webMethods JI

Tutorial

Version 4.6

September 2026
(originally released December 2004)

webMethods

This document applies to webMethods JI Version 4.6 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1999–2026 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their suppliers. All rights reserved.

The name Software AG, webMethods, and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. Other company and product names mentioned herein may be trademarks of their respective owners.

Document ID: JI-TUT-46-20260923

Table of Contents

About this Guide 5
Tutorial Overview 5
Prior Knowledge Required 5
Assumptions 5
For More Information 6
Organization 6
Formatting Conventions 7
Documentation Set 7
Viewing the Documentation Online 9

Chapter 1. Getting Started 11
About this Chapter. 11
Introducing webMethods JI. 11
The Middleware Model 11
The webMethods JI Runtime Server Environment. 12
webMethods JI Client Libraries and Interfaces 12
Typical Development Process 13
New Terms Introduced in this Chapter 14

Chapter 2. MapMaker Overview 15
About this Chapter. 15
Introduction to MapMaker 15
The Mapping Process 16
Trail Recording 16
Screen Mapping 16
Data Field Mapping 17
Data Modeling 17
Method Definition 18
Service Definition 20
New Terms Introduced in this Chapter 21

Chapter 3. MapPlayer Overview 23
About this Chapter. 23
Introduction to MapPlayer. 23
Using MapPlayer. 23
Starting MapPlayer 24
The MapPlayer Interface 25
MapPlayer Menus 25
MapPlayer Port Identification. 26

MapPlayer Status Windows . . . 26
New Terms Introduced in this Chapter . . . 27

Chapter 4. Basic MapMaker Setup . . . 29
About this Chapter . . . 29
Starting MapMaker . . . 29
About the MapMaker User Interface . . . 30
Tree View . . . 31
Presentation View . . . 32
Properties View . . . 32
Defining the License Key . . . 34
Verifying the Path to the Java Compiler . . . 36
Verifying Resource Server Identification . . . 37
Designating the Tutorial Directories . . . 38
Defining the Host Connection . . . 40
New Terms Introduced in this Chapter . . . 41

Chapter 5. Using MapMaker to Generate Services . . . 43
About this Chapter . . . 43
Creating a webMethods JI Service . . . 43
Starting MapPlayer . . . 45
Starting MapMaker . . . 46
Recording a Trail Through a Connection to the Host . . . 47
Updating the Application Map . . . 53
Identifying to MapMaker that Two Screens are the Same . . . 54
Creating Global Variables . . . 56
Creating Actions During the Trail Recording Process . . . 57
Setting Actions to Use Global Variables . . . 57
Linking Global Variables to Actions . . . 60
Defining a Data Template . . . 61
Adding a Data Template to the Map . . . 62
Adding Data Fields to the Data Template . . . 62
Defining a Table Template . . . 63
Creating a Table Template for the Loan Transactions Table . . . 63
Adding Data Fields to the Table Template . . . 64
Using the Message Field's "More Data" Indicator . . . 65
Setting the Table Template Properties . . . 67
webMethods JI Data Modeling . . . 69
Example Input XML Document . . . 69
Example Output XML Document . . . 69
Data Structure Types . . . 70
Creating Business Entities . . . 71
Defining a Method . . . 80
Adding a Method Step . . . 80
Linking Method Steps . . . 82
Searching for Objects in the Methods Tab . . . 87
Defining Our First Method . . . 88

Mapping the Data Used by the Method 93
Defining the Service 99
Generating the Service and Testing the Client Code 101
Starting Your webMethods JI Server Environment 103
Deploying the Service 104
Confirming the Service Configuration 106
Starting the Configuration Manager 106
About the Configuration Manager User Interface 106
Locating Your Resource Server 107
Testing the Service 112
Editing the Java Client Code (MajorBankSvc_jclient3.java) 113
Compiling the Java Client Code 113
Running the Test Client Code 114
Summary of this Chapter. 116
New Terms Introduced in this Chapter 116

Chapter 6. Advanced MapMaker Techniques 119
About this Chapter. 119
Advanced Table Techniques 119
Record a New Trail 120
Reconcile New Screens 120
Define the Table Template 122
Store Values into Global Variables 123
Define the Table Template Properties 124
Define a Method 125
Define New Business Entities 126
Define the Input and Output Variables of the New Method 129
Create the Method steps 130
Define the Data Mappings for the Method 138
Add the New Method to the Service 139
Generate, Test and Deploy the Service 139

Appendix A. Additional Tutorial Information 141
Tutorial Navigation Tips 141
Customer Accounts for this Tutorial 141
Locating Accounts 142

Glossary 143

Index 1

About this Guide

Tutorial Overview

Welcome to the *webMethods JI Tutorial*. If you are new to webMethods JI, please see the *webMethods JI User's Guide* for further information about the webMethods JI solution for legacy to client/server access.

This tutorial introduces the various features of webMethods JI by walking you through the service development process. The tutorial demonstrates how to use MapMaker, the graphical development environment included with webMethods JI. The tutorial also introduces some basic information about the webMethods JI environment and the webMethods JI clients.

Prior Knowledge Required

This tutorial introduces application design using webMethods JI software, including how to use MapMaker to map legacy screens and generate services, and how to deploy services into webMethods JI's client/server environment.

In order to use webMethods JI, developers should have a working knowledge of the following:

- Windows system administration
- The location and communication requirements of existing legacy data and applications

Optional requirements include:

- Java programming languages for developing webMethods JI clients and for customizing webMethods JI services

Assumptions

This tutorial assumes that webMethods JI has been installed following the procedures in the *webMethods JI Installation and Configuration Guide*. The examples provided in this document are located in *<JI_install_dir>/examples/tutorial*, where *<JI_install_dir>* represents the webMethods JI installation directory.

Note: If the examples are not installed, see your system administrator.

It is also assumed that the system webMethods JI will run on is connected to a network. webMethods JI uses multi-casting to locate server components and, therefore, must be on a machine with an active network connection.

For More Information

If you need more information on any of the topics or procedures described in this tutorial, refer to the *webMethods JI User's Guide*. For detailed information about the API functions, refer to the *webMethods JI JClient3 Javadoc*.

Organization

This tutorial is organized as follows:

- Chapter 1 - "Getting Started" on page 11 Prepares you for using the tutorial by describing the business model upon which the tutorial is based and the directories and files used.
- Chapter 2 - "MapMaker Overview" on page 15 Provides an introduction to MapMaker and the mapping process.
- Chapter 3 - "MapPlayer Overview" on page 23 Provides an introduction to MapPlayer.
- Chapter 4 - "Basic MapMaker Setup" on page 29: Covers starting MapMaker, host selection, and designating the webMethods JI Tutorial directories to be used during the execution of this tutorial.
- Chapter 5 - "Using MapMaker to Generate Services" on page 43 Walks you through developing an interface to a legacy application using MapMaker, the graphical development environment included with webMethods JI.
- Chapter 6 - "Advanced MapMaker Techniques" on page 119 Introduces more advanced features of the MapMaker interface.
- Appendix A - "Additional Tutorial Information" on page 141
- “Glossary” on page 143
- “Index” on page 1

Formatting Conventions

The following formatting conventions are used in this manual:

Table 1. Formatting conventions

Convention	Used for..
Italics	Italics are used for files, directories, programs, and book titles. For example: *<JI_install_dir>/bin/ea_mapmaker.exe*.
`Monotype font`	A monotype font is used to represent examples of code, characters that the user enters, and prompts or messages from the system. For example: Type `ea_start` at the command prompt.
Sans-serif font	A sans-serif font is used to represent Graphical User Interface (GUI) features, such as buttons. For example: Press the Help button to display a list of help topics.
Serif bold font	This font is used for notes and warnings that require special attention. For example: Warning: You must install webMethods JI in an empty directory.

Documentation Set

webMethods JI is supplied with the manuals shown below. The documentation is delivered in Adobe Acrobat Reader Portable Document Format (PDF). No hardcopy documentation is provided, but you can print the PDF files on your local printer.

Use this guide in conjunction with other manuals provided with webMethods JI:

Table 2. webMethods JI documentation set

Title	Description
webMethods JI Release Notes	Provides information about additions and revisions to the current release of webMethods JI. The release notes are distributed in two forms, as a PDF and as a text file
webMethods JI Installation and Configuration Guide	Details installation procedures for webMethods JI.
webMethods JI Tutorial	Provides hands-on instruction about how to write webMethods JI services using MapMaker
webMethods JI User's Guide	Describes how to configure the webMethods JI environment, as well as how to use webMethods JI graphical development and system monitoring tools.
webMethods JI Client Developer's Guide	Describes how to design and develop webMethods JI client applications, as well as how to integrate webMethods JI services into third-party development environments.
webMethods JI Integration Guide	Contains information about integrating webMethods JI Services with other technologies, such as Siebel eBusiness Applications, MQSeries, Web Services, and more.
webMethods JI Supplemental Reference Guide	Contains additional information on webMethods JI commands, error codes, language translation, keyboard mapping, logging, licensing, file formats, and field attributes. This guide replaces the Appendices that were duplicated across the manual set.

Viewing the Documentation Online

Online documentation is available in the following locations:

- In JI GUIs, **Help > Help Topics** opens the OS default PDF viewer. Desktop integration and a registered PDF viewer are required.
- JI documentation is supplied in Portable Document Format (PDF). Open <JI_install_dir>/doc/_JI_Doc.pdf in a PDF viewer or Web browser, where <JI_install_dir> is the location of your JI installation.

You can also access the latest version of the documentation for Software AG products at http://documentation.softwareag.com/. As new versions become available, the documentation on this web site will be updated and the previous versions will be migrated to the Empower Product Support Web site at https://empower.softwareag.com/. If you have a maintenance contract, you can view all versions of documentation on this web site. You will find instructions for registering and obtaining a userid and password on the documentation web site.

Chapter 1. Getting Started

About this Chapter

This chapter introduces webMethods JI, explains the basic concepts, and provides a typical procedure for developing an application with webMethods JI.

In this chapter you will learn about:

- The webMethods JI solution
- The webMethods JI Runtime Server environment
- Supported clients
- Supported platforms
- Client libraries and interfaces
- Typical webMethods JI application development process

You will learn how to:

- Plan and execute the development of an application with webMethods JI

Introducing webMethods JI

webMethods JI is an application development tool and runtime server environment. It provides an industry-leading, enterprise-strength solution for real-time interaction between users in client/server networks and applications running on legacy mainframe computer systems. The webMethods JI application development tool and Application Program Interfaces (APIs) allow customers to create modern client/server interfaces to their data without re-engineering their legacy applications. The webMethods JI runtime server environment allows for large-scale implementation of these clients and services. By using load-balancing and other architectural features, a solution that can be used across large enterprise networks is provided for mainframe access.

The Middleware Model

webMethods JI is a middleware solution. It connects front-end users (“clients”) with back-end, legacy applications running on mainframes or other legacy systems. The webMethods JI runtime server environment sits in the “middle,”

controlling and monitoring connections between the clients and legacy applications. A robust service API allows services to implement modern business logic to facilitate the client to host communications (Figure 1).

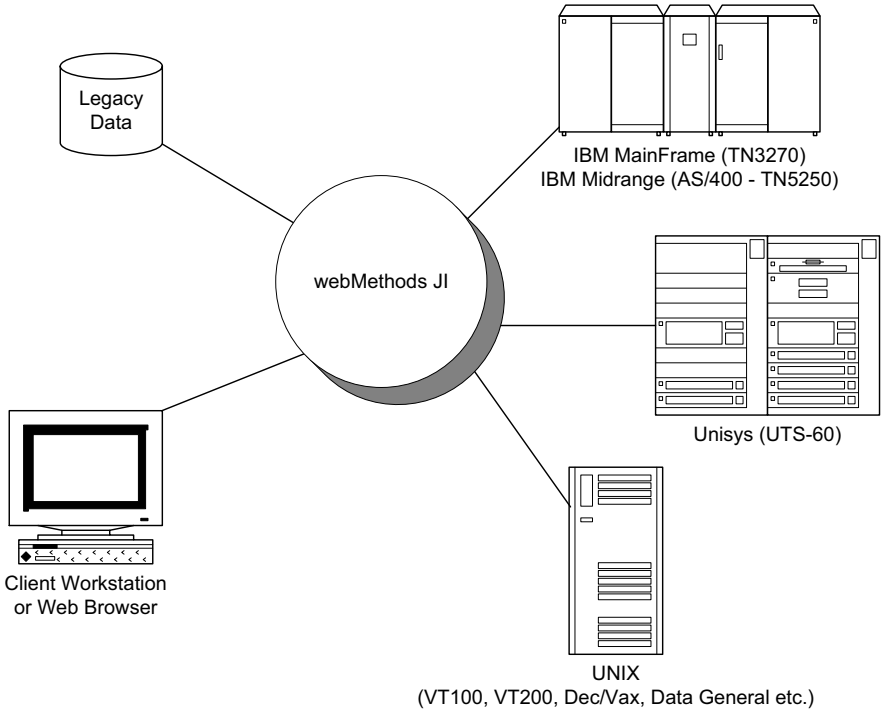

Figure 1. webMethods JI as a Middleware Solution

The webMethods JI Runtime Server Environment

webMethods JI provides a robust runtime environment that includes the components necessary to provide client-to-service connectivity, service-to-legacy application connectivity, and the tools necessary to configure and monitor multiple environments.

> **Note:** Unless otherwise noted, tools in the webMethods JI environment are developed entirely in the Java programming language.

webMethods JI Client Libraries and Interfaces

The following client libraries and interfaces are available with webMethods JI:

- **The Java Client Library Version 3 (JClient3):** JClient3 is a simplified client library written in Java. It is a set of Java packages that provide the functions required for clients developed in the Java programming language to interact

with webMethods JI services. The JClient3 is available on all supported platforms and requires Java SE 8u65 / 8u66 or newer.

- **Gateways**: Several gateways are provided with webMethods JI.
  - **XML Gateway:** A servlet-based connector that enables integration with a client using XML messaging over HTTP.
  - **SOAP Gateway:** A servlet-based connector that enables clients to invoke webMethods JI services as Web Services using SOAP messaging over HTTP.
  - **MQ Gateway:** A stand-alone gateway that allows MQ enabled clients and MQ networks to invoke webMethods JI Services.

Note: For a list of supported platforms and other system requirements, please refer to the webMethods JI Release Notes.

Typical Development Process

This tutorial follows a typical process that is recommended for developing webMethods JI clients and services. In our example, we have a long-term goal of creating a new user interface, to display customer banking data from a legacy system in a variety of modern, client/server formats.

The main stages of this process are:

1. **Define the business model:** Determine the client-side requirements for transactions and data presentation. For example, you may want the client to input an account number or a customer name to receive information about the customer's account.
2. **Establish a knowledge base of legacy system behavior:** Learn how the legacy system operates by using it to execute all transactions required by the client, from starting up to logging off.
3. **Determine connectivity requirements:** Determine data communications and connection paths needed to get to the legacy system.
4. **Design a legacy model:** Base the model on the legacy-application knowledge base, client transactions and data presentation requirements.
5. **Capture screens and fields:** Capture all screens and fields needed to connect to and use the legacy application.
6. **Create Method definition:** Record all screen transitions, inputs, actions and outputs.
7. **Develop and test the application:** Check the required code and refine as necessary.

New Terms Introduced in this Chapter

webMethods JI: An application development tool and runtime server environment, which provides a solution for real-time interaction between users in client/server networks and applications running on legacy mainframe computer systems. The webMethods JI application development tool and Application Program Interfaces (APIs) allow customers to create modern client/server interfaces to their data without re-engineering their legacy applications.

Middleware: webMethods JI is a middleware solution. It connects front-end users ("clients") with back-end, legacy applications running on mainframes or other legacy systems.

webMethods JI runtime server environment: A robust runtime environment that includes the components necessary to provide client to service connectivity, service to legacy application connectivity, and the tools necessary to configure and monitor multiple environments. The webMethods JI runtime server environment sits in the "middle," controlling and monitoring connections between the clients and the legacy applications.

Host: A computer machine where applications reside. In webMethods JI, hosts can be the machine on which components of the webMethods JI environment are running, the machine on which the telnet, TN3270, or TN5250 server reside, or the machine on which the legacy applications reside.

Chapter 2. MapMaker Overview

About this Chapter

This chapter introduces the MapMaker application and explains the basic concepts and procedures for working with MapMaker.

