root>\jars\ dummysnalu0.rar`
      - Connector Project: `dummysnalu0Connector2`

5. Set up server in WebSphere Studio Application Developer 5.1 test environment.

   a. Open **Server** perspective.

   b. Select **New** → **Server and Server Configuration**, and set the following parameters, and click **Finish**:

      - Server name: `JavaSampleServer`
      - Folder: `Servers`
      - Server Type: `Integration Test Environment`

   c. Define JAAS Authentication entries in server configuration.

      i. In the Server Configuration panel, double-click the server instance **JavaSampleServer**.

      ii. Select the **Security** tab. In the Cell Settings section, click **Add** to add a JAAS Authentication Entry and set the following parameters:

         - Alias: `CHA`
         - User ID: `db2admin`
         - Password: `db2admin`
         - Description: `JavaSample`

      iii. Click **Add** to add a JAAS Authentication Entry and set the following parameters, and click **OK**:

         - Alias: `sna`
         - User ID: `sna`

  - Password: `sna`
  - Description: `JavaSample`

d. Select the **Source** tab. In the Server Settings section, click **Add** to add IBM DB2 JDBC Provider (XA) to the JDBC Provider list.

e. In the window that pops up, select **IBM DB2** in the Database type section and **DB2 JDBC Provider (XA)** in the JDBC Provider type section. Click **Next**.

f. In the window that pops up, input name `DB2 JDBC Provider (XA)`. Click **Finish**.

g. In the Data Source tab and the Server Settings section, select the **DB2 JDBC Provider (XA)** in the JDBC Provider list and click **Add** to create a data source. In the window that pops up, select **DB2 JDBC Provider (XA)** in the type of JDBC Provider section and select **Version 5.0** data source in the Data Source type section. Click **Next**.

h. In the pop-up Modify Data Source window, set the following parameters:

  - Name: `CHADataSource`
  - JNDI name: `jdbc/CHADataSource`
  - Description: `CHA Exercise`
  - Statement cache size: `10`
  - Data source helper class name: `com.ibm.websphere.rsadapter.DB2DataStoreHelper`
  - Connection timeout: `1800`
  - Maximum connections: `10`
  - Minimum connections: `1`
  - Reap time: `180`
  - Unused timeout: `1800`
  - Aged timeout: `0`
  - Purge policy: `EntirePool`
  - Component-managed authentication alias: `CHA`
  - Container-managed authentication alias: `CHA`
  - Select the check box against **Use this data source in container managed persistence (CMP)**.

i. Select the **J2C** tab. In the J2C Resource Adapters section, click **Add** to add a Resource Adapter to the Resource Adapter list.

j. In the Create Resource Adapter window that pops up, click **OK**.

k. In the Resource Adapter list, select **dummysnalu0Connector**. In the J2C Connection Factories section, click **Add**.

l. In the Create Connection Factory window that pops up, set the following parameters, and click **OK**:

  - Name: `snalu0`

- JNDI name: `snalu0`
- Description: `snaDataSource`
- Component-managed authentication alias: `sna`
- Container-managed authentication alias: `sna`

All other fields can be left with their default values.

m. In the Resource Properties section, set the following parameters:

- TestFile: `http://127.0.0.1:9080/BTTJavaSampleWeb/response.res`
- userName: `sna`
- userPassword: `sna`

n. Save the modification to server configuration and close the **Server Configuration Content Editor**.

6. Do the EJB to RDB mapping.

a. From the J2EE perspective, right-click **BTTCHAEJB** and select **Generate** → **EJB to RDB Mapping**.

b. In the EJB to RDB Mapping window, select the **Create a new backend folder** option, and click **Next**.

c. Select the **Top Down** option and click **Next**.

d. Input the following settings, and click **Finish**:

- In Target Database section, select **DB2 Universal Database V8.1**.
- In Database name section, type `SAMPLE`.
- In Schema name section, type `db2admin`.
- Deselect **Generate DDL** and **WebSphere 3.x Compatible**.

