Tutorial

Versata Business Logic Designer for WebSphere Studio
Notice

This offering is based on technology from the Eclipse Project
http://www.eclipse.org

Copyright

Copyright © 2000-2003 Versata, Inc. All rights reserved. Printed in the United States of America.
This software and documentation package contains proprietary information of Versata, Inc. and is provided under a license agreement containing restrictions on use and disclosure. The software and documentation is also protected under copyright law. Reverse engineering of the software is prohibited.
The information in this document is subject to change without notice. Versata, Inc. provides this publication "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties or conditions of merchantability or fitness for a particular purpose.
Versata Business Logic Designer and Versata Logic Server are trademarks of Versata, Inc.
IBM, the Ready for WebSphere Studio Software Emblem, and the term Ready for WebSphere Studio Software are trademarks of IBM and are used under license from IBM. IBM does not warrant and is not responsible for the operation of this software. The developer of this software has performed all testing, including all testing required by the Ready for WebSphere Studio Software offering. For more information about the IBM Ready for WebSphere Studio Software offering, visit www.ibm.com/partnerworld/developer/websphere/ready.html
IBM and DB2 are registered trademarks and AIX, WebSphere, and WebSphere Studio Application Developer are trademarks of IBM Corporation.
Microsoft, Microsoft SQL Server, Microsoft Internet Explorer, Windows, Windows NT, and Microsoft Access are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
Oracle and SQL*Plus are registered trademarks and SQL*Net is a trademark of Oracle Corporation.
Java, JavaBeans, JavaScript, JDBC, JDK, JNDI, and Solaris are trademarks and Sun Microsystems is a registered trademark of Sun Microsystems, Inc.
Adaptive Server Enterprise, jConnect, and Sybase SQL Server are trademarks of Sybase, Inc. in the United States and/or other countries.
Informix Dynamic Server and Informix-Driver for JDBC are trademarks and Informix is a registered trademark of Informix Corporation.
Adobe, the Adobe logo, Acrobat, and the Acrobat logo are trademarks of Adobe Systems Incorporated.
All company, product, service, and trade names referenced may be service marks, trademarks, or registered trademarks of their respective owners.
Table of Contents

Introduction......................................................................................................................... 5
  Understanding the Versata Business Logic Designer for WebSphere Studio ......................... 6
  Understanding declarative business logic.................................................................................... 7

Tutorial Overview................................................................................................................. 9
  Summary of tutorial tasks .............................................................................................................. 10
  Understanding the TradeTutorial enterprise application............................................................. 11
    Understanding the TradeTutorial user interface ....................................................................... 11
    Understanding the TradeTutorial object model ....................................................................... 12

LESSON 1 Preparing for Business Logic Development................................................ 15
  Setting up the environment ......................................................................................................... 16
    Verifying DB2 and creating a database ...................................................................................... 16
    Starting WebSphere Studio Application Developer .................................................................. 17
    Reviewing the Versata Business Logic Designer plug-in ......................................................... 17
  Setting up the tutorial project modules ..................................................................................... 20
    Creating a TradeTutorial Enterprise Application project ......................................................... 20
    Building the new TradeTutorial project ................................................................................... 21
    Populating the TRADE database ............................................................................................. 22
  Verifying deployment and database connectivity ..................................................................... 24
    Reviewing transaction logic beans in WebSphere Studio Application Developer ...................... 24
    Starting the server .................................................................................................................... 25
    Configuring database connectivity in the Versata Logic Server Console ............................ 26
LESSON 2 Declaring Initial Business Rules ................................................................. 39
  Declaring rules for the TRANSACTION object .......................................................... 30
    Defining a validation for transaction type ................................................................. 30
    Defining initial value and constraint for quantity ....................................................... 31
    Defining a parent replicate rule for price ................................................................. 31
    Defining a formula rule for amount .......................................................................... 32
  Declaring rules for the HOLDING object .................................................................. 33
    Defining a formula rule for quantity on hand ........................................................... 33
    Defining sum rules for quantity bought and sold ...................................................... 34
    Modifying referential integrity enforcement ............................................................ 34

LESSON 3 Extending Rules with Java Methods .......................................................... 45
  Implementing a requirement to update account balances ............................................. 36
    Adding methods to the object’s Java implementation file ........................................ 36
    Defining action rules to update balance after each transaction ............................... 37

LESSON 4 Extending the Object Model with Derived Attributes .............................. 49
  Implementing a requirement to calculate commissions ............................................... 40
    Adding a derived attribute for holding’s current value .............................................. 40
    Adding a derived attribute for account’s portfolio value .......................................... 41
    Adding a rule to determine account’s commission rate ............................................. 41
    Adding a rule to determine transaction commission rate ......................................... 42
    Adding a derived attribute for holding’s total commissions ...................................... 42
    Updating the formula rule for transaction amount ................................................ 43

LESSON 5 Testing business logic ................................................................................ 53
  Rebuilding Java artifacts ............................................................................................. 46
  Running the Trade Tutorial user interface ................................................................. 47
    Starting the application and entering basic data ..................................................... 47
    Testing business logic for transactions .................................................................. 51
    Tracing rule execution ............................................................................................. 55
Introduction
Understanding the Versata Business Logic Designer for WebSphere Studio

The Versata Business Logic Designer for WebSphere Studio is a plug-in to the IBM WebSphere Studio Application Developer. This plug-in provides a Business Logic perspective in WebSphere Studio Application Developer, where you can develop the business logic tier for a WebSphere enterprise application. In the Business Logic perspective, you begin by creating an enterprise application project. You then can create or import the “structure” of business objects, meaning object fields and keys, into this project. Next, you can specify business logic declaratively for business objects, using the Business Logic perspective’s integrated wizards and designers.

You can use WebSphere Studio Application Developer facilities to import business objects from a variety of external data formats. These objects then can be converted to Versata business logic components, called transaction logic beans. You can review and modify transaction logic beans, using the views available in the Business Logic perspective. You can make the following modifications:

- Modify the “structure” of business objects (object fields and keys)
- Specify associations (relationships) and integrity handling between objects
- Declare business rules that control the execution of sophisticated transactions on the server, including attribute validations, aggregates, formulas, actions, and constraints.