In this chapter you will learn about:

- The MapMaker application and its uses
- MapMaker maps
- The six phases of the mapping process
- Trails and trail recording
- Screens and screen mapping
- Data fields and data field mapping
- Methods, Method definition, and the various Method steps
- Services and service definition

You will learn how to:

- Understand MapMaker

Introduction to MapMaker

MapMaker is a Java-based application included with webMethods JI that allows users to create interfaces to legacy applications. These interfaces are output from MapMaker as webMethods JI Java services, then deployed into the webMethods JI server environment.

MapMaker is designed to simplify the mapping of legacy-application screens into Java application logic. This is accomplished by creating a “map” of the host application. The map consists of the application screens, the navigational information required to traverse through the sequence of screens, and the tags and fields included on each legacy screen. This information is stored in a map file and in Java class files that are produced from within MapMaker.

For more information about MapMaker and the webMethods JI server environment, see the *webMethods JI User’s Guide*.

The Mapping Process

MapMaker is used to create maps as interfaces to legacy applications. Maps are used to automatically generate webMethods JI Java services. The MapMaker maps consist of Java source files that contain all the logic required to navigate and interact with legacy applications.

The mapping process consists of six phases:

- Trail recording
- Screen mapping
- Data field mapping
- Data modeling
- Method definition
- Service definition

Trail Recording

A trail is recorded by simply navigating through the host application. During the trail recording phase, you interact with legacy screens using the terminal emulation capabilities of MapMaker exactly as if interacting with the legacy application during normal use. This can be accomplished with a single trail or multiple trails using different paths through the legacy application. You may add new trails to a map at any time. MapMaker will combine these into a single map.

During trail recording, a snapshot is created for each screen encountered. As the trail is recorded, user inputs (actions) are mapped to their associated screen snapshots. This provides information necessary for the webMethods JI service to navigate from one screen to the next within the legacy application.

Screen Mapping

The screen mapping phase begins after the trails have been recorded. During this phase, the trails are analyzed by MapMaker to determine which snapshots have the same formatted field layout. Snapshots with the same formatted field layout are grouped together as one “screen” in the map. This grouping is based entirely on the layout of the screen’s fields and is independent of the actual text or content of the fields themselves. While screen content varies depending on the user input and the data output from the legacy application, screens are still identified based on their formatted field layout. This means the webMethods JI system is always able to recognize a screen, regardless of the contents of the field.

For example, the user inputs an account number and gets customer data from the Customer Information screen. Each time a different account number is input, the required customer data is displayed in the same Customer Information screen. The same set of formatted fields are used, but the actual data in these fields is different.

MapMaker sometimes identifies snapshots as identical or unique when they are not. This can happen when a screen does not have any formatted fields or when the entire screen consists of a single field. It can also happen if the same screen has fields whose positions vary depending on the length of data displayed in previous fields. If this happens, special screen-recognition options can be used to enable MapMaker to distinguish between two screen snapshots or to group different screen snapshots together. This is accomplished using tags and/or disabling formatted fields to prevent MapMaker from using them for screen recognition.

During screen mapping, global variables can be created for input and output actions, while global actions can be created for error-handling situations.

Data Field Mapping

After the screens are correctly recognized, you tell MapMaker which fields contain data to be extracted by the webMethods JI service. This is accomplished using Data Templates, Table Templates and the Data Fields they contain.

Data Templates are used as containers that map individual Data Fields on the legacy screen. Table Templates are containers that map repeating fields on the legacy screen. Legacy screen data is often displayed in a table format with a varying number of repeating rows that continue for several pages. Table Templates are used to define boundaries of repeating information. The fields in the first row of the table are mapped as Data Fields and repeat through the entire area defined by the Table Template.

Data Modeling

webMethods JI enhanced data modeling provides the following capabilities:

- Supporting a variety of conventional data types, rather than only a String data type. Such data types are **String**, **Number**, **Boolean**, **Date**, **Binary**, etc.
- Defining new data types. These may be structures that consist of other data types, either conventional or newly-defined. Moreover, a data type can be defined as an array of other data type objects. Data types are defined in MapMaker's **Business Entity Editor**.
- Defining variables of various types, including newly defined data types. Therefore, a variable may also be a structure consisting of various objects. A variable of a certain data type allocates memory for the data that is directed through that specific data type. Global variables are defined in MapMaker's

Business Entity Editor and Method variables are defined in MapMaker's **Edit Method Variables** dialog box. Both types of variables can be defined in the Data Mapping Editor's **Edit Variables** dialog box.

- Exporting and importing data structures to and from external sources such as XSD files. These operations are performed using MapMaker's **Business Entity Editor**.
- Defining the relationship between one data type and another. This is accomplished using MapMaker's **Structure Relationship Editor**.

  The Structure Relationship Editor supports the use of XPath Syntax. Using XPath, you can further manipulate the relationship between two corresponding objects.
- Mapping data from one data type to another. By mapping data, you are directing the flow of data within the chronological flow of a Method. Even though the relationship between different data types is already defined, it is necessary to "tell" your Method to take the data that arrives through one data type and, using Global variables or Method variables, transport it to another data type. Data Mapping is defined using MapMaker's **Data Mapping Editor**.

For detailed information on data modeling, *see Chapter 7: Data Modeling* in the *webMethods JI User's Guide*.

Method Definition

Methods define how a transaction (e.g. "open account", "get account", "balance", etc.) is accomplished against the host application. Multiple Methods can be created for a single map. Method definitions consist of the following elements:

- **Input variables** that allow clients to pass data into the Method.
- **Method steps** that perform specific functions when the Method is invoked.
- **Output variables** that allow the Method to pass data to the client.

The following Method steps can be created:

Traverse: Instructs the method to traverse from the current screen to another screen.

Fetch: Extracts data from the screen, based on the specified data template or table template.

Perform: Instructs the method to perform a specified action.

Write: Inputs data to the screen, based on the specified data template or table template.

UserInteraction: Allows users to navigate the host application manually. A read/write-enabled PATerm or JTerm window must be open on the client's display for this step to successfully function at run time.

Internal Invoke: Calls another method residing in the same map. The two methods can use the same variables and no network operations are required.

External Invoke: Calls another method residing in a different map. The method being invoked shares the same host connection as the calling method.This step provides multi-developer support, allowing different developers to work on different parts of the same legacy system. An External Invoke step requires passing input and output parameters to the external method, using Data Mapping. No network operations are required.

JClient3: Calls another method from other webMethods JI Services, using JClient3 API calls. This step allows you to access multiple, parallel legacy applications. Since the invoked methods belong to a different map, data mapping operations are required. In addition, since the invoked method is located on a different virtual machine, network operations are required as well.

IfCondition: Provides a comparison between two objects.

- If the evaluation of the Condition is "true", the step(s) on the conditional or "**branch**" link are executed.
- If the evaluation of the Condition is "false", the primary or "**next**" method logic path continues.

If the true/conditional step(s) are executed, the primary or "**next**" method logic path resumes once they have completed.

IfElseCondition: Provides a comparison between two objects.

- If the evaluation of the Condition is "true", the step(s) on the conditional or "**branch**" link are executed.
- If the evaluation of the Condition is "false", i.e. the "Else" part of the condition, the primary or "**next**" method logic path continues.

If the true/conditional step(s) are executed, the primary or "**next**" method logic path is not followed.

ForCondition: Executes the step(s) on the conditional or "**branch**" link, in a loop that is based on specified Initializer, Conditional, and Increment settings. When the **For** loop ends, the primary or "**next**" method logic path continues.

WhileCondition: Executes the step(s) on the conditional or "**branch**" link based on the specified Conditional settings. When the While loop ends, the primary or "**next**" method logic path continues.

DoWhileCondition: Executes the step(s) on the conditional or "**branch**" link, based on the specified Conditional settings. When the DoWhile loop ends, the primary or "**next**" method logic path continues.

EndIf: Indicates the completion of an IfCondition **branch** path. Once the EndIf step has been reached, the primary, **next** method path is resumed.

Continue: Indicates the completion of a ForCondition, a WhileCondition or a DoWhileCondition loop. The Continue step causes the next iteration of the loop to begin. If the loop condition is satisfied, the primary, **next** method path is resumed.

Break: Breaks the ForCondition, WhileCondition or DoWhileCondition loop, goes back to the appropriate step (ForCondition, WhileCondition or DoWhileCondition) and then resumes the primary, **next** method path.

Set: Provides a mechanism for setting a variable and/or adding data to the return map.

Thread: Provides for parallel code execution within the method. When a Thread step is added to the Method, a new branch is created representing a thread of parallel code execution. Each Thread step represents one additional thread of execution.

ThreadJoin: Adds a ThreadJoin step to the selected Method. A ThreadJoin step provides a method to 'collect' the threads within the method, and a point of convergence for the parallel code.

EndThread: Indicates the completion of a Thread branch. The completed threads within the method are "collected" using a ThreadJoin step.

Service Definition

After creating Methods, you can create services that represent logical groupings of the Methods. Multiple services can be created from one map.

New Terms Introduced in this Chapter

MapMaker: A Java-based application included with webMethods JI that allows users to create interfaces to legacy applications. These interfaces are output from MapMaker as webMethods JI Java services, then deployed into the webMethods JI server environment. MapMaker is used to record trails, maps, data and table templates, create Methods and services, and then generate and optionally deploy the services into the webMethods JI server environment.

Map: The logical representation of the screens, fields, data input and AID keys that make up the user interaction with a legacy application. Maps are created in MapMaker.

Screen: The screen, as viewed (rendered) from a legacy system.

Screen Mapping: The process of identifying screens within the host application based on field layouts. Such field layouts include the following elements: tags, fields, areas, repeating areas and repeating fields.

Trail: The linear path of all screens encountered during navigation through a host application. MapMaker records trails during host application interaction.

Trail Recording: The process of registering the screens encountered during navigation through a host application. Trails are recorded by establishing a host connection, navigating through host screens, making snapshots of each screen navigated through, and recording every action performed by the end-user.

Data Field: An individual field on the legacy screen that is added to either a data template or a table template. Data fields are used in conjunction with output variables to extract data from the legacy screen.

Data Modeling: The process of defining and editing existing data types, defining relationships between different data types, and mapping data from one data structure to another in the flow of a Method.

Method: An object-oriented entity of one or more functions. A Method consists of input variables, Method steps and output variables. A collection of Methods, initialization code, and events make up a service.

Service: A collection of Methods that answer requests from a client.

Chapter 3. MapPlayer Overview

About this Chapter

This chapter introduces the MapPlayer application, and takes you through your first steps in MapPlayer.

In this chapter you will learn about:

- The MapPlayer application
- Using MapPlayer
- MapPlayer map files
- The MapPlayer interface

You will learn how to:

- Start MapPlayer

Introduction to MapPlayer

MapPlayer is the JI utility that allows you to play back maps that have already been generated. This tutorial uses MapPlayer as a host simulator.

In this tutorial, the map file *MajorBankHost.map* will be used by MapPlayer as the legacy application simulation. This map file can be found in *<JI_install_dir>/examples/maps*, where *<JI_install_dir>* represents the webMethods JI installation directory.

For more information about MapPlayer, see the *webMethods JI User's Guide*.

Using MapPlayer

MapPlayer is a tool included with webMethods JI that allows you to play back the maps that have been generated from within MapMaker. When playing maps, MapPlayer serves as a host simulator and can be accessed by a terminal emulator (such as MapMaker), allowing the terminal emulator to interact with the map as it is played back.

Starting MapPlayer

To start MapPlayer from the command line, execute the following command at the command prompt:

```
cd <JI_install_dir>\bin
ea_mapplayer
```

> **Note:** In Windows, if shortcuts were created during the installation, MapPlayer can be started by selecting Start>Programs>webMethods JI 4.5>MapPlayer.

The MapPlayer user interface opens (Figure 2).

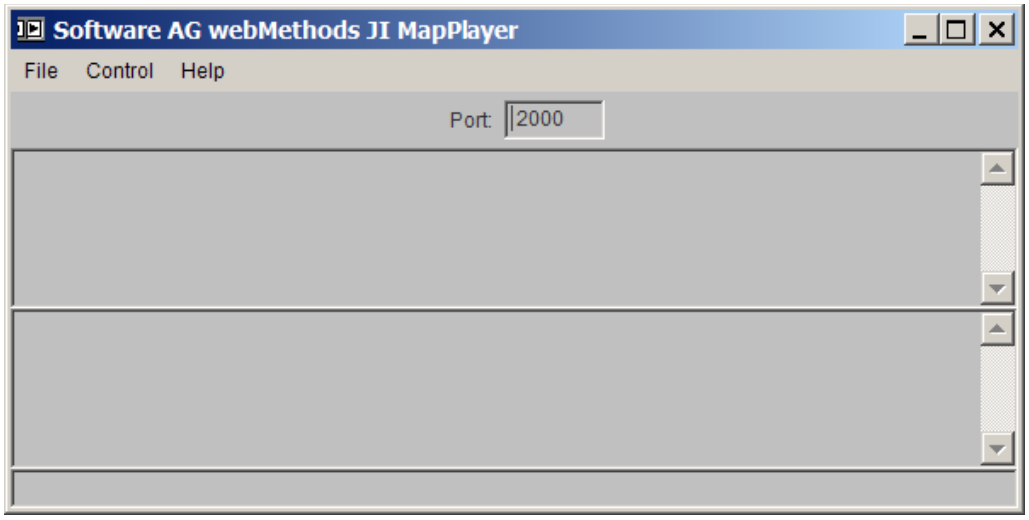

Figure 2. The MapPlayer Interface

The MapPlayer interface contains three menus, a **Port** text box and two status panes.

The MapPlayer Interface

MapPlayer Menus

File Menu

Figure 3. File Menu

This menu has two options:

- **Open:** Used to open map files.
- **Exit:** Used to exit MapPlayer.

Control Menu

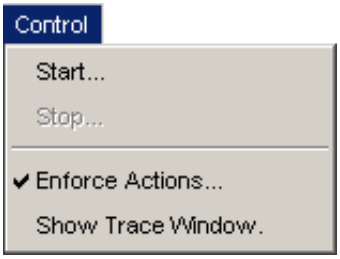

Figure 4. Control Menu

This menu has four options:

- **Start:** Used to start an open map file.
- **Stop:** Used to stop an open map file.
- **Enforce Actions:** Used to determine if actions should be enforced. This means that during runtime, whenever the end-user presses an AID key, MapPlayer checks to see if the screen requires any additional data before moving to the next screen. If such required data is missing, MapPlayer ignores the AID key.
- **Show Trace Window:** Used to show new connections. When enabled, this causes a window to open each time a new connection is made while MapPlayer is running. This window traces the data stream.

Help Menu

Figure 5. Help Menu

This menu consists of a single option that provides information about MapPlayer:

- **About:** Displays the following window (Figure 6):

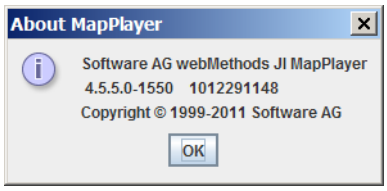

Figure 6. MapPlayer About Dialog Box

MapPlayer Port Identification

The **Port** text box

identifies the port that MapPlayer uses to listen for connections.
The default is 2000.

MapPlayer Status Windows

There are two status windows in MapPlayer:

- **The top status window** (Figure 7): Identifies the connections to MapPlayer and specifies the host name or IP address and the terminal emulator port number.

Figure 7. Top MapPlayer Status Window

- **The bottom status window** (Figure 8): Represents any activity that MapPlayer has executed. The date, time, and type of activity is logged to this window.

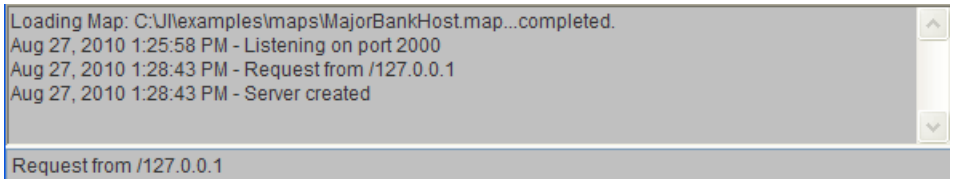

Figure 8. Bottom MapPlayer Status Window

New Terms Introduced in this Chapter

MapPlayer: A tool included with webMethods JI, which allows you to play back the maps that have been generated from within MapMaker. When playing maps, MapPlayer serves as a host simulator and can be accessed by a terminal emulator (such as MapMaker), allowing the terminal emulator to interact with the map as it is played back.