7. Deploy BTTCHAEJB, bttsvcinfra, BTTFormatterEJB and BTTJavaSampleEJB.

a. From J2EE perspective, right-click **BTTCHAEJB**. Select **Generate** → **Deploy and RMIC Code**.

b. In the window that pops up, select all the check boxes and click **Finish**.

c. From the J2EE perspective, right-click **bttsvcinfra**. Select **Generate** → **Deploy and RMIC Code**.

d. From the J2EE perspective, right-click **BTTFormatterEJB**. Select **Generate** → **Deploy and RMIC Code**.

e. In the window that pops up, select all the check boxes and click **Finish**.

f. From J2EE perspective, right-click **BTTJavaSampleEJB**, and select **Generate** → **Deploy and RMIC Code**.

g. In the window that pops up, select all the check boxes and click **Finish**.

8. Associate the BTTJavaSample project with the server configuration.

   In the Servers view, right-click and select **Add and Remove Projects**. Add **BTTJavaSample**, and click **Finish**.

9. Start the JavaSampleServer.

   In the JavaSampleServer view, right-click and select **Start**.

10. Configure the BTTJavaSampleClient project classpath

    Add the ***XERCESJAR*** variable.

11. Run the JavaClient.

    a. From the WebSphere Studio Application Developer menu, select **Run** → **Run**.

    b. Double-click **Java application** in the new window, and select **Browse** in the right panel.

    c. Select **BTTJavaSampleClient** → **Search under the main-class** → **OpenDesktop**, and click **Run**.

B.2 Setting up the HTML sample application

The following procedure describes how to install HTML sample application in WebSphere Studio Application Developer Integration Edition 5.1.1 and run the sample in one of the test environment configurations. Follow this procedure if you are running WebSphere Studio Application Developer Integration Edition 5.1.1 on Windows.

To set up the application in WebSphere Studio Application Developer Integration Edition 5.1.1, perform the following tasks:

1. Copy external files.

   a. Locate the EAR file <toolkit_root>\samples\HtmlSampleBPApplication\BTTHTMLSampleBPEAR.ear

   b. Extract the **BTTHTMLSampleWeb.war** file from BTTHTMLSampleBPEAR.ear, then extract the **\dse** directory from the BTTHTMLSampleWeb.war.

   c. In your OS, create a directory, `c:\dse`.

   d. Copy all the files in the \dse directory you extracted from BTTHTMLSampleWeb.war to the c:\dse directory you created.

2. Create database and tables.

   a. Run the following in the DB2 command window to create a database named `sample`:

      ```
      DB2 CREATE DATABASE SAMPLE
      ```

      You can set user and password for database sample, for example, set both user and password as `db2inst1`.

   b. Create three tables.

      i. Create a directory called `c:\temp`.

      ii. Copy **<toolkit_root>\dbtools\Windows\DB2\tableDefinition\cha\createCHATables.ddl** to the c:\temps directory.

      iii. Shift to c:\temp directory and open the DB2 command window.

      iv. In DB2 command window, run:

      ```
      DB2 CONNECT TO SAMPLE USER db2admin USING db2admin
      ```

      v. In DB2 command window, run:

      ```
      db2 -tvf createCHATables.ddl
      ```

      You will see messages indicating that CHAChildren, CHAInstance, and CHAControl tables have been created successfully.

3. Import BTTHTMLSampleBPEAR.ear.

   a. Start **WebSphere Studio Application Developer Integration Edition 5.1.1** . From the menu bar, open **J2EE** perspective. Select **File** → **Import** → **EAR file**, and click **Next**.

   b. In the Import wizard, set the following parameters:

      - EAR File: `<toolkit_root>\samples\HtmlSampleBPApplication\BTTHTMLSampleBPEAR.ear`
      - Enterprise Application project name: `BTTHTMLSampleBPEAR`

   c. Click **Next**.

   d. Select **BTTHTMLSampeBP.jar**, and click **Finish.**

4. Import dummysnalu0.rar.

   a. Open **J2EE** perspective and from the menu bar, select **File** → **Import** → **RAR file**, and click **Next**.

   b. In the Import wizard, set the following values, and click **Finish**:

      - Connector File: `<toolkit_root>\jars\ dummysnalu0.rar`
      - Connector Project: `dummysnalu0Connector`

5. Open **J2EE** perspective. In the Project Navigator Panel, select **BTTHTMLSampleBP** and right-click **Source**, and then **Delete** .