Once you have declared business rules, you can use WebSphere Studio Application Developer facilities to construct a set of Java artifacts from these business logic definitions, and to deploy these business logic components into a WebSphere Application Server as part of an enterprise application. As business requirements change, you can modify business rules in the Business Logic perspective, then rebuild and redeploy components. All updating is automated, including any logic dependencies.

At run time, the deployed business logic components execute using services provided by the Versata Logic Server. The Versata Business Logic Designer includes a development version of the Versata Logic Server that is installed into WebSphere Application Server. The Versata Logic Server services implement many J2EE “Best Practices”, including lazy object instantiation, optimistic locking, and a shared transaction cache, in order to improve business logic performance and flexibility.

Versata business logic components can be accessed by any client application through their standard EJB interfaces, through a JSP tag library provided by Versata, or through the JDBC-like Versata client libraries.
Understanding declarative business logic

The Versata Logic Server captures and executes declarative business logic for large, J2EE-based application systems. Business logic is the processing that occurs during transaction processing, independent of the source of the transaction. Declarative business logic specifies the desired result of transactions in business oriented terms (‘what’), rather than in the technology-oriented terms (‘how’) associated with procedural approaches. The declarative approach provides significant improvements in time to market and maintenance, dramatically reduces the complexity and risk of large projects, and enables non-technical staff to cooperate effectively with IT to deliver business systems.

Declarative processing is performed in the context of underlying strategic technology, such as application servers and J2EE. Declarative logic is deployed as industry standard components, executable on the Versata Logic Server engine that operates within the context of WebSphere Application Server. Versata components can be extended with non-automated logic using conventional procedural techniques, such as Java event handlers, inheritance, and delegation. The Versata Logic Server manages the execution of declarative business logic across multiple objects. This cross-object logic is automatically reused across all transactions, ordered based on dependencies to maintain correctness, and optimized for performance.
INTRODUCTION

UNDERSTANDING DECLARATIVE BUSINESS LOGIC
Tutorial Overview
Summary of tutorial tasks

This tutorial serves as a hands-on introduction to the functionality of the Versata Business Logic Designer for WebSphere Studio. This document walks you through the tasks involved in constructing and testing the declarative business logic for a J2EE application. The tutorial includes the following lessons.

- “Preparing for Business Logic Development” on page 15
  This lesson walks you through the tasks of setting up the environment and creating an instance of the tutorial enterprise application in your WebSphere Studio Application Developer workspace.

- “Declaring Initial Business Rules” on page 29
  This lesson walks you through the definition of business rules in the WebSphere Studio Application Developer Business Logic perspective’s Transaction Logic Bean Editor. Types of rules covered include value table validation, formulas, sums (aggregates), parent replicates, constraints, and referential integrity enforcement.

- “Extending Rules with Java Methods” on page 35
  This lesson walks you through the tasks for enabling Java method calls from declarative business rules. These tasks include the addition of Java method code to an object’s Java implementation file and the definition of action rules.

- “Extending the Object Model with Derived Attributes” on page 39
  This lesson walks you through the tasks that allow calculation of values not stored in the database and use of these values for declarative business logic. These tasks include the addition of non-persistent attributes to the project, the definition of derivation rules for these new attributes, and the definition of derivation rules for other, dependent attributes.

- “Testing business logic” on page 45
  This lesson discusses how to run the tutorial’s JSP application to test business logic.
Understanding the TradeTutorial enterprise application

The business logic built in this tutorial is designed for an enterprise application named TradeTutorial, based on IBM’s WebSphere performance benchmark application named Trade.

The IBM Trade application is an end-to-end web application modeled after an online brokerage. Trade leverages J2EE components to provide a set of user services, such as login/logout, stock quotes, buy, sell, and account details, through a standards based HTTP protocol. The TradeTutorial application supports a subset of these services.

You install the files for the TradeTutorial enterprise application by using WebSphere Studio Application Developer’s New Project wizard to create an instance of the TradeTutorial example in your workspace. For instructions, see “Creating a TradeTutorial Enterprise Application project” on page 20.

Note: In addition to the TradeTutorial enterprise application, the Versata Business Logic Designer also includes a TradeAdvanced example. You can create an instance of this example to review the implementation of the rules defined in this tutorial, as well as examples of common customizations. For information, see the Examples documentation.

Understanding the TradeTutorial user interface

As part of the tutorial, you will execute the TradeTutorial application user interface to test the business logic you define in tutorial tasks. The TradeTutorial application user interface supports:

- Logging a user onto the system
- Viewing a portfolio (a set of holdings)
- Finding the quoted price for a stock
- Buying and selling holdings

The TradeTutorial application user interface is installed as the TradeTutorialJSP.war web module when you create an instance of TradeTutorial example in your workspace.
Understanding the TradeTutorial object model

This tutorial’s tasks build business logic on an object model representing online brokerage data for the TradeTutorial application. This model includes the following objects:

- **ACCOUNT.** Represents a user account.
- **HOLDING.** Represents a set of a particular stock’s shares owned by an account.
- **PROFILE.** Represents information about a user.
- **QUOTE.** Represents a price for a share of a particular stock.
- **TRANSACTION.** Represents an instance where a set of a particular stock’s shares are bought or sold.
- **TRANSTYPE.** Represents valid types of transactions, defined as buy and sell. Serves as a value table. (Note that you will need to populate this table with data as part of setup, so that this data can be used in validation rules.)
These objects’ attributes, keys, and relationships are illustrated in the following diagram.

The TradeTutorial object model is installed as the TradeTutorialLogic.jar EJB module when you create an instance of TradeTutorial example in your workspace. During the tutorial, you will export the associated database schema to a DB2 UDB database, so you can maintain a persistent data model.
Note: The tutorial demonstrates Versata’s virtual attributes feature. This feature allows you to add attributes with values that are calculated for the purpose of derivation rules, without persisting these attributes in a data source. Therefore, during the tutorial you will add attributes to the business logic object model, without adding them to the database. For more information, see “Extending the Object Model with Derived Attributes” on page 39.
LESSON 1

Preparing for Business Logic Development
Setting up the environment

Before you begin defining declarative business rules, you need to verify that all required programs are installed and running. Also, it is a good idea to become familiar with the interface for the Versata Business Logic Designer for WebSphere Studio. This section walks you through these tasks.