MapPlayer Map File: A file, generated by MapMaker, which is used by MapPlayer to simulate a host connection. The webMethods JI development kit is provided with the *MajorBankHost.map* file.

Data Stream: The flow or stream of information between computer programs. Data on the data stream is represented using "character encoding" and is transferred using a mutually agreed upon protocol.

Chapter 4. Basic MapMaker Setup

About this Chapter

This chapter takes you through your first steps in MapMaker.

In this chapter you will learn about:

- The MapMaker user interface, its views and panels
- Host connection definitions
- Java definitions
- Server definitions
- Default directory definitions

You will learn how to:

- Start MapMaker
- Define the host connection
- Verify the path to the Java compiler
- Verify resource server identification
- Designate the Tutorial Directories

Starting MapMaker

Start the MapMaker graphical development environment as follows:

1 Enter the following in a DOS prompt:

```
cd <JI_install_dir>\bin
ea_mapmaker
```

> **Note:** In Windows, if shortcuts were created during installation, MapMaker can be started by selecting Start>Programs>webMethods JI 4.5>MapMaker.

The **webMethods JI MapMaker** user interface opens (Figure 9).

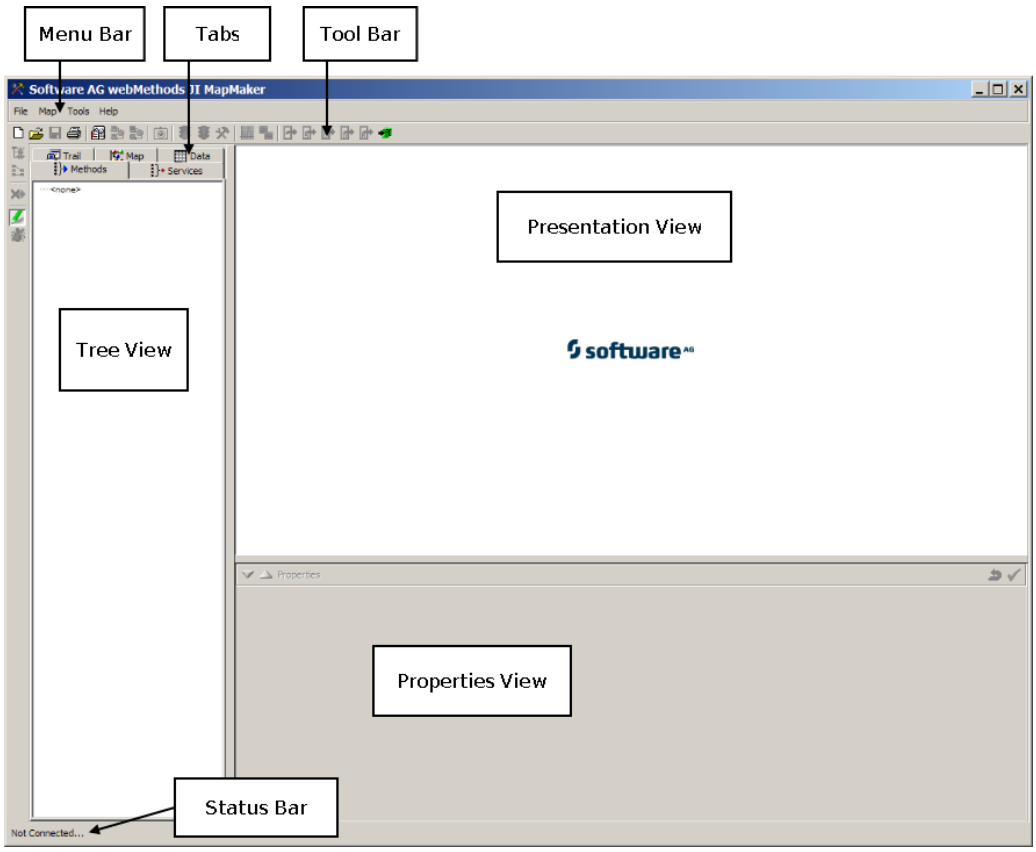

Figure 9. The MapMaker User Interface

About the MapMaker User Interface

The MapMaker interface consists of three primary panes, or views:

- The "Tree View" (on page 31)
- The "Presentation View" (on page 32)
- The "Properties View" (on page 32)

Note: The “look and feel” of the MapMaker graphical interface can be customized. The available “look and feel” options are Metal, CDE/Motif, and Windows. This property is determined in the **Appearance** tab of the MapMaker **Properties** dialog box (displayed by selecting **File > Properties** from the menu). The screen snapshots and descriptions in this tutorial use the Windows “look and feel.”

Tree View

The left pane of the MapMaker interface is the Tree view. The Tree view is a representation of hierarchical information consisting of a "root" component (for example, the Trail tree in the **Trail** tab), which can be expanded to expose underlying sub-components (also called "nodes" or "branches"). Typically, these sub-components can be expanded to reveal their own sub-components, creating the hierarchical tree format. A "**+**/**-**" symbol before each component in the tree identifies if the node can be expanded to reveal additional nodes. A "**+**" indicates that the node can be expanded and a "**-**" indicates that the node is already expanded. The absence of either a "**+**" or "**-**" indicates that the node cannot be expanded.

> **Note:** When using the Motif "look and feel," a root node handle is used in place of the "+/-" symbols.

The Tree view is further delineated by "tabs" that run across the top of the Tree view. Each tab represents a "tabbed panel" containing a hierarchical tree of its own. The MapMaker interface includes five tabbed panels:

- **Trail Panel:** Graphically displays the trail that is recorded during navigation through the legacy application.
- **Map Panel:** Graphically displays the map of the legacy application, generated after the trail has been recorded.
- **Data Panel:** Used to add Data Templates and Table Templates to the screens included in the map.
- **Methods Panel:** Used to define the Methods that are included with the service for this map.
- **Services Panel:** Used to define the services to be generated from MapMaker for this map.

Presentation View

The Presentation view is the upper-right pane of the MapMaker interface. This view allows you to process host screens in different ways, depending on the tab that is selected in the Tree view:

Tree Tab	Presentation View Functionality
Trail	When you are connected to the legacy host, this view displays an emulation of the current host session. This emulation is the same screen you would see on a terminal connected directly to the host or in a terminal emulator. During trail recording, the screens from the legacy host are saved as "snapshots" and are included in the trail. After trail recording, the saved screens are displayed in the Presentation View. You can view a saved screen by clicking on the screen component in the Tree View (**Trail**, **Map**, or **Data** panels only).
Map	Defines tags in the recorded screens.
Data	Defines data templates and table templates in the recorded screens.
Methods	Illustrates the flow of the method that is selected in the Tree view.
Services	Illustrates the map's JI Services and the Methods attached to each Service in the Tree view. The Presentation view remains empty.

Note: The Presentation view correctly interprets and handles all field attributes, including attributes (such as blinking) that are not displayed.

Properties View

The bottom right pane of the MapMaker interface is the **Properties** view. This view allows you to view and edit the properties of the object selected in the Tree view or in the Presentation view. Figure 10 shows an example **Properties** view, displaying the settings of the **MajorBank** host that is selected in the Tree view.

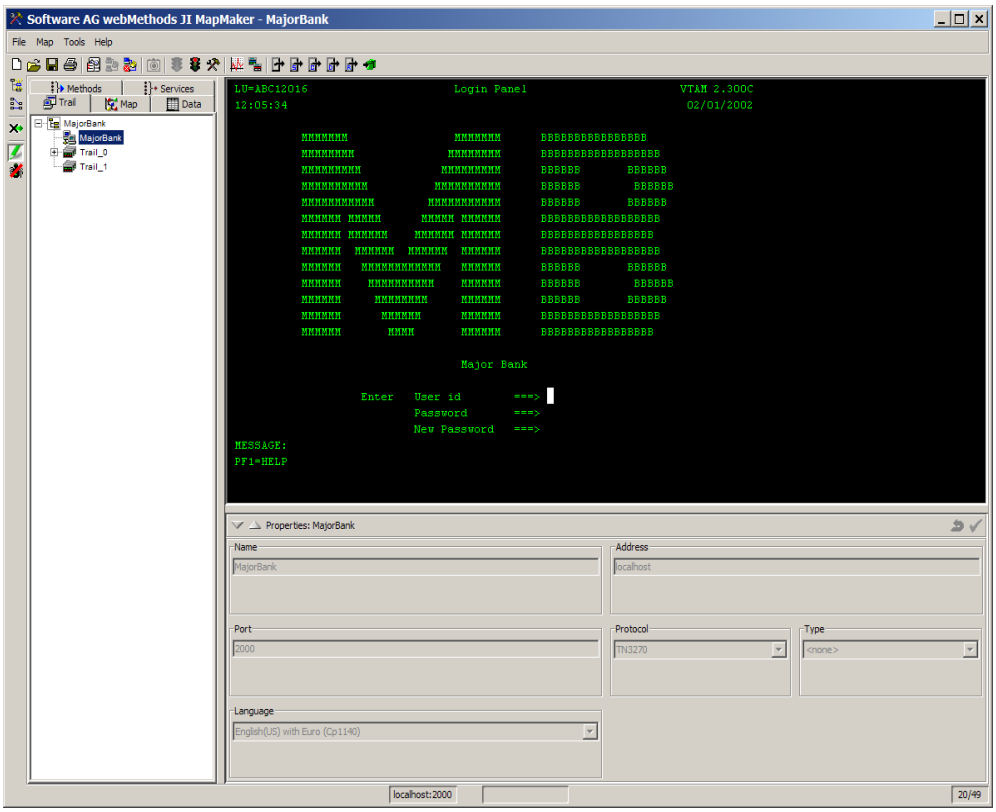

Figure 10. Properties View

The selected object's name is indicated in the **Properties** view's caption (e.g. **Properties: MajorBank**).

Collapsing and Expanding the Properties View

The **Properties** view can be minimized and expanded using the arrow buttons located at its upper left corner. Once these buttons are clicked, a corresponding floating arrow is displayed:

- The Contract button is followed by a floating Expand button .
- The Expand button is followed by a floating Contract button .

The floating arrow indicates the view's pervious position, allowing you to conveniently restore it.

Cancelling and Applying Property Updates

When there are unsaved changes to the object's properties, the **Revert** and **Apply** buttons (located at the upper right corner of the **Properties** view) are enabled.

- Click the **Revert** button to cancel all recent updates made across all tabs, and restore the properties that were last saved.

- Click the **Apply** button to enforce all the updates made across all tabs since the last save.

> **Note:** Selecting another component in the Tree view or in the Properties view performs an automatic apply.

Properties View Tabs

The **Properties** view consists of tabs, which vary as a function of the object selected in the Tree view. All objects share the following tabs:

- **General:** Defines the object's basic settings, such as its custom class.
- **Comments**: Used to annotate the object. The comments are included in the object's generated code.

Defining the License Key

Import the XML license in Configuration Manager, then configure MapMaker:

1. From the **File** menu select **Properties**. The **Properties** dialog box is displayed.
2. Select the **Licensing** tab. This tab displays the MapMaker license and allows you to update it as needed (Figure 11).

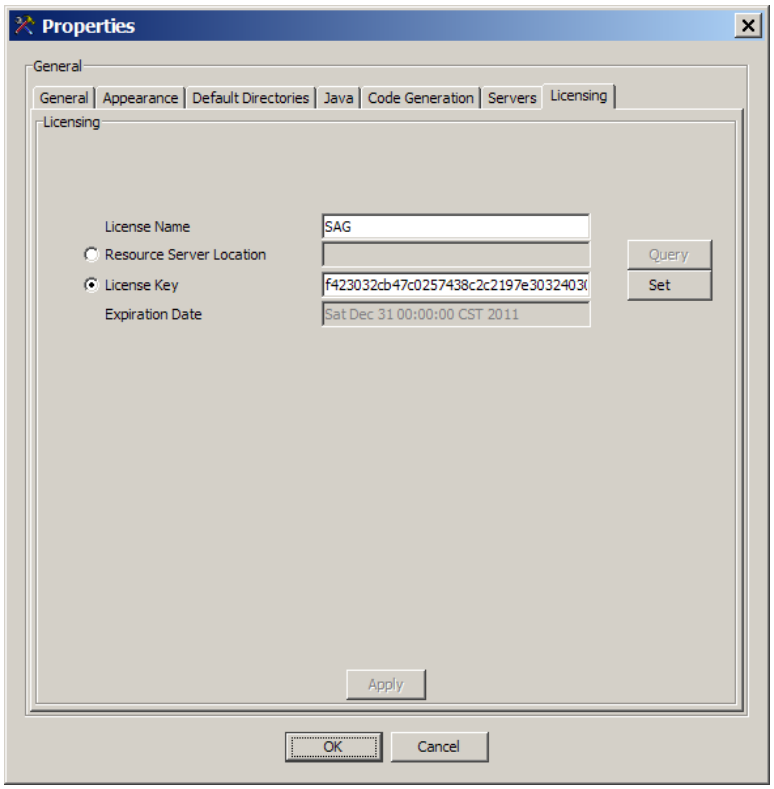

Figure 11. Properties Dialog Box > Licensing Tab

Option	Description
License Name	Imported name from Configuration Manager.
Resource Server Location	This option is used to automatically locate and obtain a MapMaker license. Enter the host and port (e.g. `localhost:30002`) and click **Query**. webMethods JI automatically completes the **License Key** and **Expiration Date**.
License Key	Manual entry of an existing legacy MapMaker key. Enter the license key and click **Set**. MapMaker checks the validity of the key and completes the **Expiration Date**.

Option	Description
Expiration Date	The date and time on which the license expires. This date is automatically calculated based on either the **Resource Server Location** or the **License Key**.

3 Enter the imported license name in the **License Name** field.
4 Use one of these methods (manual entry is for existing legacy keys):
   - For an existing legacy license, enter its key in **License Key** and press **Set**.
   - For an imported XML license, enter the Resource Server host and port in **Resource Server Location** and press **Query**.
5 Press the **Apply** button to apply the new license key.

Verifying the Path to the Java Compiler

In order to successfully generate code, a java compiler (*javac*) must be identified in the **Java** tab of the MapMaker **Properties** dialog box.

To identify a java compiler, follow these steps:

1 Select **File > Properties**. This opens the MapMaker **Properties** dialog box.
2 Click on the **Java** tab to display the Java panel of the dialog box (Figure 12).

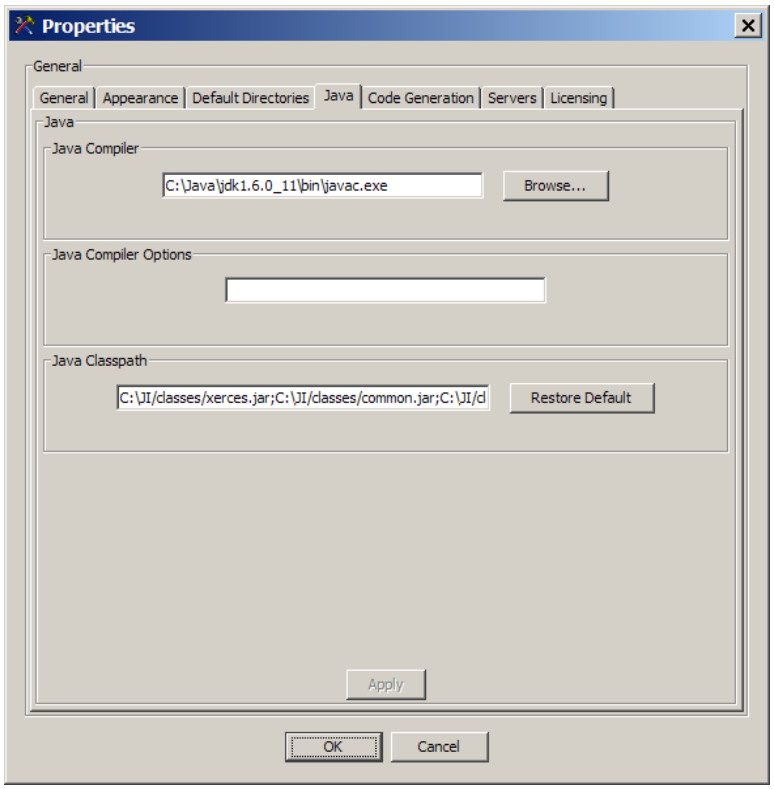

Figure 12. The Java Compiler Path in the MapMaker Properties Dialog Box

3 Enter the location of the **Java Compiler** (`javac`) to be used by MapMaker to generate services.

> **Note:** MapMaker code generation requires Java Development Kit (JDK) 17, including Azul Zulu. Select the Java 17 javac compiler.