6. Import BTTHTMLSampleBP.jar to BTTHTMLSampleBP project.

   a. Extract the **BTTHTMLSampleBP.jar** file from <toolkit_root>\samples\HtmlSampleBPApplication\BTTHTMLSampleBPEAR.ear

   b. Open **J2EE** perspective. In the Project Navigator Panel, right-click **BTTHTMLSampleBP** and select **Import** → **Zip file**. Set the following parameters, and click **Finish**:

      - From Zip file: `<toolkit_root>/ear/BTTHTMLSampleBP.jar`
      - Into folder: `BTTHTMLSampleBP`

7. Edit the source folder.

   a. Open the **J2EE** perspective. In the Project Navigator Panel, right-click **BTTHTMLSampleBP** and select **Properties** → **Java Build Path**.

   b. In the Source panel, select **BTTHTMLSampleBP/source** and click **Edit**.

   c. In the window that pops up, select **Project as source folder option**. Click **OK**.

   d. Select the check box against **Allow output folders for source folders**, and click **OK**.

8. Correct the build path error.

   a. Open **J2EE** perspective. In the Project Navigator Panel, right-click **BTTHTMLSampleBP** and select **PropertiesJava Build Path**.

   b. In the Libraries Panel, select **WebSphere V5.1 EE JRE** and click **Remove**.

   c. Click **Add Library**.

   d. In the window that pops up, select **JRE System Library**. Click **Next**.

   e. Select **WebSphere v5.1 EE JRE** and click **Finish**.

   f. Click **OK**.

9. Use BPEL file to generate the deploy code.

   a. Open **J2EE** perspective. In the Project Navigator Panel, select **BTTHTMLSampleBP**. Right-click **AccountTransfer.bpel** and select **EnterPrise Services** → **Generate Deploy Code**.

   b. In the window that pops up, click **OK**.

10. From the menu bar, select **Project** → **Rebuild All**.

11. Modify the WSDL file and rebuild the project BTTHTMLSampleWeb.

   a. Open **J2EE** perspective. In the Project Navigator Panel, select **BTTHTMLSampleWeb** and open **AccountTransferInterface.wsdl**.

b. Modify the following part:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:import namespace=http://btt.ibm.com/base
schemaLocation="BTTSystemData.xsd">