The following products are prerequisites for using this tutorial:

- IBM WebSphere Studio Application Developer GM
- DB2 UDB 7.2

Note: The tutorial delivered with the Versata Business Logic Designer for WebSphere Studio uses an object model based on an IBM DB2 UDB database and ships with configuration files for DB2. You may use another type of database, but you will need to create your own database configuration scripts (DDL and DML, or XMI files using WebSphere Studio Application Developer’s Data perspective). Also, the tutorial has not been tested with databases other than DB2.

Verifying DB2 and creating a database

You need to create a database called TRADE in DB2 to hold the deployed object model for the tutorial.

Note: If you already have a database called TRADE that you do not want to overwrite, you can create a database with variation on this name, such as TRADETUT.

As you are working with DB2, it is recommended that you log in as the db2admin user, with the correct password for that user. The schema provided for the tutorial is designed for that user.

To verify your DB2 installation:

1. From the Start menu, open the Services dialog as follows:
   - Windows NT. Choose Start>Settings>Control Panel>Services.

2. In the Services dialog, verify the status for DB2 services as follows:
   - DB2 - DB2. Status: started; Startup: automatic
   - DB2 - DB2CTLSV. Status: started; Startup: automatic.
To create a database for the tutorial object model:

1. From the Start menu, choose Programs>IBM DB2>Control Center.
2. In the Control Center hierarchy, expand the machine, the Instances folder, and the DB2 folder.
3. Right-click the Databases folder and choose Create>Database Using Wizard.
4. In the Create Database Wizard, enter a database name of TRADE (or another name of your choosing, then click Finish to accept defaults for other options.
5. After the listing new database appears in the Control Center window, close it.

Starting WebSphere Studio Application Developer

All tutorial tasks are completed in the IBM WebSphere Studio Application Developer environment. This environment includes a development WebSphere Application Server, where you can deploy objects for review and testing.

Note: Be sure to open WebSphere Studio Application Developer to a new workspace each time you run through this tutorial.

To start WebSphere Studio Application Developer:

1. From the Start menu, choose Programs>IBM WebSphere Studio>Application Developer.
2. In the dialog that appears requesting a workspace specification, enter a descriptive directory name, such as workspace-tutorial, and click OK.

Note: If the workspace specification dialog does not open when you run WebSphere Studio Application Developer, run it from the command line with the following option: <IBM WebSphere Studio Application Developer Install Directory>\wsappdev.exe -setworkspace.

Reviewing the Versata Business Logic Designer plug-in

Once you have started WebSphere Studio Application Developer, you need to verify that the Versata Business Logic Designer plug-in is available. The tutorial’s rules declaration tasks require the use of the plug-in’s Business Logic perspective.

To verify Versata Business Logic Designer installation:

1. From the WebSphere Studio Application Developer main menu, choose Help>About IBM WebSphere Studio Application Developer.
2. The following icon should be visible in the dialog that appears:
3. Click the Feature Details button and, in the dialog that appears, scroll down to the Versata listing, select it, and click the Plug-In Details button.
   The dialog should list the following plug-ins:
   - Versata Business Logic Designer
   - Versata Business Logic Designer Examples
   - Versata Business Logic Designer Help
   - Versata Business Logic Designer Model
   4. Click OK to close all of the help dialogs.

To open the Business Logic perspective:
1. From the WebSphere Studio Application Developer main menu, choose Window>Open Perspective>Other.
2. In the Select Perspective dialog, choose Business Logic and click OK.
   Notice that the Business Logic perspective is represented by the following button in WebSphere Studio Application Developer’s left vertical toolbar:

   ![Business Logic perspective button]

   The Versata Business Logic Designer Welcome page should be in WebSphere Studio Application Developer’s upper right window. This page provides introductory information about the Versata Business Logic Designer and links to additional information. If this page does not open automatically, you can open it manually.

   To open the Versata Business Logic Designer plug-in Welcome page:
   1. From the WebSphere Studio Application Developer main menu, choose Help>Welcome.
   2. In the Welcome dialog, choose Versata Business Logic Designer and click OK.

   To review the views available in the Business Logic perspective:

   Note the following views are available in WebSphere Studio Application Developer’s Business Logic perspective:
   - In the upper left window:
     - Navigator view. Lists all files in the current project, grouped in the same manner as Navigator listings for the J2EE perspective, with the addition of a Servers group.
     - J2EE Hierarchy view. Lists all J2EE-related files, grouped in the same manner as J2EE Hierarchy listings for the J2EE perspective.
     - Business Logic view. Lists transaction logic beans.
In the upper right window:

Editors for selected objects, layered on top of each other, including the following:

- Transaction Logic Bean Editor. When a transaction logic bean is selected, this multiple tabbed view provides editors and wizards where you can review and modify transaction logic beans. For more details about the tabs available in this view, see page 24.
- Association Editor. When the Details button for an association listed in the Transaction Logic Bean Editor is clicked, this editor allows you to review and modify an association’s properties.
- WebSphere Studio Application Developer Java editor. When a transaction logic bean’s Java implementation file is selected, displays the object’s Java implementation code for review and/or modification.
- Other WebSphere Studio Application Developer editors. When another type of project file is selected, displays its contents for review.

In the lower left window:

Outline. Lists contents of the object displayed in the upper right editor window. For example, when the Transaction Logic Bean Editor is open, the outline lists the transaction logic bean’s attributes and rules. When a Java editor is open for an implementation file, the outline lists its imported packages, class, declared variables, and methods.

In the lower right window:

Standard WebSphere Studio Application Developer Tasks, Servers, and Console tabs.
Setting up the tutorial project modules

Before you begin business logic definition, you need to set up an enterprise application project in WebSphere Studio Application Developer for the TradeTutorial application. The Versata Business Logic Designer provides a TradeTutorial example that you can use as a basis. You need to build this project, export the project's object model to the DB2 database you created, and set up database connectivity information in the Versata Logic Server Console.

Creating a TradeTutorial Enterprise Application project

To create a TradeTutorial Enterprise Application project in WebSphere Studio Application Developer:

1. Click the button to open the Business Logic perspective and click the J2EE Hierarchy view so it is displayed.
2. Right-click in the J2EE Hierarchy view, and choose New>Project.
3. In the left pane of the New Project dialog, expand Examples and select Business Logic. In the right pane, select TradeTutorial. Click Next.
4. In the Create Versata Business Logic Trade Tutorial dialog, accept the defaults and click Finish.
5. The TradeTutorial project should now be visible under the Enterprise Applications node of the J2EE Hierarchy view. When you expand its Modules node, you should see listings for TradeTutorialJSP.war, TradeTutorialLogic.jar, and VLSEJB.jar.