Verifying Resource Server Identification

In order to successfully deploy a service into your resource database, your Resource Server must be identified (or the **Enable Multicasting** property must be selected) in the **Servers** tab of the MapMaker **Properties** dialog box.

To verify that the settings in this dialog box are correct, proceed as follows:

1 Select **File > Properties**. This opens the MapMaker **Properties** dialog box.

2 Click on the **Servers** tab to display the **Resource Server** panel of the dialog box (Figure 13).

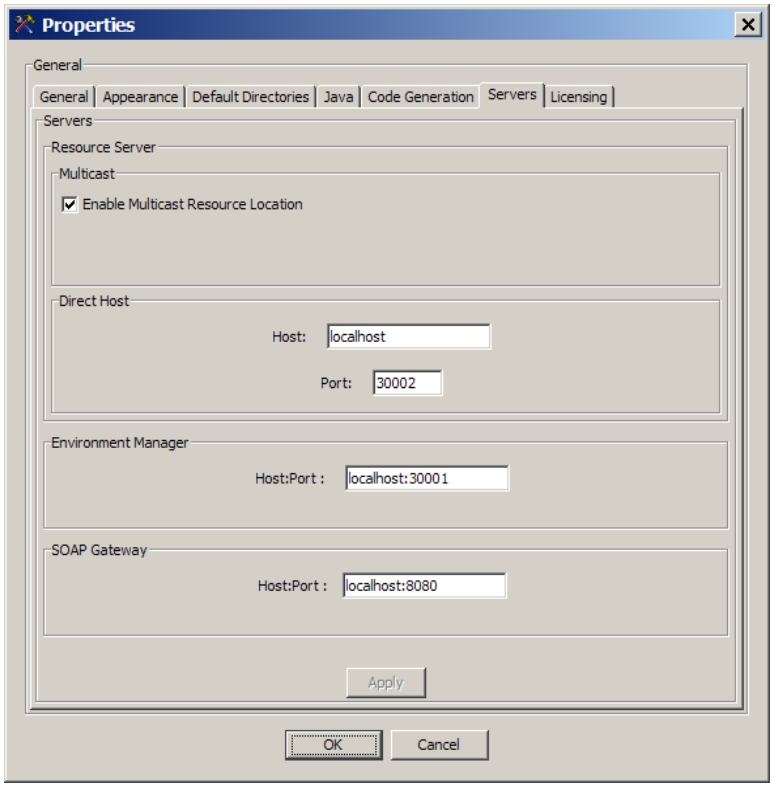

Figure 13. MapMaker Properties > Servers Tab

3 If you will be using a remote system for the webMethods JI runtime, change the **Direct Host** and **Environment Manager Host:Port** definitions. For example, if you will be running the webMethods JI runtime on a server named `ABC.XYZ`, then replace `localhost` with `ABC.XYZ` under **Direct Host** and **Environment Manager**.

   If you have changed the default port numbers for the runtime, you will also need to change the **Port** numbers in this tab.

4 If you will only be using the Resource Server as defined under **Direct Host** for deploying services, you can disable Multicast Resource Location by clearing the **Enable Multicast Resource Location** check box.

Designating the Tutorial Directories

For the purpose of these tutorial exercises (as well as ease of use), it is recommended that you designate and actually create specific directories to contain the generated code, jar files and services that will be generated during the execution of the tutorial procedures. These directories are specified in the MapMaker **Properties** dialog box using the **Default Directories** tab.

To define the default directories:

1 Select **File > Properties**. This opens the **Properties** dialog box.

2 Select the **Default Directories** tab (Figure 14).

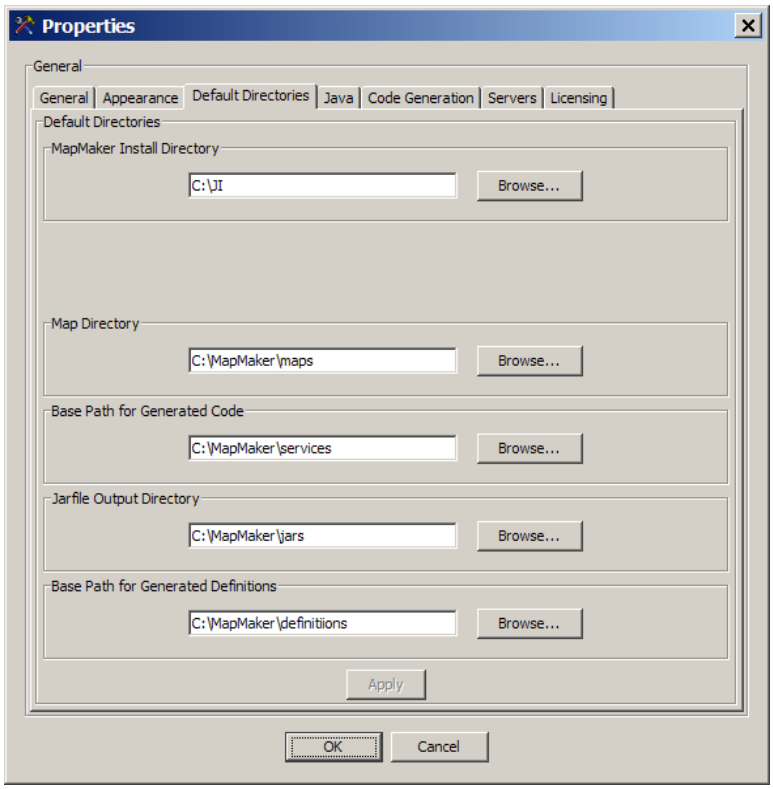

Figure 14. MapMaker Properties > Default Directories

3 In the **Default Directories** tab, set the directories to the following values:

Default Directories Selection	Change the Directory Path/Name to...
Map Directory	C:\MapMaker\maps
Base Path for Generated Code	C:\MapMaker\services
Jarfile Output Directory	C:\MapMaker\jars
Base Path for Generated Definitions	C:\MapMaker\definitions

4 Click **Apply** and then click **OK** to close the **Properties** dialog box.

Defining the Host Connection

To define the connection from MapMaker to MapPlayer, follow these instructions:

1 In MapMaker, select **File** > **Hosts**. The **Hosts** dialog box opens (Figure 15).

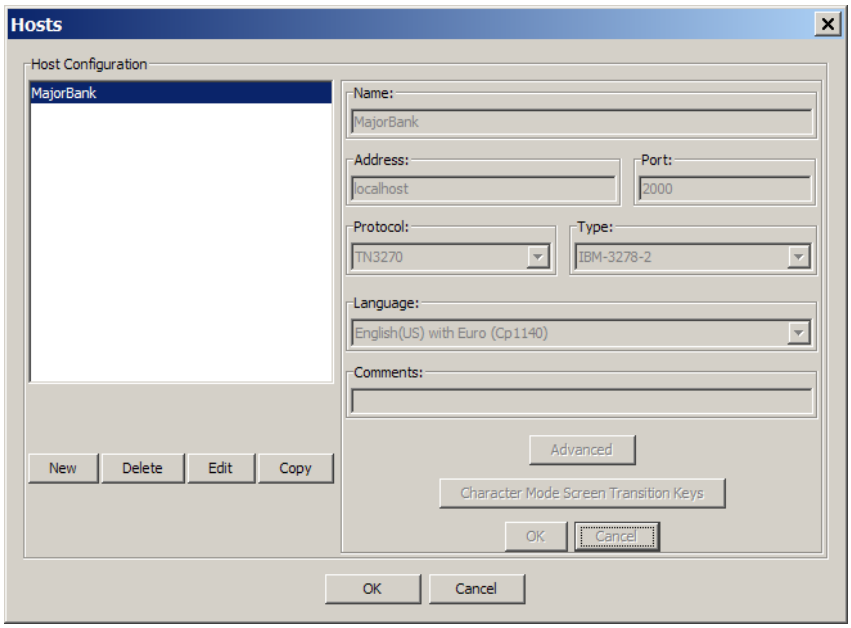

Figure 15. MapMaker Hosts Dialog Box

2 Click the **New** button and enter the following information:

Option	Description
Name	Enter `MajorBank`.
Address	Enter `localhost`. **Note:** If MapPlayer is not running on your local machine, you should enter the name or IP address of the host on which MapPlayer is running.
Port	Enter the port number on which MapPlayer is running (the default port setting for MapPlayer is 2000).

The remaining options in the dialog box can be left at their default settings or can be left blank.

3 Click the bottom **OK** button to close the **Hosts** dialog box.

New Terms Introduced in this Chapter

Trail Panel: Graphically displays the trail that is recorded during navigation of the legacy application.

Map Panel: Graphically displays the map of the legacy application, generated after the trail is recorded.

Data Panel: Used to add Data Templates and Table Templates to the screens included in the map.

Methods Panel: Used to define the Methods that are included with the service for this map.

Services Panel: Used to define the services that will be generated from MapMaker for this map.

Host Connection: A connection, used by MapMaker, to the Host application or to MapPlayer. MapMaker uses a host connection to record trails. Furthermore, the host connection settings are used for generated code, to determine how the service will connect to the host during runtime.

Resources: The components of webMethods JI that are managed by the Resource Server. These components include Resource Databases and license files.

Resource Database: A database that is used in the webMethods JI environment to store environment and service configuration, along with service code and maps.

Resource Server: Manages the communication between environment managers and the webMethods JI resources.

Multicasting: A connectionless IP networking communication in which applications on the IP network broadcast information over a well-known socket.

Chapter 5. Using MapMaker to Generate Services

About this Chapter

This chapter walks you through all the stages of creating a webMethods JI service in MapMaker. In this chapter you will learn about:

- webMethods JI services.
- Data Modeling.
- The basic techniques for creating Methods (transactions) and services.

You will learn how to:

- Connect to the Host and record a trail.
- Update the application map.
- Create global variables.
- Link global variables to actions.
- Define a data template.
- Define a table template.
- Create Data Structures.
- Define a Method.
- Define a service.
- Generate the service and test the client code.
- Start your webMethods JI server environment.
- Deploy a service.
- Confirm the service configuration.
- Test the service.

Creating a webMethods JI Service

The following steps detail the sequence followed to create a webMethods JI service, which is the interface to your legacy application:

1. **"Starting MapPlayer"** - Describes how to start MapPlayer and how to open the *MajorBankHost.map* file that serves as the legacy simulation for this tutorial.
2. **"Starting MapMaker"** - Describes the steps required to launch MapMaker.

3 **"Recording a Trail Through a Connection to the Host"** - Describes how to create a host connection definition, which is used by MapMaker to connect to the host that provides access to the legacy application.
4 **Connecting to the Host** - Describes how to create a host connection definition, which is used by MapMaker to connect to the host that provides access to the legacy application.
5 **"Updating the Application Map"** - Describes how to update maps, create the screens and tie actions to screen snapshots.
6 **"Creating Global Variables"** - Details the steps required to create global variables for the login name, password and account numbers in the tutorial application.
7 **"Linking Global Variables to Actions"** - Describes how to link the new global variables to the actions that input the login name, password and account number into the legacy screen.
8 **"Defining a Data Template"** - Describes how to map fields on the legacy screen. These fields can be used to extract data from the screen.
9 **"Defining a Table Template"** - Describes how to map repeating data that will be extracted by the service.
10 **"webMethods JI Data Modeling"** - Describes how to create the data structures (i.e. Business Entities) to be used for input and output.
11 **"Defining a Method"** - Describes how to define a webMethods JI service Method.
12 **"Defining the Service"** - Describes how to include Methods in a webMethods JI service.
13 **"Generating the Service and Testing the Client Code"** - Provides instructions for generating the service and test clients.
14 **"Starting Your webMethods JI Server Environment"** - Provides instructions for starting your webMethods JI runtime server environment. The runtime server environment consists of one or more Resource Databases, Resource Servers and Environment Managers.
15 **"Deploying the Service"** - Describes how to deploy the generated service code into the Resource Database.
16 **"Confirming the Service Configuration"** - Describes how to confirm the configuration of the service for your runtime server environment. The webMethods JI Configuration Manager is the graphical interface used to configure the service.
17 **"Testing the Service"** - Describes how to test the service using the generated Java test client.

Starting MapPlayer

The MapPlayer host simulation must be started before connecting to it from MapMaker.

1 Start the MapPlayer application using the following command:

```
cd <JI_install_dir>\bin
ea_mapplayer
```

Note: In Windows, if shortcuts were created during installation, MapPlayer can be started by selecting Start>Programs>webMethods JI 4.5>MapPlayer.

The **webMethods JI MapPlayer** window opens (Figure 16).

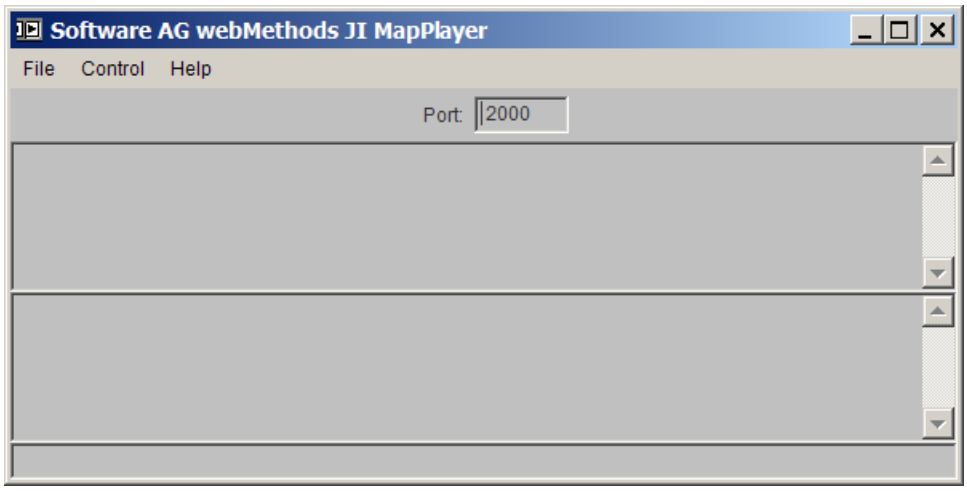

Figure 16. The MapPlayer Window

2 Select **File > Open**. The **Open Map File** dialog box opens.

3 Browse to the directory in which the *MajorBankHost.map* map file is located.

Note: The MajorBankHost.map file is included in the webMethods JI installation, and its default location is *<JI_install_dir>/examples/maps*.

4 After locating the *MajorBankHost.map* file, click on the **Open** button to open the file in MapPlayer.

Note: When the file is successfully loaded, the MapPlayer status bar displays the message “Map loaded and ready...”.

5 Select **Control > Start**. This starts the map file and instructs MapPlayer to accept incoming requests from terminal emulators (such as MapMaker).

After starting the map, you will use MapMaker to connect to MapPlayer and begin constructing your first map.

Starting MapMaker

To start the MapMaker graphical development environment, proceed as follows:

1 Start the MapMaker application using the following command:

```
cd <JI_install_dir>\bin
ea_mapmaker
```

> **Note:** In Windows, if shortcuts were created during installation, MapMaker can be started by selecting Start>Programs>webMethods JI 4.5>MapMaker.

The **webMethods JI MapMaker** user interface opens (Figure 17).