 </xsd:import>

</xsd:schema>
```

c. Save the file. Right-click **BTTHTMLSampleWeb** and select **Rebuild Project**.

12. Set up server in WebSphere Studio Application Developer 5.1 test environment.

a. Open the **Server** perspective.

b. Click **New** → **Server and Server Configuration** and set the following parameters, and click **Finish**:

- Server name: `HTMLSampleServer`
- Folder: `Servers`
- Server Type: `Integration Test Environment`

c. Define JAAS Authentication entries in server configuration.

i. In the Server Configuration panel, double-click the server instance **HTMLSampleServer**.

ii. Select the **Security** tab. In the Cell Settings section, click **Add** to add a JAAS Authentication Entry and set the following parameters:

- Alias: `CHA`
- User ID: `db2admin`
- Password: `db2admin`
- Description: `HTMLSample`

iii. Click **Add** to add a JAAS Authentication Entry, set the following parameters, and click **OK**:

- Alias: `sna`
- User ID: `sna`
- Password: `sna`
- Description: `HTMLSample`

d. Select the **Data source** tab. In the Server Settings section, Click **Add** to add an IBM(R) DB2 JDBC Provider (XA) to the JDBC Provider list.

e. In the window that pops up, select **IBM DB2** in the DataBase type section and **DB2 JDBC Provider (XA)** in the JDBC Provider type section. Click **Next**.

f. In the window that pops up, input the name `DB2 JDBC Provider (XA)` and click **Finish**.

g. In the Data Source tab and Server Settings section, select **DB2 JDBC Provider (XA)** in the JDBC Provider list and click **Add** to create a Data Source. In the window that pops up, select **DB2 JDBC Provider (XA)** in the type of JDBC Provider section and select **Version 5.0** data source in the data source type section. Click **Next**.

h. In the pop-up Modify Data Source window, set the following parameters:

- Name: `CHADataSource`
- JNDI name: `jdbc/CHADataSource`
- Description: `CHA Exercise`
- Statement cache size: `10`
- Data source helper class name: `com.ibm.websphere.rsadapter.DB2DataStoreHelper`
- Connection timeout: `1800`
- Maximum connections: `10`
- Minimum connections: `1`
- Reap time: `180`
- Unused timeout: `1800`
- Aged timeout: `0`
- Purge policy: `EntirePool`
- Component-managed authentication alias: `CHA`
- Container-managed authentication alias: `CHA`
- Select the check box for the field **Use this data source in container managed persistence (CMP)**

i. Select the **J2C** tab. In the J2C Resource Adapters section, click **Add** to add an Resource Adapter to the Resource Adapter list.

j. In the Create Resource Adapter window that pops up, click **OK**.

k. In the Resource Adapter list, select **dummysnalu0Connector**. In the J2C Connection Factories section, click **Add**.

l. In the Create Connection Factory window that pops up, set the following parameters, and click **OK**:

- Name: `snalu0`
- JNDI name: `snalu0`
- Description: `snaDataSource`
- Component-managed authentication alias: `sna`
- Container-managed authentication alias: `sna`

All other fields can be left with their default values.

m. In the Resource Properties section, set the following parameters:

- TestFile: `http://127.0.0.1:9080/BTTHTMLSampleBPWeb/response.res`

- userName: `sna`
- userPassword: `sna`

n. Save the modification to server configuration and close the server configuration content editor.

13. Do EJB to RDB mapping.

   a. From the J2EE perspective, right-click **BTTCHAEJB** and select **Generate** → **EJB to RDB Mapping**.

   b. In the EJB to RDB Mapping window, select **Create a new backend folder** option, click **Next**.

   c. Select **Top Down** option and click **Next**.

   d. Carry out the following settings, and click **Finish**:

   - In Target Database section, select **DB2 Universal Database(TM) V8.1**
   - In Database name section, type `SAMPLE`.
   - In Schema name section, type `db2admin`.
   - Uncheck the check boxes against **Generate DDL** and **WebSphere 3.x Compatible**.

14. Deploy BTTCHAEJB, BTTFormaterEJB, BTTHTMLSampleBPEJB, bttsvcinfra, BTTHTMLSampleEJB.

   a. From the J2EE perspective, right-click **BTTCHAEJB**. Select **Generate** → **Deploy and RMIC Code**.

   b. In the window that pops up, select all the check boxes and click **Finish**.

   c. From the J2EE perspective, right-click **BTTFormaterEJB**, select **Generate** → **Deploy and RMIC Code**.

   d. In the window that pops up, select all the check boxes and click **Finish**.

   e. From the J2EE perspective, right-click **BTTHTMLSampleBPEJB**, select **Generate** → **Deploy and RMIC Code**.

   f. In the window that pops up, select all check boxes, and click **Finish**.

   g. From the J2EE perspective, right-click **bttsvcinfra**, select **Generate** → **Deploy and RMIC Code**.

   h. In the window that pops up, check all the check boxes and click **Finish**.