Note: If the WebSphere Studio Application Developer’s Preferences dialog is set to Perform build automatically upon resource modification, a dialog may appear asking whether to update the server configuration. Although this dialog may look like an error, this behavior is expected and does not indicate any problems. You should accept this update.
PREPARING FOR BUSINESS LOGIC DEVELOPMENT
SETTING UP THE TUTORIAL PROJECT MODULES

Note: The VLSEJB.jar module contains the PLSContextHome and VLSContextHome beans that provide system functionality for the Versata Business Logic Designer.

The TradeTutorialLogic EJB project includes a system bean called VLSUtil, that provides utility services for the EJB module.

Building the new TradeTutorial project

The next step is to build the new enterprise application project, including all of its modules.

To build the new project:

1. From the WebSphere Studio Application Developer main menu, choose Project>Rebuild All.

When the TradeTutorial project is built, the Versata Business Logic Designer plug-in creates a server called Server-TradeTutorial.wsi and a server configuration called Config-TradeTutorial.wsc.
Note: If the server and server configuration are not visible on the J2EE Hierarchy tab after the build, close the project, then reopen it. The server and server configuration should now be visible.

Populating the TRADE database

The TradeTutorialLogic project includes database definition files in XMI format, that you can use to populate the database tables required to persist the attributes of this tutorial’s transaction logic beans.

Note: The pre-supplied configuration files are for DB2 UDB 7.2 only. To review the supplied database configuration information, expand the Databases node on the J2EE Hierarchy tab, then continue expanding nodes until table listings are visible. You can select a table node to review its details in a WebSphere Studio Application Developer editor. You should have set up a TRADE database (or a database with a name of your choice) in a previous step. If you have not yet done this, see “Verifying DB2 and creating a database” on page 16, before proceeding with the following data export task.
As you are working with DB2, it is recommended that you log in as the db2admin user, with the correct password for that user. The schema provided for the tutorial is designed for that user.

To export TradeTutorialLogic data to the DB2 TRADE database:

1. On the J2EE Hierarchy tab, right-click the TradeTutorialLogic: TRADE (DB2 UDB V7.2) node under the Databases node, and select Export to server.

2. In the first Data Export dialog, make sure all check boxes are enabled (so all objects are exported), and click Next.

3. In the Data Export Options dialog, choose Commit changes only upon success. Also, if you have exported the model to the server previously, enable the Drop statements check box. Click Next.

4. In the Database Connection dialog, set the connection name to TradeConnection, the database name to TRADE (or to the other name you chose), and the User ID and Password according to your DB2 setup. Then click Finish to complete the export.
Before you begin defining declarative business rules, it is a good idea to verify that transaction logic beans have been deployed successfully, and that connectivity between the run-time Versata Logic Server and the DB2 TRADE database has been configured successfully.

**Reviewing transaction logic beans in WebSphere Studio Application Developer**

To review transaction logic beans:

1. In the Business Logic perspective, click the Business Logic tab. A TradeTutorialLogic node should appear. Expand this node to view the transaction logic beans for ACCOUNT, HOLDING, PROFILE, QUOTE, TRANSACTION, and TRANSTYPE listed below it.

   ![Figure 4 Business Logic view listing TradeTutorialLogic transaction logic beans](image)

2. On the Business Logic view, double-click the ACCOUNT transaction logic bean. Review the tabs that appear in the ACCOUNT Logic Bean Editor. This view includes the following tabs:
   - Overview tab. Provides a high-level view of a transaction logic bean’s characteristics. On this tab, you can add or remove attributes, associations, constraints, and action rules for the transaction logic bean. You also can add or modify description text and extended properties for the bean, and edit its data source information and Java implementation file.
Attributes tab. Provides details about selected transaction logic bean attributes. On this tab, you can add or remove attributes, and define derivation and values list validations for selected attributes. You also can modify the data type, add or modify description text, and add or modify extended properties for a selected attribute.

Constraints tab. Provides details about selected transaction logic bean constraints. On this tab, you can add, remove, or modify constraints.

Actions tab. Provides details about selected transaction logic bean action rules. On this tab, you can add, remove, or modify action rules.

3. If desired, review the Logic Bean Editor for other transaction logic beans.

Starting the server

You should attempt to start the server for the TradeTutorial project in order to verify that the development WebSphere Application Server is problem-free. Also, the server needs to be running in order to test database connectivity in the Versata Logic Server Console.

To start the server:

1. Click the Servers tab in the lower right corner.
2. On the Servers tab, right-click Server-TradeTutorial.wsi and choose Start.
   
   The server attempts to start, logging its output to the Console tab. When the server has started successfully, a message ending with Server server1 open for e-business appears.

Note: If you receive an error that publishing failed, double-click Server-TradeTutorial.wsi on the J2EE hierarchy tab to open its editor. Respond yes to the WebSphere Studio Application Developer warning dialog asking if you want the server to be reconfigured automatically. When the server editor opens, close it and click the Yes button in the dialog asking whether to save changes. Then reattempt to start the server.
The following figure illustrates the messages that appear when the server starts successfully.

![Console output for successful server start](image)

If problems occur, choose the Project>Rebuild All menu option again, then try to restart the server.

**Configuring database connectivity in the Versata Logic Server Console**

At run time, business logic is executed by the Versata Logic Server. This business logic may result in updated data values. To ensure that these updates complete successfully, you need to configure connectivity between the Versata Logic Server and the database that stores data for transaction logic beans. The Versata Logic Server Console provides a user interface where you can perform this configuration. In this task you configure connectivity for the TRADE database and its associated transaction logic beans.

*To configure connectivity between the Versata Logic Server and the TRADE database:*

1. Click the button to open the Business Logic perspective.
2. From the main toolbar, click the following button to start the Versata Logic Server Console:

![Start button](image)

3. After some time, when the Login dialog appears, enter sa as the user, do not complete the other fields, then click OK.
4. In the Versata Logic Server Console, expand the machine node, the Administration node, the Object Packages node, the TradeTutorialLogic_blb node, and the Data Objects node. Six data objects should be listed: ACCOUNT, HOLDING, PROFILE, QUOTE, TRANSACTION, and TRANSTYPE.