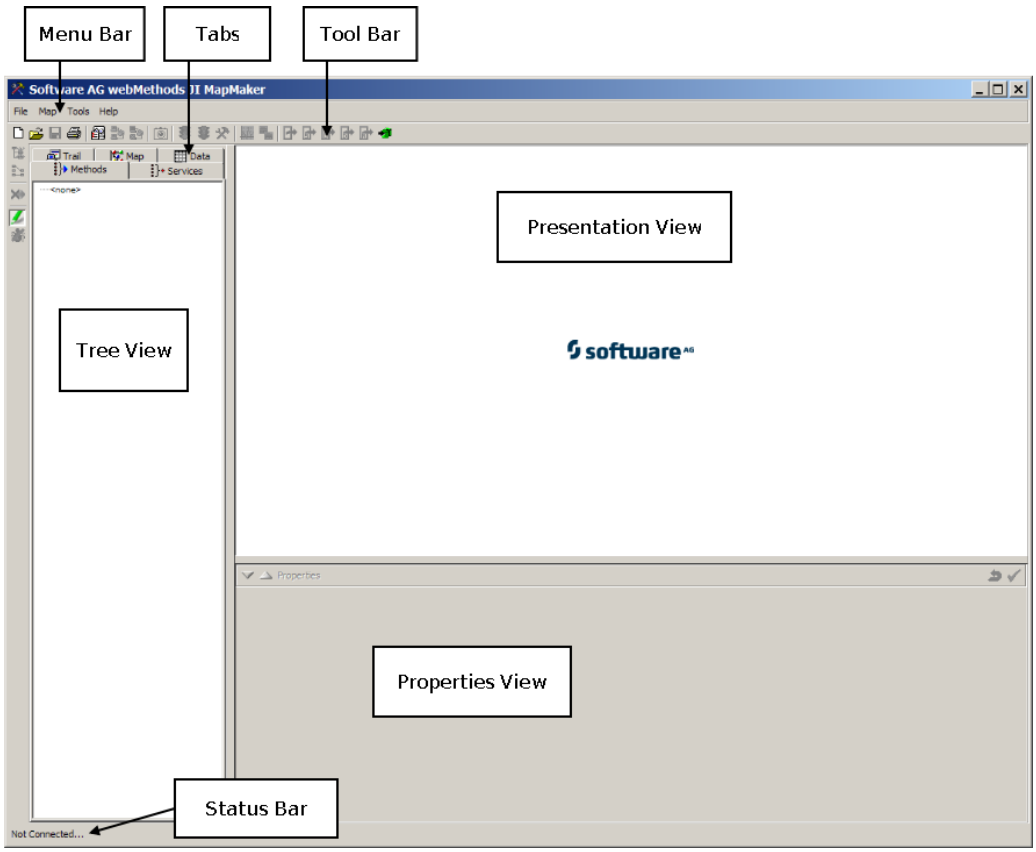

Figure 17. The MapMaker User Interface

Recording a Trail Through a Connection to the Host

This section explains how to connect to the host you have defined in the previous chapter (see "Defining the Host Connection" on page 40), in order to record a trail. There are two different ways to connect to the host and to begin trail recording:

- Starting the trail first, before connecting to the host.
- Connecting to the host first, then navigating to a specific point in the legacy application and only then starting the trail.

When creating your initial trail, you need to capture the login screen and other initial navigational information in the trail. For best results, we recommend that in this case you start the trail first, prior to connecting to the host.

Subsequent trails may not require the login screen. For example, you may want to create trails that traverse from various screens back to the main menu. In this case, you would connect to the host first and navigate to the correct screen. Only then you would start trail recording, perform the required navigational steps and stop trail recording once you have returned to the main menu.

For the purposes of this exercise, we will record the login screen. This means we will start trail recording first, prior to connecting to the host.

To begin trail recording through a connection to the host, proceed as follows:

1 To create a new map, select **File** > **New** or click on the `New` button .

A new map whose default name is **Map_0** is shown in the **Trail** tab's Tree view (Figure 18).

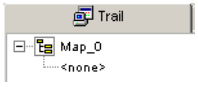

Figure 18. A New Map Added to the Trail Tab's Tree View

2 To connect to the host, select **Map** > **Start Trail** or click on the **Start Trail** button .

The **Connect to Host** dialog box opens (Figure 19).

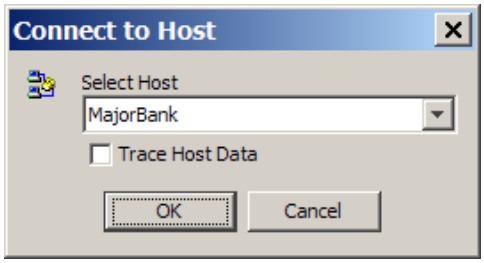

Figure 19. Connect to Host Dialog Box

3 Select the **MajorBank** host you have defined in the previous chapter (see "Defining the Host Connection" on page 40) from the **Select Host** list, and click **OK**.

The opening login screen of the *MajorBankHost.map* map file is displayed in the Presentation view (Figure 20).

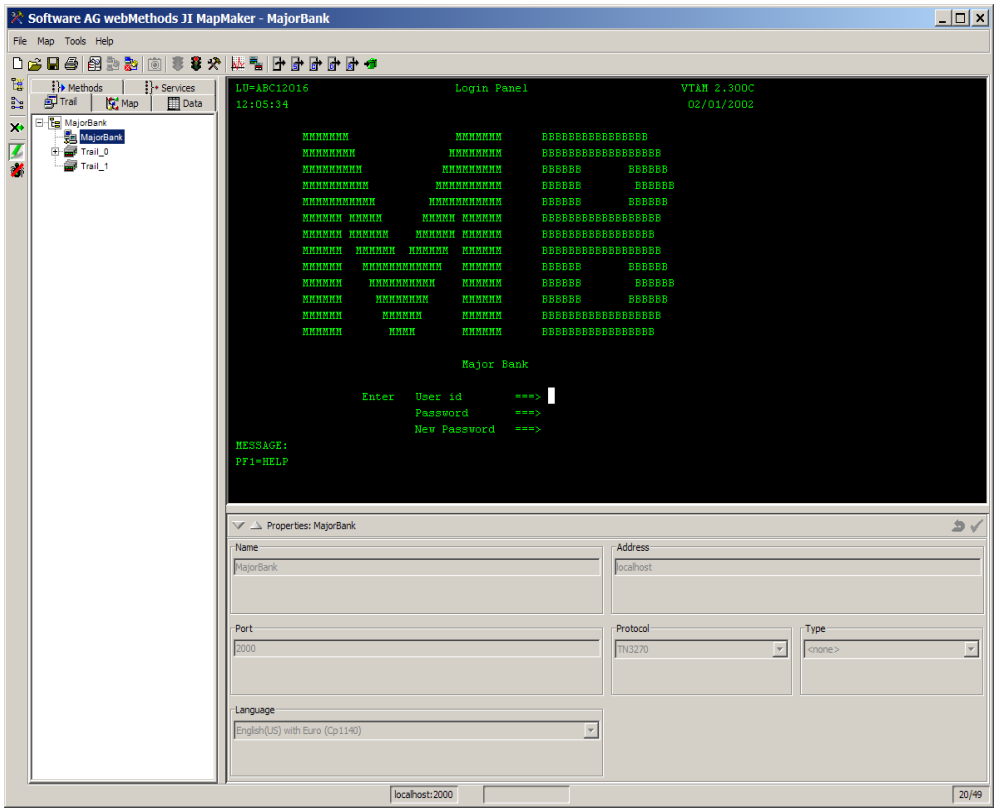

Figure 20. Login Screen

During trail recording, MapMaker basically functions like a terminal emulator, so you can simply interact with the legacy application just like you would if attached to the mainframe.

While you are interacting with the legacy application, MapMaker records the screens, keystrokes and transition information (AID keys), and saves this information in the **Trail** Tree.

Note: By default, MapMaker's bottom right pane displays the 'Properties' view, which shows the properties of the object selected in the Tree. You can toggle the display of the 'Properties' view using the Contract button and the Expand button . For more information on the 'Properties' view, see "Properties View" on page 32.

4 To log into the application, type `mbuser` in the **User id** field and press the Tab key to advance to the **Password** field.

5 In the **Password** field, enter `mbpass` and press the **Enter** key.
This advances the webMethods JI tutorial banking application to the **Main Menu** screen (Figure 21).

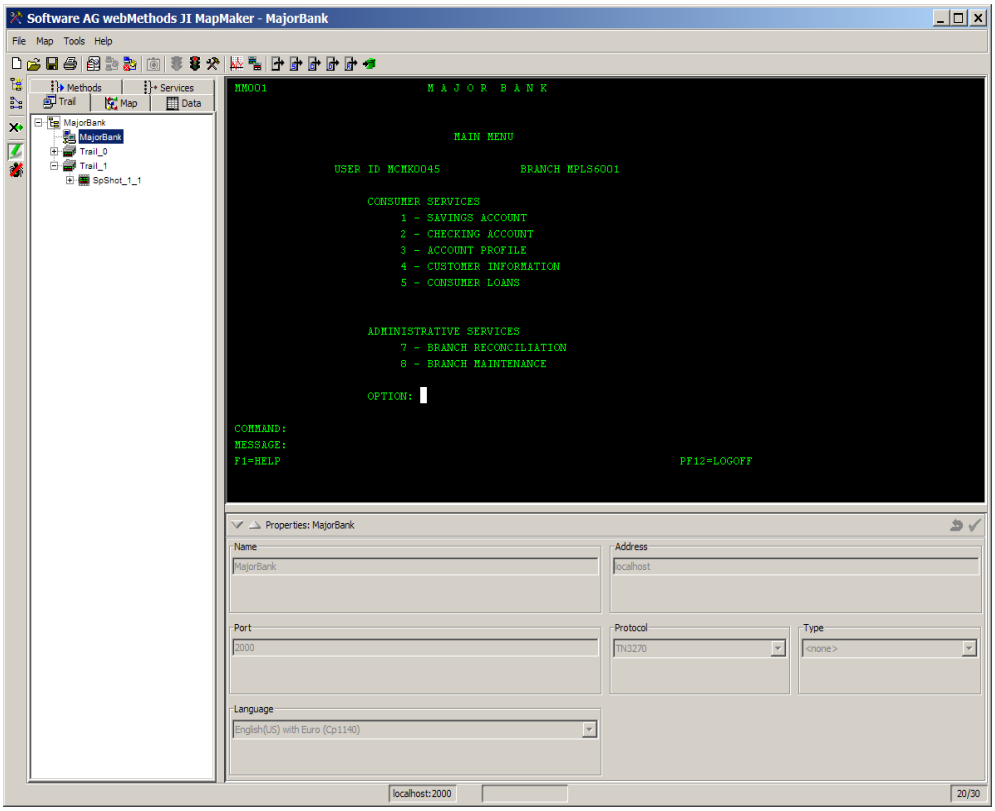

Figure 21. The Main Menu

Note: A screen snapshot ('SpShot_1_0') has been added to the 'Trail' Tree on the left side of the MapMaker interface. This snapshot is a picture of the application's login screen. Selecting the snapshot in the Tree displays its associated screen in the Presentation view. After viewing the snapshot, you can return to the active host session by clicking on the host name (e.g. 'MajorBank') or icon in the 'Trail' Tree.

6 At the prompt in the **Main Menu** screen, type `5` and press **Enter** to advance to the **Customer Lookup** screen (Figure 22).

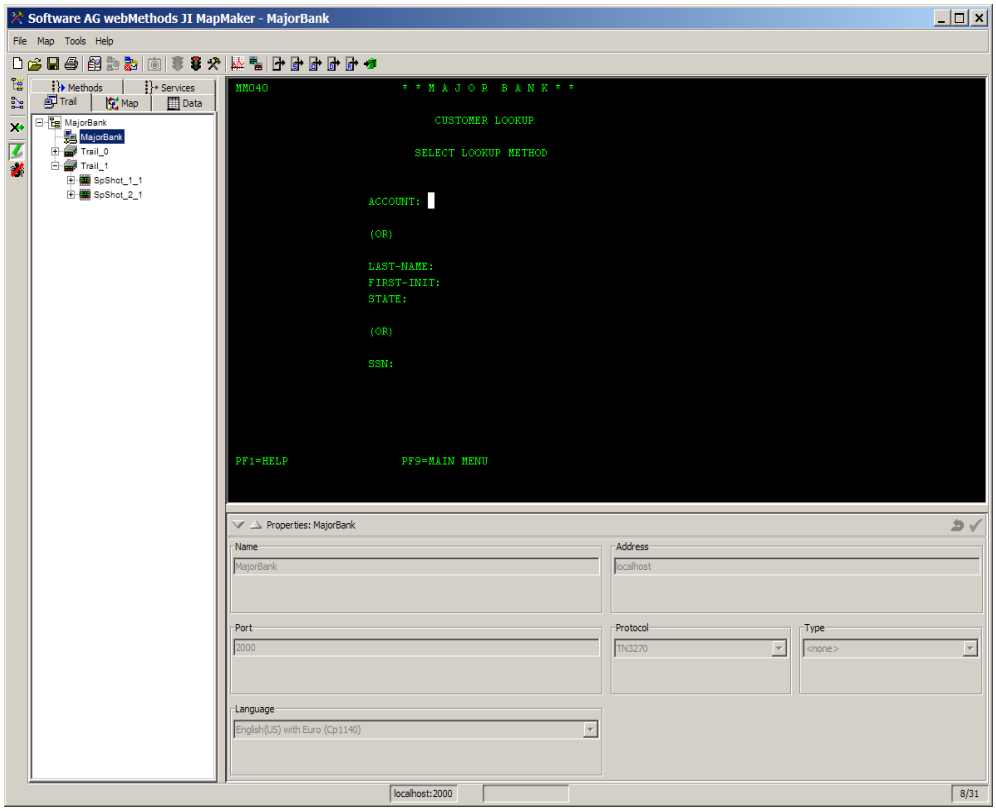

Figure 22. The Customer Lookup Screen

Note: Another screen snapshot (SpShot_2_0) is added to the list when the AID key (Enter) is pressed.

7 In the **Customer Lookup** screen, enter `1234567898` in the **Account** field and press **Enter**. This accesses the **Consumer Loans** screen (Figure 23).

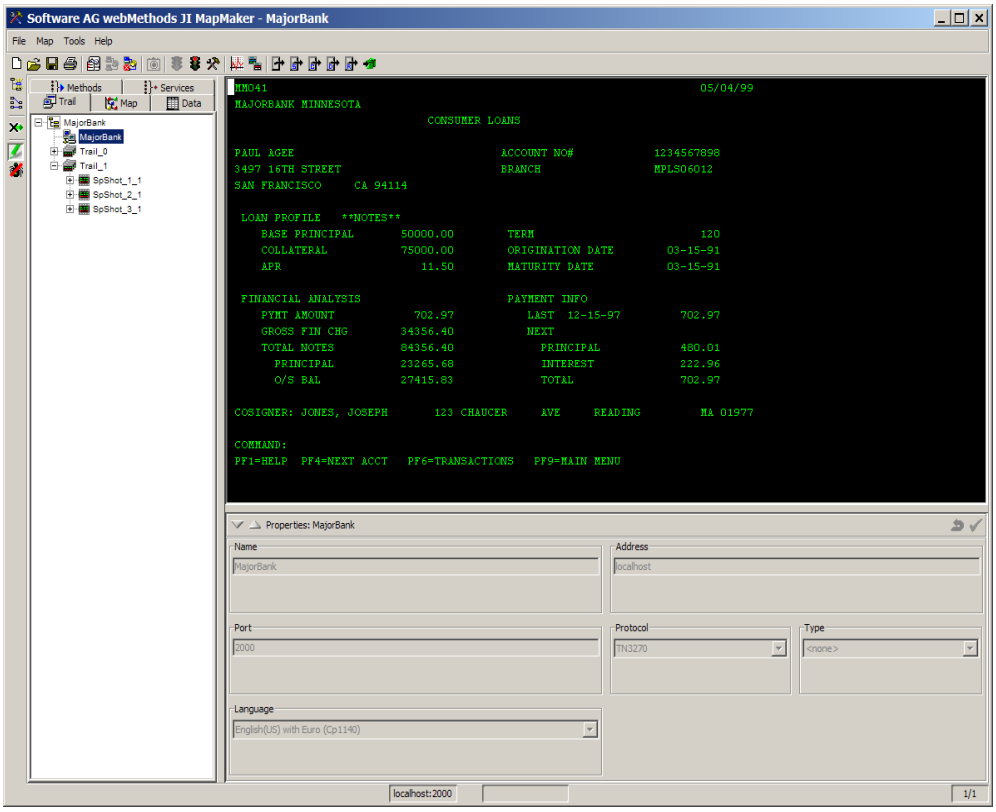

Figure 23. The Consumer Loans Screen

8 Access the loan **Transactions** for this account by pressing **PF6** (**F6** on the keyboard).

The **Loan Transactions** screen is displayed (Figure 24).