   i. From the J2EE perspective, right-click **BTTHTMLSampleEJB**, select **Generate** → **Deploy and RMIC Code**.

   j. In the window that pops up, select all the check boxes and click **Finish**.

15. Associate the BTTHTMLSample project with the server configuration.

   In the Servers view, right-click and select **Add and Remove Projects**. Add **BTTHTMLSampleBPEAR**. Click **Finish**.

16. Start HTMLSampleServer.

    In the HTMLSampleServer view, right-click and select **Start**.

17. Run HTMLSample.

    Open the Web browser and input the following URL:

    ```
    http://serverName:9080/BTTHTMLSampleWeb/btt/html/sign/prepareSignIn.do
    ```

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG247160`

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, SG247160.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
SG247160.zip	Zipped Code Samples

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	3 MB minimum
Operating System:	Windows or Linux. See Appendix A, “Branch Transformation Toolkit development and runtime requirements” on page 477 for detailed requirements
Processor:	Pentium® 1 GHZ or higher
Memory:	512 MB or higher

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Abbreviations and acronyms

API	application programming interface
BMS	basic mapping support
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
CCI	Common Client Interface
CHA	Common Hierarchical Area
CMP	container managed persistence
CMP	Container Managed Persistence
CMP	Container-Managed Persistence
Context	context=
CRUD	create, retrieve, update, and delete
CSS	Cascading Style Sheet
EAR	Enterprise Application Archive
EBCDIC	Extended Binary Coded Decimal Interchange Code
EIS	Enterprise Information System
EJB	Enterprise JavaBean
EJBQL	EJB Query Language
EMF	Eclipse Modeling Framework
FDML	Flow Definition Markup Language
GUI	graphical user interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP over SSL
IBM	International Business Machines Corporation
IDE	integrated development environment
ITSO	International Technical Support Organization
J2EE	Java 2 Platform, Enterprise Edition
JAAS	Java Authentication and Access Service
JAR	Java archive
JCA	J2EE Connector Architecture
JMS	Java Message Service
JSF	JavaServer Faces
JSP	JavaServer Pages
JVM	Java Virtual Machine
MFS	Message Format Service
MSR	magnetic stripe reader
SAE	Single action EJB
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
WAR	Web Application
WAR	Web archive
WAR	Web archiving
WDO	WebSphere Data Objects
WLM	Workload Management
WSDL	Web Services Description Language
WSIF	Web Services Invocation Framework
XPATH	XML Path Language

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 about ordering these publications, see "How to get IBM Redbooks" on page 509. Note that some of the documents referenced here may be available in softcopy only.

- *WebSphere Studio Application Developer Version 5 Programming Guide*, SG24-6957
- *Exploring WebSphere Studio Application Developer Integration Edition 5.0*, SG24-6200
- *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318

Online resources

These Web sites and URLs are also relevant as further information sources:

- Apache Struts

  http://struts.apache.org/index.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

IBM Branch Transformation Toolkit 5.1 Migration and Usage Guidelines

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM Branch Transformation Toolkit 5.1

Migration and Usage Guidelines

Migrate from Branch Transformation Toolkit 4.3 to 5.1

Build applications using Branch Transformation Toolkit 5.1

Test and troubleshoot your application

This IBM Redbook shows in detail IBM Branch Transformation Toolkit for WebSphere Studio Version 5.1 and explains how to migrate from Branch Transformation Toolkit Version 4.3 to Branch Transformation Toolkit 5.1. We provide guidelines for the architecture of the target solution, the correct use of migration tools, and migration-sizing. This book also describes how to build new applications in Branch Transformation Toolkit V5.1. We explain both top-down and bottom-up development, and discuss how to use WebSphere Studio and Branch Transformation Toolkit development plug-ins. We also describe rich Java client development using the Branch Transformation Toolkit.

This book is intended for the following audiences:

- Solution architects who require a description of IBM Branch Transformation Toolkit for WebSphere Studio and how to build a solution
- IT professionals and executives who require a broad understanding of the architecture of the Branch Transformation Toolkit and its implementation
- Readers who are familiar with object-oriented software and related development techniques, and have a general knowledge of Java 2 Platform, Enterprise Edition (J2EE) and related technologies

SG24-7160-00 ISBN 0738495530