26
5. Expand the Data Servers node, then right-click DataServer0 and choose Change Name. In the dialog, enter Tutorial. Click the Save button on the Versata Logic Server Console main toolbar.

6. Enter configuration values in the Tutorial data server editor’s Connection Properties tab.
   - For Data Server type, choose DB2-UDBjdbc from the dropdown.
   - Enter your database login and password. (Note: As you are working with DB2, it is recommended that you log in as the db2admin user, with the correct password for that user. The schema provided for the tutorial is designed for that user.)
   - Set the Database to TRADE (or to the other name you chose).
   - Set the Schema to DB2ADMIN.

7. Click the Save button on the Versata Logic Server Console main toolbar.
8. Test the connection with the database by clicking the Test Connection button. A dialog should be displayed indicating that the test was successful. Close this dialog and the Versata Logic Server Console window and its DOS window.

![Testing the connection in the Versata Logic Server Console](image.png)
LESSON 2

Declaring Initial Business Rules
Declaring rules for the TRANSACTION object

In the TradeTutorial application, in order to buy or sell stock, users enter transactions. This section walks you through the creation of declarative business rules on the TradeTutorial object model’s TRANSACTION object. You use the Versata Business Logic Designer for WebSphere Studio to define rules on the object’s attributes.

Once Java artifacts are built and deployed to the server, these rules provide processing for the entry of transactions, determination of prices, and determination of transaction amounts when users run the TradeTutorial user interface, a JSP application.

These rules are designed to meet the following requirements:
- There are two types of transactions: buy and sell.
- Each buy transaction must purchase at least five shares.
- A transaction price is obtained from the quote object.
- The amount of a transaction is derived by multiplying price times quantity.

**Note:** You can only test the rules defined in this section by running the TradeTutorial user interface after you have completed all of the tutorial’s business logic definition tasks. For information, see “Testing business logic” on page 45.

Defining a validation for transaction type

The TRANSACTION object includes a transType attribute. There is a business requirement to limit valid values for this attribute to “buy” or “sell”. In order to enforce this requirement, you can define a validation rule that checks the value of this attribute against valid type values stored in the TRANSTYPE object.

**To define a validation rule for the transType attribute:**

1. In the Business Logic perspective, click the Business Logic view. Double-click the TRANSACTION node.
2. On the Overview tab, select the transType attribute and click the Details button.
3. On the Attributes tab, click the Add button under Validation.
4. In the Create Validation Rule dialog, choose ValidValues from the first dropdown. Choose TRANSTYPE for the validating object, and choose type for the validating attribute. Click Finish.
5. Click the Save button on the main toolbar.
DECLARING INITIAL BUSINESS RULES
DECLARING RULES FOR THE TRANSACTION OBJECT

Note: By default WebSphere Studio Application Developer builds and compiles an object each time you save changes. If you want to disable this setting, from the main toolbar, choose Window>Preferences. In the Preferences dialog, disable the check box for Perform build automatically on Resource Modification. Note that the first build is the slowest. Later rebuilds and compiles will be significantly quicker.

Defining initial value and constraint for quantity
You perform two definitions in order to enforce the business requirement that each buy transaction must purchase at least five shares. The first definition sets the initial value for TRANSACTION’s quantity attribute to 5. This initial value displays whenever a user creates a new transaction, and remains unless the user changes the value. The second definition is a constraint that rejects transactions where the value for quantity is less than 5.

To set the initial value for the quantity attribute:
1. On the Attributes tab of the TRANSACTION Logic Bean Editor, select the quantity attribute.
2. In the Initial Value box, enter 5.
3. Click the Save button on the main toolbar.

To define a constraint for quantity:
1. Click the Constraints tab.
2. Click the Add button.
3. In the Add Constraint dialog, enter a Name of MinQuantity. In the Reject When box, enter quantity < 5. Click Finish.
4. Click the Save button on the main toolbar.

Defining a parent replicate rule for price
In order to enforce the requirement that a share price for a transaction is obtained from the quote object, you define a parent replicate rule on TRANSACTION’s price attribute.

To define a parent replicate for the price attribute:
1. Click the Attributes tab.
2. Select the price attribute and click the Add button under Derivation.
3. In the Create Derivation Rule dialog, choose ParentReplicate from the Derivation dropdown. Choose Quote from the Role Name dropdown, and price from the Related Attribute dropdown. Do not enable the Maintained check box. Click Finish.
4. Click the Save button on the main toolbar.

**Defining a formula rule for amount**

In order to enforce the requirement that the amount of a transaction is equal to the price times the quantity you define a formula rule on TRANSACTION’s amount attribute.

**Note:** You will update this formula rule in a later chapter to meet changing business requirements.

*To define a formula on the amount attribute:*

1. On the Attributes tab, select the amount attribute.
2. Click the Add button under Derivation.
3. In the Create Derivation Rule dialog, choose Formula from the Derivation dropdown.
4. For the expression, enter `price * quantity`. Click Finish.
5. Click the Save button on the main toolbar.
6. Close the TRANSACTION Logic Bean Editor.
Declaring rules for the HOLDING object

In the TradeTutorial model, the HOLDING object contains summary information for all of a particular account’s transactions. This section walks you through the creation of declarative business rules on this object. These rules allow tracking of each account’s stock transactions. These rules are designed to meet the following requirements:

- An account’s quantity on hand of a particular stock is equal to the quantity bought minus the quantity sold.
- An account’s quantity bought is equal to the sum of the quantity attribute values for all account’s transactions of the type “buy”.
- An account’s quantity sold is equal to the sum of the quantity attribute values for all account’s transactions of the type “sell”.
- A HOLDING object is created automatically for an account when its initial buy transaction occurs.

Note: You can test the rules defined in this section by running the TradeTutorial user interface after you have completed all of the tutorial’s business logic definition tasks. For information, see “Testing business logic” on page 45.

Defining a formula rule for quantity on hand

In order to enforce the requirement that the quantity on hand of a holding is equal to the quantity bought minus the quantity sold, you define a formula rule on HOLDING’s qtyOnHand attribute.