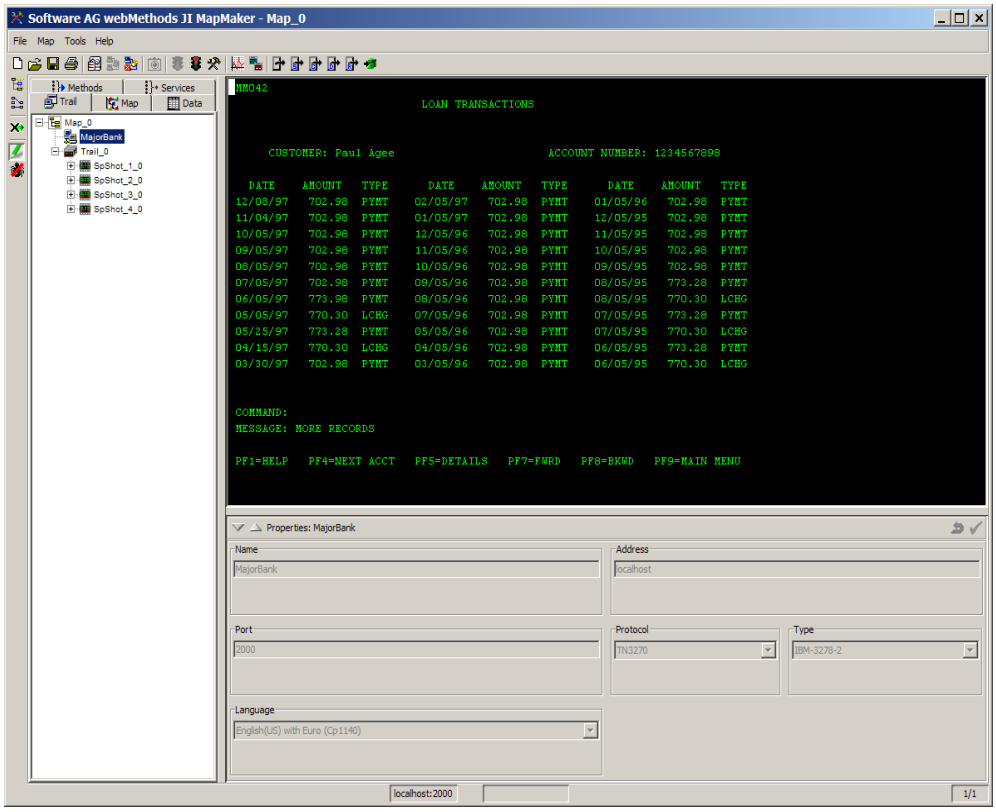

Figure 24. The Loan Transactions Screen

9 Observe that the **Loan Transactions** screen includes a **MESSAGE** field (at the bottom), indicating that **MORE RECORDS** are available for display.
Press **PF7** (**F7** on the keyboard) to advance to the next screen of transactions.

10 Once again, observe that **MORE RECORDS** are available for display.
Press **PF7** to advance to the next screen of transactions.

11 On this screen, the **MESSAGE** field contains the message **END RECORDS**, indicating the this is the last screen of data.
Press **PF4** (**F4** on the keyboard) to return to the **Customer Lookup** screen (Figure 22).

12 Press **PF9** (**F9** on the keyboard) to return to the **Main Menu** screen (Figure 21).

13 Press **PF12** (**F12** on the keyboard) to return to the **Login** screen (Figure 20).

14 This is the final screen in our first trail. Stop the trail recording mode by selecting **Map** > **Stop Trail** or clicking on the **Stop Trail** button .

15 At this point, you should also disconnect from the host (MapPlayer), by selecting **File** > **Disconnect** or clicking the **Disconnect** Button .

16 Save the map by selecting **File** > **Save As** and naming the map `MajorBank.map`.

Note: Be sure to include the extension ".map" in all of your maps.

Updating the Application Map

Now that the Trail recording is complete, the map needs to be updated. The update operation causes MapMaker to automatically analyze the trail and form an application map based on the host screens, the keyboard input and the AID keys.

1 To switch to the Map panel, click the **Map** tab in the Tree view (Figure 25).

Figure 25. Map Tree View

Currently, this panel is empty, except for the **Global Actions** branch that has no sub-components. After updating the map, map screens are listed in this panel, based on the snapshots that MapMaker took during the Trail recording.

2 To update the map, select **Map > Update Map** or click on the **Update Map** button ( ).
A dialog box is displayed, stating the update of the Map has been completed. Click **OK** to continue.

Note: Once the map is updated, the **Map** Tree displays six screens (**Screen_0** to **Screen_5**). The screens displayed in the Tree are listed in the order in which they were recorded.

3 Next, we will rename the screens in the map. The screens are given generic names by default, and at this point it would be better to have more specific names, to make recognition easier later on in the service generation process.

To rename an object, right-click on its name and select **Rename** from the pop-up menu.

Note: Alternatively, you can rename an object by selecting it and pressing the 'PF2' key (F2 on the keyboard). The object name becomes available for editing, and you can specify the new name and press 'Enter' to apply the change.

Rename the screens as follows:

Default Screen Name	New Screen Name
Screen_0	scrnLogon
Screen_1	scrnMainMenu
Screen_2	scrnCustomerLookup
Screen_3	scrnConsumerLoans
Screen_4	scrnLoanTransactions

Identifying to MapMaker that Two Screens are the Same

Notice that there is an extraneous screen in the Tree view, **Screen_5**. This screen, just like the one we renamed **scrnLoanTransactions**, contains loan transactions; however, it is identified by MapMaker as a different screen.

By default, screens are identified by the number, location and length of the fields they contain. As is the case with most host applications, if a screen containing a table has fewer records than the screen can hold, the remaining empty records are simply defined with space-holder fields.
In the case of **Screen_5** and **scrnLoanTransactions**, MapMaker identifies them as being different because they have a different number of fields.

You can easily identify to MapMaker that **Screen_5** and **scrnLoanTransactions** are actually the same screen, by manipulating **Screen_5**'s sub-components.

When a screen node is expanded in the **Map** Tree, the following sub-components are displayed:

Sub-component	Description
Fields	List the formatted fields that MapMaker automatically detected in the screen. MapMaker uses these formatted fields to identify the screen.
Tags	Identify one of the following: • A common element on a screen, in order to combine screens • A unique element on a screen, in order to separate screens
Actions	Include all of the user interactions, plus the keystrokes that were used to traverse between screens. **Note:** For more information on actions, see "Creating Global Variables" on page 56.
Snapshots	List the snapshots included in this screen component. MapMaker determines which screens are identical based on the layout of their formatted fields, regardless of the data displayed in the fields. As a result, two or more screens in a trail are often interpreted as identical by MapMaker. All identical screens are listed together, and only those that are unique have their own Screen component in the map.

The following procedure identifies that **Screen_5** and **scrnLoanTransactions** are the same screen:

1. Expand **scrnLoanTransactions** by clicking on the **+** to its left.
2. Right-click on the **Fields** branch and select **Disable All Fields** from the shortcut menu. This tells MapMaker not to use the fields as the criteria for identifying the screen.
3. Now we need to provide MapMaker with an alternative criterion that identifies this screen. This criterion is a Tag, which singles out a unique identifier on the screen and can be used to label that screen.

   Using the mouse and the standard "drag" selection technique, select **MM042** in the upper left corner of the screen. When the **MM042** is contained in the white box, right-click in the box and select **Add Tag** (Figure 26).

Figure 26. Adding a Tag

Tag_0 is created under the **Tags** screen node (Figure 27). MapMaker can now identify all screens containing the tag (in this case, **scrnLoanTransactions** and **Screen_5**) as the same screen.

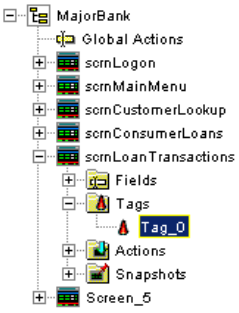

Figure 27. A New Tag in the Map Tree

4 To combine the screens, update the map again (select **Map > Update Map** or click on the **Update Map** button).

5 An **Attention** dialog box is displayed, stating that this update will cause **Screen_5** to be lost.

Click **Yes** to continue with the update.

6 A dialog box is displayed, stating the update of the Map has been completed.

Click **OK.**

7 Observe that Screen_5 has been removed from the **Map** tab, and that the screens have been combined: expand the **scrnLoanTransactions** branch and then expand its **Snapshots** branch. Note that the **Snapshots** branch now contains three snapshots.

8 Click on each snapshot to see the associated screen image. Note that snapshot **SpShot_7_0** contains the screen image that was formerly named Screen_5.

Creating Global Variables

Next, we will create global variables that allow you to store information (data) in the map.

Creating Actions During the Trail Recording Process

In the course of the trail recording process, you entered the account number `1234567898` into the **Customer Lookup** screen, and pressed the Enter AID key to advance to the **Consumer Loans** screen (see Step 7 on page 50). The 'Update Map' operation not only identified the unique screens within the trail, but also turned the keystrokes (i.e. `1234567898`) and aid keys (i.e. `Enter`) entered into *Actions*.

Actions define the keystroke input operations required to move from one screen to another. For example, the **Customer Lookup** screen (**scrnCustomerLookup**) now includes an action named **toscrnConsumerLoans**. Selecting this action in the **Map** Tree view shows its properties in the **Properties** view. The **Action Input** tab of these properties defines entering **1234567898** and pressing the **Enter** key as the operations required to move to the **Consumer Loans** screen (see Figure 28).

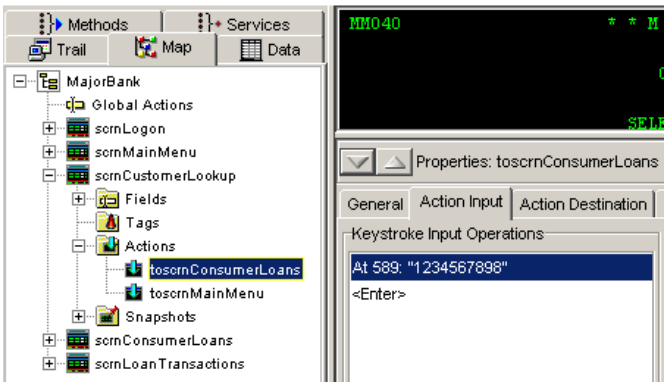

Figure 28. The toscrnConsumerLoans Action's Keystroke Input Operations

> **Note:** Actions are automatically named after their destination screens. These names are not configurable.

When executed, the **toscrnConsumerLoans** action enters the specified account number into the **Customer Lookup** screen, and presses the **Enter** key to move to the **Consumer Loans** screen.

Setting Actions to Use Global Variables

Currently, the **toscrnConsumerLoans** action can only enter a specific account number (`1234567898`) into the **Customer Lookup** screen. However, we need to be able to look up *any* loan account. This is accomplished by taking the following actions:

1. Creating a *global variable* that holds the loan account number.
2. Setting the action properties to use the account number held by the global variable.

Global variables are defined in the **Business Entity Editor**. For now, you will use the **Business Entity Editor** to define global variables in their simplest form. You will learn more about the **Business Entity Editor** in the Data Modeling section later in this chapter (see "webMethods JI Data Modeling" on page 69).

To define a global variable, proceed as follows:

3 Open the **Business Entity Editor** (Figure 29) by selecting **Tools** > **Edit Business Entities…** or by clicking the **Edit Business Entities** button on the tool bar.

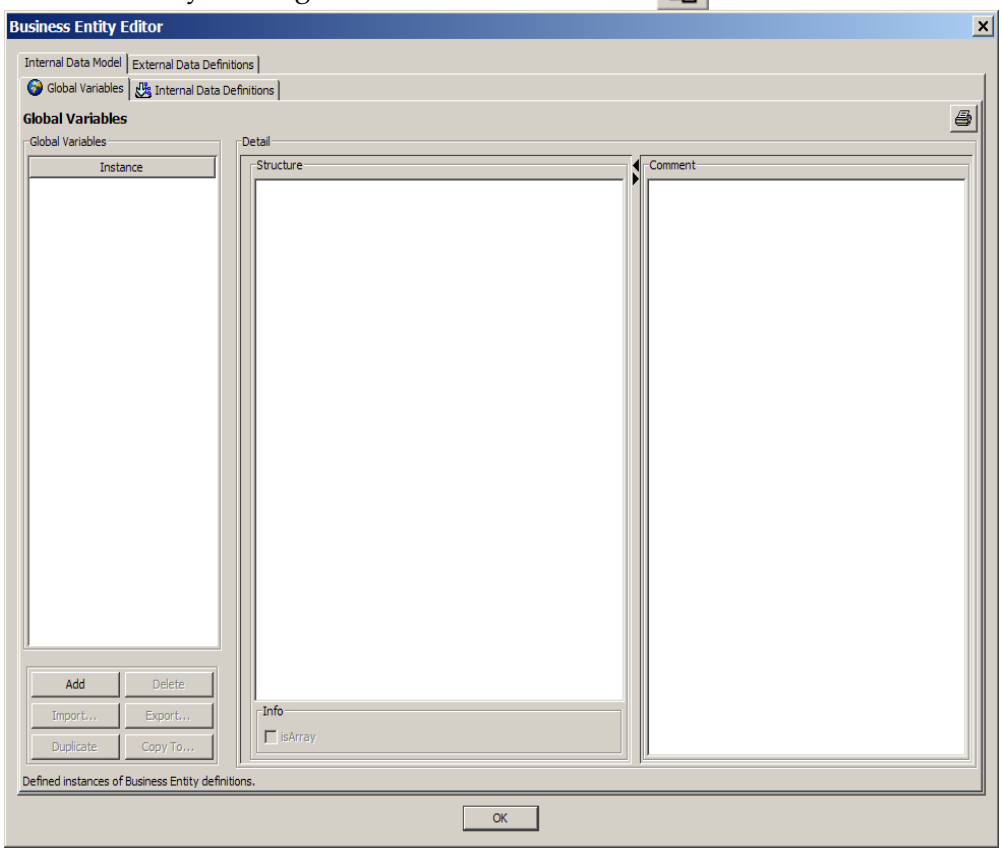

Figure 29. The Business Entity Editor

4 In the **Global Variables** tab, click the **Add** button.

The **Choose a Type** dialog box is displayed (Figure 30).

5 In the **Select Type** list, choose **InternalString** as the type of global variable to be created and click **OK**.

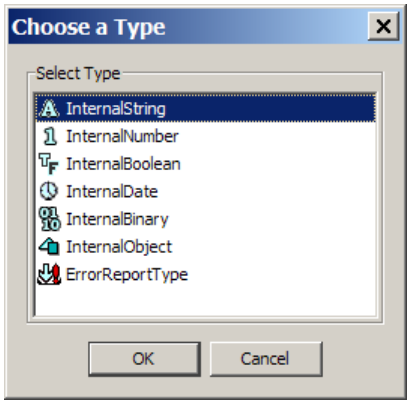

Figure 30. Choose a Type Dialog Box

A new global variable (**gvString**) is added to the **Global Variables' Instance** list (Figure 31).

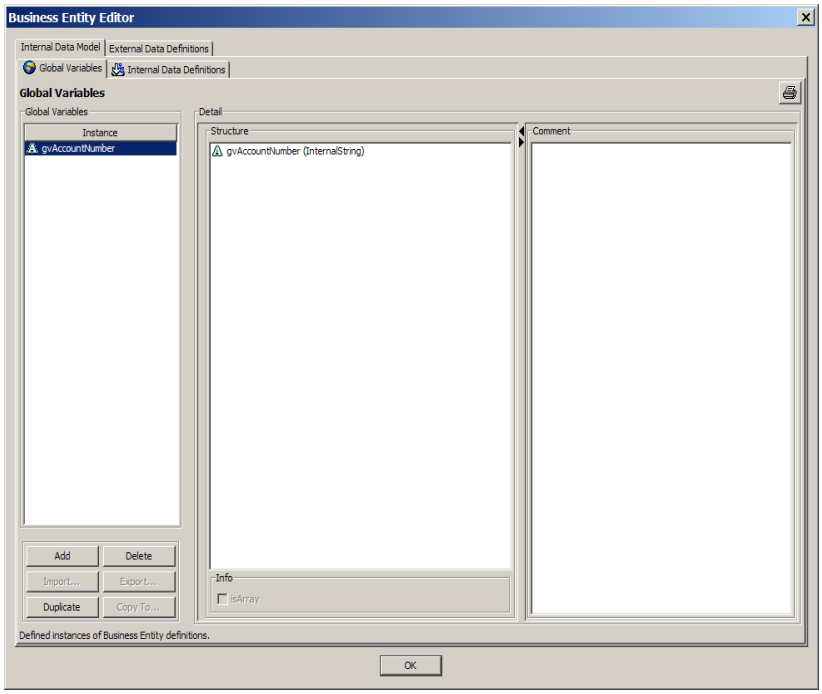

Figure 31. Business Entity Editor > New Global Variable

6 Rename the new global variable by right clicking on it in the **Instance** list, selecting **Rename** from the shortcut menu, specifying the name `gvAccountNumber` and pressing the **Enter** key.

7 Click **OK** to close the **Business Entity Editor**.

Linking Global Variables to Actions

After creating a global variable, you need to link it to the action that will use it. The global variable **gvAccountNumber** you have just created is to be used by the **toscrnConsumerLoans** action, contained in the screen **scrnCustomerLookup**. Proceed as follows:

1. To include the **gvAccountNumber** global variable in the **toscrnConsumerLoans** action, you must first expand the Tree view (if it is not already expanded). Click on the "**+**" symbol next to the screen named **scrnCustomerLookup**, to reveal its sub-components: **Fields**, **Tags**, **Actions**, and **Snapshots**.
2. Next, expand the **Actions** component, to reveal the action named **toscrnConsumerLoans** that is associated with this screen (see Figure 28).
3. Select the **toscrnConsumerLoans** action in the Tree to display its properties in the **Properties** view.

> **Note:** If the 'Properties' view is not displayed, click on the Expand button ( ) at the bottom of the Presentation view, or right-click on 'toscrnCustomerLoans' and select 'Properties' from the pop-up menu.

4. Display the **Action Input** tab (Figure 32).

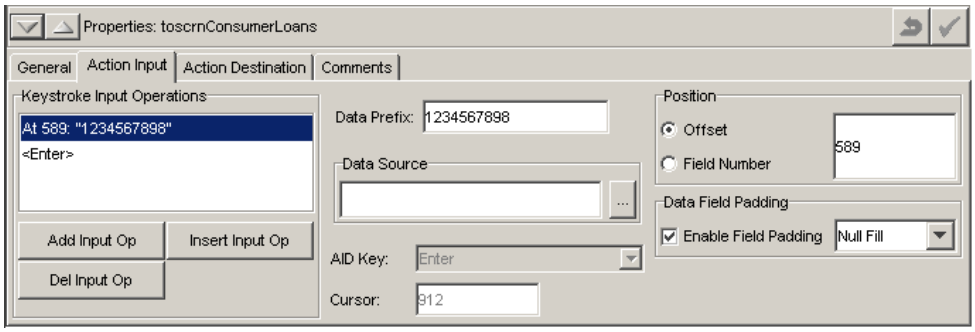

Figure 32. Action Properties View > Action Input Tab

As you can see in the **Data Prefix** property, the account number recorded for this action is **1234567898**.

5. In the **Keystroke Input Operations** list, select **At 589: "1234567898"**, and then delete the account number **1234567898** from the **Data Prefix** property.
6. To assign a global variable to this keystroke input operation, click the ellipsis button ... located in the **Data Source** field.

   The **Choose a Data Source** dialog box is displayed (Figure 33).

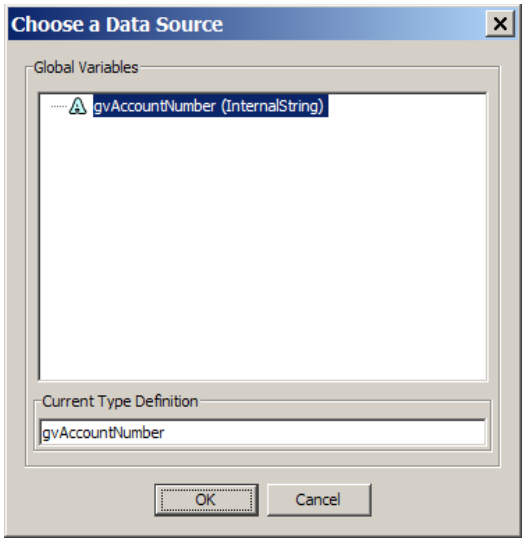

Figure 33. Choose a Data Source Dialog Box

7 Select the global variable **gvAccountNumber (InernalString)**, which is the only one currently listed, and click **OK**.

8 Accept the changes to the Keystroke Input Operation by clicking the **Apply** button at the upper right corner of the **Properties** view.

gvAccountNumber is added to the selected Keystroke Input Operation.

You have now changed the **toscrnConsumerLoans** action to use an account number stored in the global variable **gvAccountNumber**.

9 Save the map by selecting **File** > **Save**.

Defining a Data Template

At this point, you can define templates that identify the data to be returned from the captured screens. A template defines specific fields on the legacy screen, from which data is to be extracted. After these fields are defined, they are retrieved using the "Fetch" steps defined in the service Methods. The Methods extract the information from these fields, and either return it to the client or use it as input for other legacy application fields.

There are two types of templates: data templates and table templates. Data templates are used to return non-tabular data, while table templates, as their name implies, are used to return tabular (row/column) data.

We will now define a data template for the **Consumer Loans** screen (**scrnConsumerLoans**).

Adding a Data Template to the Map

To add a data template to the map:

1 Switch to the Data panel of the Tree view by clicking on the **Data** tab.

2 Highlight the **scrnConsumerLoans** screen to display its snapshot in the Presentation view.

3 Right-click on the screen name and select **Add Data Template** from the shortcut menu (Figure 34).

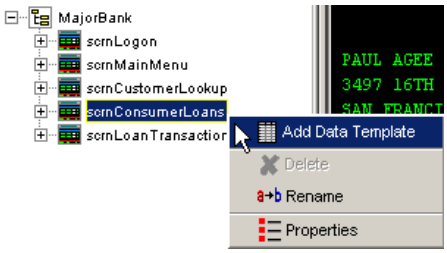

Figure 34. Add Data Template

This adds a new data template node (**dtscrnConsumerLoans**) to the screen.

4 Right-click on **dtscrnConsumerLoans**, choose **Rename** from the pop-up menu and enter the name `dtLoanDetails`.

Adding Data Fields to the Data Template

To add data fields to the data template:

1 With the data template sub-component (**dtLoanDetails**) highlighted in the Tree, we will now add the customer name field to the data template, by double-clicking on **PAUL AGEE** on the legacy screen. Double-clicking on a formatted field automatically selects it.

> **Note:** If the field is not a formatted field, you can click and drag to select it.

2 After the customer name field is selected, right-click on it and select **Add Data Field** from the pop-up menu (Figure 35).

Figure 35. Adding a Data Field

This adds a new data field node (**DataField_0**) to the screen.

3 Right-click the new Data Field node, choose **Rename** from the pop-up menu and enter the name `dfName`.

4 Select the data template component in the Tree once again, and repeat Step 1 to Step 3 for each field listed in the table below.

Legacy Screen Field	Change the Field Name to...
Account Number (`1234567898`)	`dfAccountNumber`
Base Principal (`50000.00`)	`dfBasePrincipal`
Collateral (`75000.00`)	`dfCollateral`
Term (`120`)	`dfTerm`

Note: You can first add all the fields, and then rename them.

Defining a Table Template

Next, we will add a table template. A table template is comparable to a data template, except that the table template is used to extract data from *repeating*, tabular areas on the legacy screen.

Creating a Table Template for the Loan Transactions Table

To create a table template for the loan transactions table:

1 Highlight the **scrnLoanTransactions** screen component in the Tree view, to display the corresponding snapshot in the Presentation view. This screen contains a repeating table of information about the customer's account.

2 With the **scrnLoanTransactions** screen highlighted in the Tree view, select the table in the Presentation view, by clicking in its upper left corner and dragging to enclose the table in a red box (Figure 36).

After selecting the table, you can resize the selected area as needed by positioning the cursor over the table template's resizing handles and dragging them to the desired location.

Note: As this table repeats across the screen, ensure that the table is selected exactly as depicted in Figure 36 below.

3 When the table is selected correctly, right-click on it and select **Add Table Template** from the pop-up menu (Figure 36).

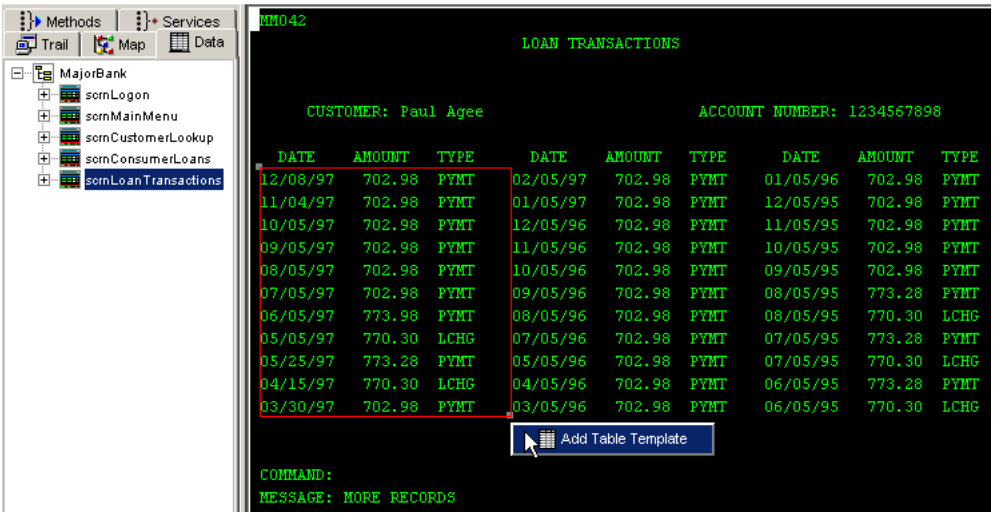

Figure 36. Adding a Table Template

A new table template named **ttscrnLoanTransactions** is added to the **scrnLoanTransactions** node of the Tree.

Adding Data Fields to the Table Template

To add data fields to the table template:

1 With the **ttscrnLoanTransactions** table template highlighted in the Tree, add the **Date** field from the legacy screen to the table template. This is done by double-clicking on the first date field in the table to select it (**12/08/97**), then right-clicking it and selecting **Add Data Field** from the shortcut menu (Figure 37).

> **Note:** When you double-click on a formatted field, it is automatically selected. If the field is not formatted, click and drag a rectangle to select it.

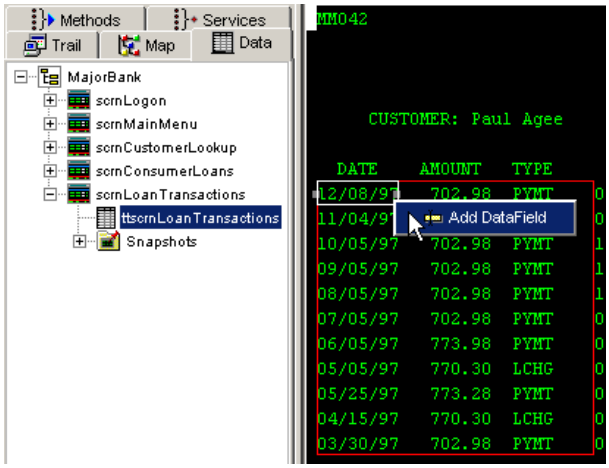

Figure 37. Adding a Data Field to a Table Template

A new data field sub-component (**DataField_0**) is added to the **ttscrnLoanTransactions** node of the Tree.

2 Right-click on the new data field, select **Rename** from the pop-up menu and change the name of the field to `dfDate`.

3 Highlight the table template component in the Tree view again, and repeat Step 1 to Step 2 for the next two fields in the table, renaming them as specified in the table below.

> **Note:** After adding each field, make sure that the table template is highlighted in the Tree view before attempting to add the next field.

Legacy Screen Field	Change the Field Name to...
Amount (`702.98`)	`dfAmount`
Type (`PYMT`)	`dfType`

Using the Message Field's "More Data" Indicator

As noted during the recording process, this table can span across screens. When the table spans across multiple screens, the **Message** field contains a "`MORE RECORDS`" string. If there is only one screen of data, or when the last screen of data is reached, the message field contains an "`END RECORDS`" string. The **Message** field can be used to determine if there are additional data screens to be obtained.

When reading data from a screen, the data can either be returned to a client or stored in a global variable. We now need to read the contents of the **Message** field, and use this value to determine whether to page forward to obtain additional data. To do this, another global variable is required.

1. Define a second global variable for storing the **Message** field value:
   - **a** Open the **Business Entity Editor** (Figure 29) by selecting **Tools** > **Edit Business Entities…** or by clicking the **Edit Business Entities** button on the tool bar.
   - **b** In the **Global Variables** tab, click the **Add** button.
     The **Choose a Type** dialog box is displayed (Figure 30).
   - **c** In the **Select Type** list, choose **InternalString** as the type of global variable to be created and click **OK**.
     A new global variable named **gvString** is added to the **Instance** list of the **Global Variables** pane.
   - **d** Rename the new global variable **gvMessage**, and click **OK** to exit the **Business Entity Editor.**
   - **e** Define a new data template for storing the **Message** field:
   - **f** In the **Data** tab's Tree, select **scrnLoanTransactions**.
   - **g** Right-click and select **Add Data Template** from the pop-up menu.
     A new data template (**dtscrnLoanTransactions**) is added to the screen.

> **Note:** The Data Templates and Table Templates listed in the Tree view are sorted alphabetically. Therefore, once you add or rename these templates, they may be moved to maintain the appropriate order.

   - **h** Right-click the new data template, select **Rename** from the pop-up menu and enter the name `dtMessage`.
2. Add the **Message** data field to the data template:
   - **a** With the **dtMessage** data template highlighted in the Tree, double-click the **Message** field value (**MORE RECORDS**) in the Presentation view in order to select it.
   - **b** Right-click on **MORE RECORDS** and select **Add DataField.**
     A new data field (**DataField_0**) is added to the data template.
   - **c** Right click on the new data field, select **Rename** from the pop-up menu and enter the name `dfMessage`.
3. Store the **dfMessage** data field value into a global variable:

a Select the `dfMessage` data field element in the Tree view to display its properties in the **Properties** view (Figure 38).

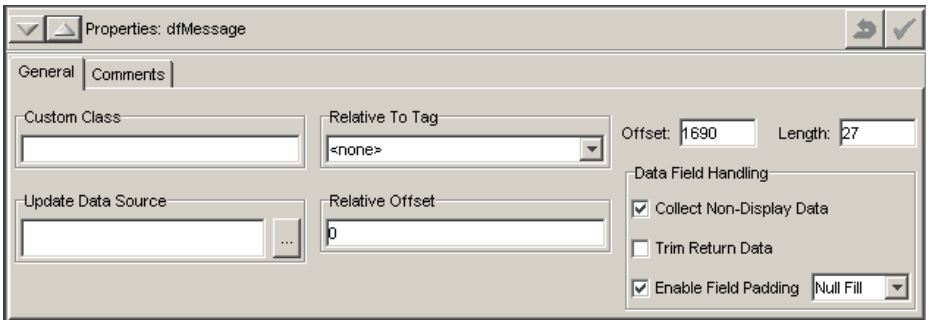

Figure 38. Data Field Properties > General Tab

b In the **General** tab of the **Properties** view, click on the ellipsis button ... located under **Update Data Source**.
The **Choose a Data Source** dialog box is displayed (Figure 33).

c In the **Global Variables** list, select **gvMessage** and click **OK**.

4 Accept the changes to the global variable properties by clicking on the **Apply** button ✓.
You have now set the **dfMessage** data field to store its contents (e.g. '`MORE REOCRDS`') in the global variable **gvMessage**.

5 Save the map by selecting **File > Save**.

Setting the Table Template Properties

To set the table template properties:

1 Select the **ttscrnLoanTransactions** table template in the Tree to display its properties in the **Properties** view (Figure 39).

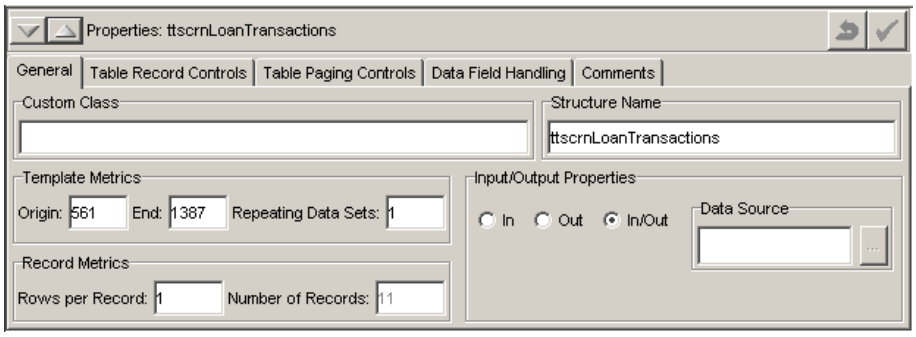

Figure 39. Table Template Properties - General Tab

2 In the **General** tab of the **Properties** view, go to the **Template Metrics** section and change the **Repeating Data Sets** value from `1` to `3`. This reflects the three occurrences of the Loan Transactions table on this screen, and will cause the Table Template to repeat three times across the screen.