To define a formula on the qtyOnHand attribute:

1. In the Business Logic view, double-click HOLDING to open the Transaction Logic Bean Editor.
2. On the Overview tab, select the qtyOnHand attribute and click the Details button.
3. On the Attributes tab, click the Add button under Derivation.
4. In the Create Derivation Rule dialog, choose Formula from the Derivation dropdown.
5. For the expression, enter qtyBought - qtySold. Click Finish.
6. Click the Save button on the main toolbar.
**Declaring Initial Business Rules**

Declaring rules for the HOLDING object

---

**Defining sum rules for quantity bought and sold**

In order to enforce the requirements for calculation of values for quantity bought and sold, you define sum rules on HOLDING’s qtyBought and qtySold attributes.

*To define sum rules for the qtyBought and qtySold attributes:*

1. On the Attributes tab, select the qtyBought attribute.
2. Click the Add button under Derivation.
3. In the Create Derivation Rule dialog, choose Sum from the Derivation dropdown.
4. For the Role Name, choose Transactions. For the Related Attribute, choose quantity.
5. For the expression, enter `transType = 1 /*Buy*/`. Click Finish.
6. Click the Save button on the main toolbar.
7. On the Attributes tab, select the qtySold attribute.
8. Click the Add button under Derivation.
9. In the Create Derivation Rule dialog, choose Sum from the Derivation dropdown.
10. For the Role Name, choose Transactions. For the Related Attribute, choose quantity.
11. For the expression, enter `transType = 2 /*Sell*/`. Click Finish.
12. Click the Save button on the main toolbar.

---

**Modifying referential integrity enforcement**

In order to enforce the requirement that a HOLDING object is created at the time of an account’s first transaction, you modify the referential integrity rules for the association between the HOLDING and TRANSACTION objects.

*To modify referential integrity for the HOLDING-TRANSACTION association:*

1. Click the Overview tab of the HOLDING Transaction Logic Bean Editor.
2. Under Associations, select Transactions and click the Details button.
3. Under Rule for Child Inserts and Updates, choose InsertParentIfNone from the dropdown.
4. Click the Save button on the main toolbar.
5. Close the HOLDING Logic Bean Editor and the HOLDING-TRANSACTION Association Editor.
LESSON 3

Extending Rules with Java Methods
Implementing a requirement to update account balances

In the scenario for this tutorial, after you have implemented initial business requirements by declaring business rules, a new requirement arises. The Versata Business Logic Designer for WebSphere Studio allows you to implement this new requirement by declaring some new rules for transaction logic beans. The Versata Logic Server handles all logic dependencies when you rebuild the project in a later lesson.

The new requirement is for the account balance to be updated after each transaction. You implement this requirement by adding code to the ACCOUNT object’s Java implementation file. The Versata Business Logic Designer exposes Java code so you can add your own code in a straightforward manner. Once you have added Java methods for updating account balance, you define an action rule to call each method when defined conditions are met.

Note: You can test the rules defined in this section by running the TradeTutorial user interface after you have completed all of the tutorial’s business logic definition tasks. For information, see “Testing business logic” on page 45.

Adding methods to the object’s Java implementation file

You need to add two methods to the ACCOUNT object’s Java implementation file in order to implement the update account balance requirement. One method, called debit, debits the account balance when a buy transaction occurs. The other method, called credit, credits the account balance when a sell transaction occurs.

To add Java methods to the ACCOUNT implementation file:

1. In WebSphere Studio Application Developer’s Business Logic perspective, double-click ACCOUNT in the Business Logic view.
2. On the Overview tab of the ACCOUNT Logic Bean editor, scroll down to the Java section. Notice that two files are listed. The ACCOUNTBaseImpl file includes all generated code, and should not be edited. You need to add code to the ACCOUNTImpl file.
3. Select the ACCOUNTImpl file and click the Open button to open the file in a Java editor.
   
   Note: You also can open ACCOUNTImpl by double-clicking it on the Navigator tab (located in the TradeTutorialLogic folder in the EJB Project folder).
4. Add the following code at the end of the file, before the closing bracket.
5. Click the Save button on the main toolbar.
6. Close the Java editor and the ACCOUNT Logic Bean Editor.

Defining action rules to update balance after each transaction

Account balances need to be updated based on values for TRANSACTION object attributes. In order to implement the account balance update requirement, you need to define two action rules on the TRANSACTION object. These actions call the new methods you just added to the ACCOUNTImpl file.

The first action rule calls the debit method when a buy transaction occurs and a new TRANSACTION object is inserted. The second action rule calls the credit method when a sell transaction occurs and a TRANSACTION object is deleted. As you define these rules, notice that they rely on the association between TRANSACTION and ACCOUNT.

To define an action rule to update account balance when a buy transaction occurs:

1. In the Business Logic perspective, in the Business Logic view, right-click TRANSACTION and choose Add>Action.
2. In the Add Action dialog, enter a name of Buy. For Condition, enter Inserting and transType = 1. For Action To Take, enter getAccount().debit(amount). For Description, enter Debits account balance when stock is bought. Click Finish.

To define an action rule to update account balance when a sell transaction occurs:

1. In the Business Logic perspective, in the Business Logic view, again right-click TRANSACTION and choose Add>Action.
2. In the Add Action dialog, enter a name of Sell. For Condition, enter Inserting and transType = 2. For Action To Take, enter getAccount().credit(amount). For Description, enter Credits account balance when stock is sold. Click Finish.
3. If you want, you can double-click TRANSACTION in the Business Logic view to review the new actions in the Transaction Logic Bean Editor.
LESSON 4

Extending the Object Model with Derived Attributes
Implementing a requirement to calculate commissions

Once again, after you have declared new rules, another business requirement has arisen. Once again, you can declare additional rules.

This latest requirement is for a commission to be calculated for each transaction, based on the account’s portfolio size. You implement this requirement by adding a series of derivations to arrive at the required calculation. These derivations include a formula for a holding’s current value, a sum for an account’s portfolio value, a formula for an account’s commission rate, and a parent replicate for a transaction’s commission. You also modify the formula for transaction amount, to incorporate commissions.

Some of these derivations involve attributes that currently do not exist in the database. The Versata Business Logic Designer allows you to create non-persistent attributes for the purpose of calculations. These attributes are not stored in the database and are calculated only as necessary for business logic execution.

Note: You can test the rules defined in this section by running the TradeTutorial user interface after you have completed all of the tutorial’s business logic definition tasks. For information, see “Testing business logic” on page 45.

Adding a derived attribute for holding’s current value

In order to enable the calculation of a holding’s current value, you need to add a non-persistent attribute that is calculated by multiplying the holding’s quantity on hand and the price obtained from a quote object.

To enable calculation of a holding’s current value:

1. In the Business Logic perspective, in the Business Logic view, right-click HOLDING and choose Add>Attribute.
2. In the first Add Attribute dialog, enter a name of currentValue, a data type category of Number, and a data type of Double. Disable the Persisted check box. Click Finish.
3. In the next Add Attribute dialog, click Cancel to close it.
4. Double-click HOLDING to open the Transaction Logic Bean Editor.
5. On the Overview tab, select the currentValue attribute and click the Details button.
6. On the Attributes tab, click the Add button under Derivation.
7. In the Create Derivation Rule dialog, choose Formula from the Derivation dropdown.
8. For Expression, enter getQuote().getprice() * qtyOnHand. Click Finish.
9. Click the Save button on the main toolbar.
Extending the Object Model with Derived Attributes
Implementing a Requirement to Calculate Commissions

Adding a derived attribute for account’s portfolio value

In order to enable the calculation of an account’s portfolio value, you need to add a non-persistent attribute that is calculated by summing the current values of all of the account’s holdings.

To enable calculation of an account’s portfolio value:

1. In the Business Logic view, right-click ACCOUNT and choose Add > Attribute.
2. In the first Add Attribute dialog, enter a name of portfolioValue, a data type category of Number, and a data type of Double. Disable the Persisted check box. Click Finish.
3. In the next Add Attribute dialog, click Cancel to close it.
4. Double-click ACCOUNT to open the Transaction Logic Bean Editor.
5. On the Overview tab, select the portfolioValue attribute and click the Details button.
6. On the Attributes tab, click the Add button under Derivation.
7. In the Create Derivation Rule dialog, choose Sum from the Derivation dropdown.
8. For Role Name, choose Holdings. For Related Attribute, choose currentValue. Do not enter an expression. Click Finish.
9. Click the Save button on the main toolbar.

Adding a rule to determine account’s commission rate

In order to enable the calculation of an account’s commission rate, you need to define a formula rule for the ACCOUNT.commRate attribute that provides two different values, depending on the value of the ACCOUNT.portfolioValue attribute. For portfolio values greater than $10,000, the commission for each transaction is $5.00, while for smaller portfolio values, the commission is $10.00.

To enable calculation of an account’s commission rate:

1. On the Attributes tab of the ACCOUNT Logic Bean Editor, select the commRate attribute and click the Add button under Derivation.
2. In the Create Derivation Rule dialog, select Formula from the Derivation dropdown.
3. For Expression, enter the following. Then click Finish.

   If(portfolioValue > 10000) Then
      $value = 5.00
   Else
      $value = 10.00
   End If
4. Click the Save button on the main toolbar.
5. Close the ACCOUNT Logic Bean Editor.

Adding a rule to determine transaction commission rate

In order to enable the calculation of a commission rate for a transaction, you need to add a rule for the TRANSACTION.commission attribute that replicates the value of the parent account’s commission rate.

To enable calculation of a transaction’s commission rate:
1. On the Business Logic view, double-click TRANSACTION.
2. On the Overview tab, select commission and click the Details button.
3. On the Attributes tab, click the Add button under Derivation.
4. In the Create Derivation Rule dialog, choose ParentReplicate from the Derivation dropdown.
5. For Role Name, choose Account. For Related Attribute, choose commRate. Do not enable the Maintained check box. Click Finish.
6. Click the Save button on the main toolbar.

Adding a derived attribute for holding’s total commissions

In order to enable the calculation of total commission charges for each holding, you need to add a non-persistent attribute that is calculated by summing the commissions charged for all of a holding’s transactions.

To enable calculation of a holding’s total commissions:
1. On the Attributes tab of the HOLDING Logic Bean Editor, click the Add button.
2. In the first Add Attribute dialog, enter a name of commissions, a data type category of Number, and a data type of Double. Disable the Persisted check box. Click Finish.
3. In the next Add Attribute dialog, click Cancel to close it.
4. Click the Save button on the main toolbar.
5. On the Attributes tab, click the Add button under Derivation.
6. In the Create Derivation Rule dialog, choose Sum from the Derivation dropdown.
7. For Role Name, choose Transactions. For Related Attribute, choose commission. Do not enter an expression. Click Finish.
8. Click the Save button on the main toolbar.
9. Close the HOLDING Logic Bean Editor.
Updating the formula rule for transaction amount

In order to incorporate commission charges into the amount value for each transaction, you need to modify the formula rule you defined previously for TRANSACTION.amount.

To modify the calculation of transaction amount:

1. On the Attributes tab of the TRANSACTION Logic Bean Editor, select amount.
2. Delete the existing expression and enter the following:

   ```sql
   If(transType = 1/*Buy*/) Then
       $value = (quantity * price) + commission
   ElseIf(transType = 2/*Sell*/) Then
       $value = (quantity * price) - commission
   Else
       $value = quantity * price
   End If
   ```

3. Click the Save button on the main toolbar.
4. Close the TRANSACTION Logic Bean Editor.
EXTENDING THE OBJECT MODEL WITH DERIVED ATTRIBUTES
IMPLEMENTING A REQUIREMENT TO CALCULATE COMMISSIONS
In order to test the business logic you have defined in this tutorial’s lessons, you need to complete the following steps.

1. “Rebuilding Java artifacts” on page 46

2. “Running the Trade Tutorial user interface” on page 47, which includes:
   - “Starting the application and entering basic data” on page 47
   - “Testing business logic for transactions” on page 51

3. “Tracing rule execution” on page 55
Rebuilding Java artifacts

In order to test the business logic implemented by the rules you defined in the preceding lessons, you need to rebuild the TradeTutorialLogic project, then stop and restart the server so it registers changes.

To rebuild the TradeTutorialLogic project:

1. From the WebSphere Studio Application Developer main menu, choose Project>Rebuild All.
2. Click the Servers tab in the lower right window. Right-click Server-TradeTutorial.wsi and choose Stop.
3. Right-click Server-TradeTutorial.wsi and click Restart.
Running the Trade Tutorial user interface

After you have rebuilt the project, and stopped and restarted the server, you can run the Trade Tutorial user interface (a JSP application) to review the execution of business rules.

Starting the application and entering basic data

In order to test business logic for transaction entry, you first need to enter some basic data, including valid transaction types and stock quote data for some example stocks.

Note: Underlying data is cached in the page. Perform an explicit refresh (using the Refresh button on the page) if data updates do not appear when transitioning between related pages.

To run the Trade Tutorial application user interface:

1. In WebSphere Studio Application Developer’s Navigator view, right-click the TradeTutorialJSP node and click Run on Server.
2. In the Server Selection dialog, choose Use an Existing Server and select Server-TradeTutorial.wsi. Click Finish.
3. Messages display on the Console tab, then the JSP page opens in the main WebSphere Studio Application Developer editor window.

![Trade Tutorial Application](image)

**Figure 7** Trade Tutorial Login page

4. On the Login page, enter any name for the Account Name and click the Login button. A new account will be created.

**Note:** If the error “field portfolioValue was not found in DS account” appears, you forgot to stop and restart the server after the project rebuild. Click the Logout button, then stop and restart the server. Now you can retry steps 1-3.
5. A page opens for the new account you created.

6. Click the Transaction Types button.
7. As the text on the Transaction Types page states, you need to define two Transaction Types in order for rules to function correctly. To add a Transaction Type, click the Add button, enter 1 for Type and **Buy** for Description. Click the Add button again, enter 2 for Type and **Sell** for Description.

8. Click the Account button to return to the Account page.

9. Click the Quotes button.
10. On the Quotes page, click the Add button. Enter a symbol, a price, and a description for some stock. (Your entry can be real or fictitious.) Repeat until you have added several stocks, then click the Save button.

11. Click the Account button to return to the Account page.

**Testing business logic for transactions**

You now can create transaction entries for an account, in order to review the operation of business rules you defined for the TRANSACTION, HOLDING, and ACCOUNT objects.

**Note:** Instructions for defining these rules are available in the following sections: “Declaring rules for the TRANSACTION object” on page 30, “Declaring rules for the HOLDING object” on page 33, “Implementing a requirement to update account balances” on page 36, and “Rebuilding Java artifacts” on page 46.

To test business logic for transactions:

1. On the Account page, click the Portfolio button.
2. On the Portfolio page, click the Add button.
3. Enter a symbol that matches one of those you entered on the Quotes page in the last task, then click the Transactions button to enter a transaction for this stock.

4. On the Transactions page, click the Add button. Notice that a quantity of 5 is displayed. This entry appears because of a setting for the quantity attribute’s initial value. For more information, see “Defining initial value and constraint for quantity” on page 31.
5. Enter a unique number for ID. Click Save.
   - The Price value should be calculated automatically, because of a parent replicate rule defined on Price. For more information, see “Defining a parent replicate rule for price” on page 31.
   - The Amount value should be calculated automatically, because of a formula defined for amount. For more information, see “Updating the formula rule for transaction amount” on page 43.
   - The Commission value should be calculated automatically, because of a parent replicate rule defined for commission. This rule is dependent on other rules, including a formula for holding’s current value, a sum for account’s portfolio value, and a formula for account’s commission rate. For more information, see “Adding a derived attribute for holding’s current value” on page 40, “Adding a derived attribute for account’s portfolio value” on page 41, and “Adding a rule to determine account’s commission rate” on page 41, and “Adding a rule to determine transaction commission rate” on page 42.

---

Figure 12  Trade Tutorial saved transaction
6. Click the Portfolio button to return to the Portfolio page.
   - The Bought field has been populated, because of a sum rule for the qtyBought attribute. For more information, see “Defining sum rules for quantity bought and sold” on page 34.
   - The On-Hand field has been populated, because of a formula rule for the qtyOnHand attribute. For more information, see “Defining a formula rule for quantity on hand” on page 33.
   - The Current Value field has been populated. For information, see “Adding a derived attribute for holding’s current value” on page 40.
   - The Commissions field has been populated. For information, see “Adding a derived attribute for holding’s total commissions” on page 42.

7. Repeat steps 2-6 as desired, in order to review the execution of business logic as you enter transactions.
8. When you have finished entering transactions, from the Portfolio page, click the Account button to return to the Account page.

- The Commission Rate field value may have changed, if the portfolio value is now greater than $10,000. For more information, see “Adding a rule to determine account’s commission rate” on page 41.

- The Balance field has been populated. This value is calculated through an action rule that calls a Java method. For information, see “Implementing a requirement to update account balances” on page 36.

- The Portfolio Value field has been populated. For information, see “Adding a derived attribute for account’s portfolio value” on page 41.

9. When you are done reviewing the application, click the Logout button and close the web browser window.

Tracing rule execution

Another way to review rule execution is to use the Versata Logic Server Console’s Tracing Monitor while you run the Trade Tutorial user interface.
To trace rule execution for TradeTutorialJSP:

1. In WebSphere Studio Application Developer’s Navigator view, right-click the TradeTutorialJSP node and click Run on Server to start the application again.

2. From the WebSphere Studio Application Developer main menu, click the Versata Logic Server Console button.

3. In the Login dialog, enter sa for Admin Login, leave the other fields blank, and click Login.

4. In the Versata Logic Server Console, under the Monitor node, expand User Sessions, select the System Administrator node (not Console:System Administrator). Enable the Trace user activity check box.
   
   You now can review the tracing output after any or all steps you take in the TutorialJSP application.

5. Return to the TradeTutorialJSP login page. Enter a new account name and click Login.
6. Return to the Versata Logic Server Console and review tracing output.

7. On the Account page, click the Portfolio button.

8. On the Portfolio page, click the Add button, then enter a symbol that you previously entered on the Quotes page. Then click the Transactions button.

9. On the Transactions page, enter a unique ID and a quantity, then click Save. You can review business logic calculations for the TRANSACTION object.

10. Return to the Versata Logic Server Console and review tracing output.

11. You can repeat steps 9 and 10 as desired.
12. When you are done reviewing tracing output, you can return to the Versata Logic Server Console and click the Save Event Log button. The following dialog appears.

![Figure 16  Tracing event log dialog.](image)

13. Click OK to dismiss the dialog. Then close the Versata Logic Server Console window and its DOS window. Also, click the Logout button on the JSP application and close the web browser.