3 Display the **Table Paging Controls** tab (Figure 40).

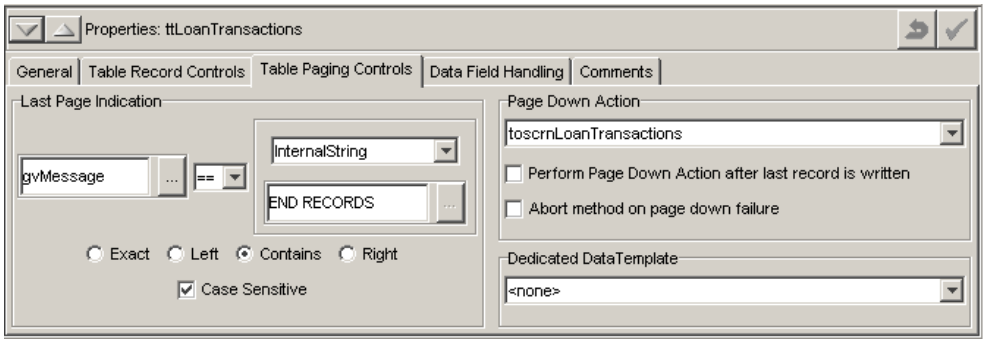

Figure 40. Table Template Properties > Table Paging Controls Tab

4 Configure the **Last Page Indication** section using the gvMessage global variable you have defined in the previous procedure (“Using the Message Field’s “More Data” Indicator” on page 65), as shown in Figure 41:

   a Set the first text box to **gvMessage** in one of the following ways:
   - Enter the text `gvMessage`, *Or*
   - Click on the ellipsis button ... to display the **Choose a Data Source** dialog, and select **gvMessage** from the **Global Variables** list.

   b Make sure the middle drop-down list is set to = =.

   c From the object type drop-down list on the right, select **InternalString**.

   d In the text box below, enter `END RECORDS`.

   Make sure the **Contains** radio button is set.

   Make sure the **Case Sensitive** check box is set.

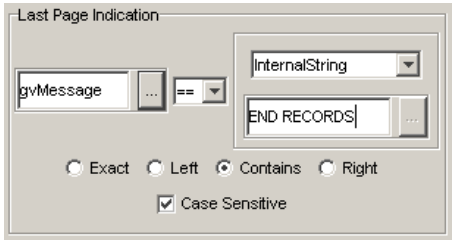

Figure 41. Table Template Properties > Table Paging Controls Tab: Last Page Indication

5 Next, you need to define how the table template is to page down.
During the trail recording process, the PF7 key was used to advance to the next page, i.e. the next set of transactions on the **scrnLoanTransactions** screen. Therefore, the action to be selected from the **Page Down Action** drop-down list is **toscrnLoanTransactions**.

6 Finally, define which data template to use to read the **Message** field.
In the **Dedicated Data Template** list, select **dtMessage**.

7 Click **Apply** ✓ to apply the changes.

   At this point, the required templates have been added and the map is complete.

8 Select **File > Save** to save the map file.

webMethods JI Data Modeling

Now that you have defined the specific fields on the legacy screen from which data is to be extracted, the next step is to perform data modeling. Data modeling is the process of defining each piece of data as being of a particular data type, and of defining a data type-specific "container" for each piece of data. This "container" is known as a variable, and its function is to transfer the data between the client and the legacy application. Other functions of data modeling are:

- Defining the relationship between one data type and another.
- Mapping the flow of data within the chronological flow of a method.

webMethods JI provides extensive data modeling capabilities. Unlike other tools, which require you to use their data formatting standards or use external pre/post processing (such as XSLT or third-party conversion tools), webMethods JI's data modeling allows webMethods JI to conform to your data standards without those requirements.

To illustrate the use of data modeling, we will impose the following input and output data requirements on the Method you will be creating. Since this Method (transaction) will be obtaining loan information for a given account, the account number needs to be passed in from the client in the form of an XML document. The Method will then gather the data that has been identified on the **Consumer Loans** and **Loan Transactions** screens, and return the data as an XML document.

Example Input XML Document

This chapter uses the following XML document as an example of data input:

```
<AccountNumber></AccountNumber>
```

Example Output XML Document

This chapter uses the following XML document as an example of data output:

```
<LoanInformation>
    <AccountNumber></AccountNumber>
    <Name></Name>
    <Profile>
        <BasePrincipal></BasePrincipal>
        <Collateral></Collateral>
        <Term></Term>
    </Profile>
    <Transactions>
        <Amount></Amount>
        <Date></Date>
```

```
        <Type></Type>
    </Transactions>
</LoanInformation>
```

Note: The 'Transactions' element of the XML document repeats for the actual number of transactions being returned.

Data Structure Types

Data Structures, also known as Business Entities, are essentially structures that define how data is transferred through the runtime. A Business Entity may represent either a single data object, or a structure consisting of various data objects. During runtime, every "crossroad" the information goes through is represented by Business Entities.

There are two main types of Business Entities:

- "External Business Entities (XBEs)" on page 70.
- "Internal Business Entities (IBEs)" on page 71.

External Business Entities (XBEs)

External Business Entities (XBEs, in short) represent data structures that interact with data sources that are external to webMethods JI. These external data sources include client input and client output, in one of the following forms:

- XML documents
- Map-List-Map (MLM) structures, required by webMethods JI programmatic APIs
- Legacy (green screen) applications

Accordingly, there are three types of XBEs: XML/XSD, MLM and Legacy.

XML/XSD XBEs XML/XSD XBEs represent XML documents. These are specified using XML Schema Definitions (XSDs), which include element types (e.g. XSDStringValue, XSDDecimalValue etc.) and particles to describe the minimum and maximum occurrences.

MLM XBEs MLM XBEs represent Map-List-Map data structures, used by webMethods JI's proprietary programmatic APIs. The latter also support sending and receiving XML documents.

Legacy XBEs Legacy XBEs represent data templates and table templates. These XBEs are created automatically anytime you create a data or table template. The XBEs also reflect any changes made to their corresponding templates.

> **Note:** Legacy XBEs are for reference purposes only, and cannot be edited. Any changes to legacy XBEs must be made directly in their corresponding data or table templates.

Internal Business Entities (IBEs)

Internal Business Entities (IBEs) represent data structures that are internal to webMethods JI. IBEs can be thought of as the common data object or working objects within webMethods JI. XBEs must always be converted (copied) into an IBE for Method steps to work against; and likewise, IBEs must always be converted (copied) to XBEs when interacting with an external data source.

Creating Business Entities

webMethods JI allows you to create Business Entities in several ways:

- Within MapMaker, using the Business Entity Editor. The Business Entity Editor allows you to create IBEs, XSD XBEs and MLM XBEs.
- XSD XBEs can also be created by importing a sample XML document.
- Business Entities of different types can be created (i.e. copied) from an existing Business Entity type. For example, an existing IBE can be copied to an XSD XBE.

Business Entities can have relationships to each other, so that when one Business Entity is converted into another Business Entity, webMethods JI knows how to move the element values from one Business Entity to the other. These relationships can be defined either manually or automatically. These relationships can also be defined or overridden during Method execution.

We will use the following procedures to create the necessary Business Entities for our example application:

- "Creating an XSD XBE to Represent the Input XML Document" on page 71
- "Creating XSD XBEs to Represent the Output XML Document" on page 73

Creating an XSD XBE to Represent the Input XML Document

To create an XSD XBE to represent the input XML document:

1. Open the **Business Entity Editor** by clicking on the `Edit Business Entities` button.
2. Select the **External Data Definitions** tab and select its **XML/XSD** tab.
3. Click the **Add** button.

   A new XML/XSD XBE (**XSD_0**) is added to the **Structure Definition** list.

4 Right-click on the new entry, select **Rename** and rename the new XSD XBE to `AccountNumber`. The name of the XSD XBE defines the root element name of the XML document. In the case of this input XML document, it only consists of a root element (i.e. `AccountNumber`).

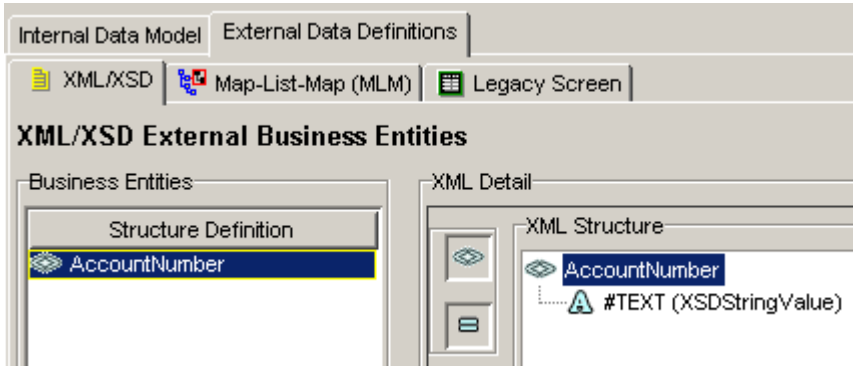

Figure 42. A New XML/XSD Business Entity

5 Since XBEs must always be converted (copied) into IBEs for Method steps to work against them, you will now create an IBE representing the **AccountNumber** XML/XSD XBE.

With the **AccountNumber** XML/XSD XBE selected, click the **Copy to** button.

The **Copy to** dialog box (Figure 43) is displayed.

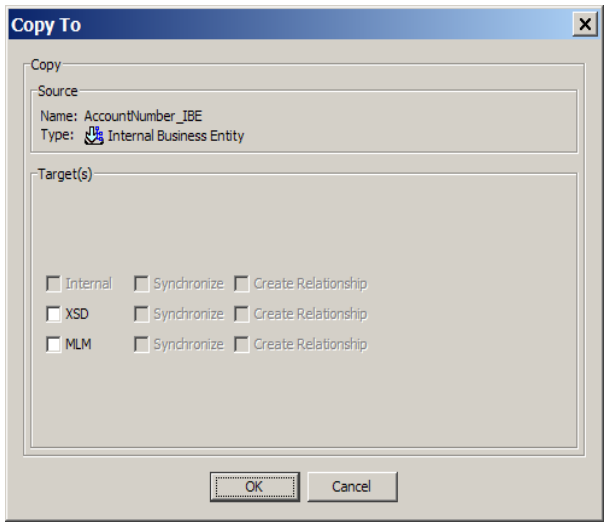

Figure 43. Copy To Dialog Box

The 'Copy to" operation allows you to create Business Entities of different types from an existing type. In this case, you are creating an IBE from the XML/XSD XBE.

In addition, the 'Copy to...' operation automatically defines the relationship between the source (e.g. **AccountNumber XML/XSD XBE**) and the target (e.g.

AccountNumber_IBE) Business Entities, using the **Synchronize** and **Create Relations** check boxes.

Synchronization enables you to maintain an element-per-element identity between the source BE and the copied BE(s), so that manually updating an element of the source BE automatically updates all corresponding elements in the synchronized, copied BEs.

Creating relations automatically creates one-to-one mapping relations between each element of the source BE and the corresponding elements of the copied BE(s). This saves you the need to define each relation manually, and enables information to be automatically transferred between the mapped elements.

6 In the **Target(s)** section, **Internal** is set by default and its **Synchronize** and **Create Relation** check boxes are automatically set.

Click **OK** to create the new IBE.

A new IBE named **AccountNumber_IBE** is added to the **Structure Definition** list of the **Internal Data Definitions** tab.

Creating XSD XBEs to Represent the Output XML Document

Next, you will create the Business Entities for handling the XML output requirement. For this you will construct the following Business Entities:

1 An IBE copied from the legacy XBE.

2 The corresponding XML/XSD XBE, created from the above IBE using a 'Copy to' operation.

> **Note:** There are several ways to create the required IBEs. The following steps illustrate one way to accomplish this, and also illustrate other capabilities of webMethods JI's data modeling.

Creating an IBE from the ttscrnLoanTransactions Legacy XBE The IBE is created by performing a 'Copy to' operation on the **ttscrnLoanTransactions** legacy XBE.

3 In the **Business Entity Editor**, select the **External Data Definitions** tab and choose its **Legacy Screen** tab.

4 In the **Structure Definition** list, select the **ttscrnLoanTransactions** XBE and click on the **Copy to…** button.

The **Copy to** dialog box is displayed (Figure 43).

> **Note:** While 'ttscrnLoanTransactions' is a simple structure containing three elements, this technique illustrates that existing BEs can be used not only to create BEs of a different type, but also as building blocks for more complex BEs.

5 Since **Internal** is already selected in the **Target(s)** section, just click **OK** to create the new IBE.

A new IBE named **ttscrnLoanTransactions_IBE** is added to the **Structure Definition** list of the **Internal Data Definitions** tab (located under the **Internal Data Model** tab).

Creating an IBE Structure Representing the LoanInformation Output XML Document The LoanInformation Output XML document (see "Example Output XML Document" on page 69) consists of four elements, which in turn have their own child elements.

Accordingly, you need to create an IBE structure that represents the LoanInformation document as a whole, and then add four IBE sub-structures corresponding to the above elements and child elements.

The following table lists all XML elements to be defined as IBEs:

Element	Child Elements	See...
<LoanInformation>	<AccountNumber> <Name> <Profile> <Transactions>	Step 6 on page 74
<AccountNumber>	Text	Step 7 on page 76
<Name>	Text	Step 7 on page 76
<Profile>	<BasePrincipal>, <Collateral>, <Term>	Step c on page 76
<Transactions>	<Amount>, <Date>, <Type>.	Step d on page 75

6 To create the IBE structure representing the LoanInformation XML document, proceed as follows:

a Select the **Internal Data Definitions** tab under the **Internal Data Model** tab.

Note: As you can see, you have already defined two IBEs: 'AccountNumber_IBE' and 'ttLoanTransactions_IBE'. Both of these IBEs were created using the 'Copy to…' operations you have performed above.

There is also a third IBE named 'ErrorReportType'.
This particular type of IBE is a system-level IBE, which is automatically populated when an error occurs within a Method. This structure can then be used by the onFail processing within the Method (see the description of the onFail link mode under "Linking Method Steps" on page 82).

b Click the **Add** button to create a new IBE.
The new IBE (**IBE_0**) is added to the **Structure Definition** list.

c Right click the new IBE, select **Rename** and enter the name `LoanInfo`.

d To create the IBE sub-structure representing the `Transactions` element of the XML document, proceed as follows:

e In the **Internal Data Definitions** tab, go to the **Structure Definition** list and select **LoanInfo**.

f In the **Structure** section, right click on **LoanInfo** and select **Add Structure Definition**. The **Choose a Type** dialog box opens. Select **ttscrnLoanTransactions_IBE** and click **OK**.
A new sub-structure branch (**sub_0**) is added to the **LoanInfo** structure.

The **sub_0** IBE sub-structure is based on the **ttscrnLoanTransactions_IBE** IBE you have copied from the **ttscrnLoanTransactions** Legacy XBE. Therefore, **sub_0** automatically includes all components defined for the Legacy XBE:

- **dfDate**
- **dfAmount**
- **dfType**

This saves you the trouble of creating this sub-structure manually.

Note: When you select 'sub_0', you may notice that its ''isArray' check box is automatically set in the **Info** section (located at the bottom of the **Structure** section).
An array is a repeating data type, for example: a table containing a repeating set of rows. Most data type elements may be set as arrays, with a few exceptions, such as a structure's root element.

In this case, the IBE in question was copied from a Legacy XBE representing a table template. webMethods JI registers that this IBE represents repeating data, but it does not set the IBE's **isArray** check box when it is a root element. However, once you add this IBE as a sub-element of another IBE, the **isArray** check box is automatically set.

For more information on arrays, see the webMethods JI User Guide.

Next, we will add the rest of the IBE structure elements (Step 7 and Step c) and then rename them so that they correspond to the names of the XML document elements (Step 8).

7 To create the IBE sub-structures representing the `AccountNumber` and `Name` elements of the XML document, proceed as follows:

a Make sure that **LoanInfo** is still selected in the **Structure** list.

b Click the **New InternalString** button twice, to add two String sub-structures (**sub_1** and **sub_2**) to the **LoanInfo** structure.

c To create the IBE sub-structure representing the `Profile` element of the XML document, proceed as follows:

d With **sub_2** selected in the **Structure** list, click the **New InternalStructure** button to add a new internal substructure (**sub_3**) to **LoanInfo**.

e With **sub_3** selected in the **Structure** list, click the **New InternalString** button three more times, to add three InternalString elements (**sub_0**, **sub_1** and **sub_2**) to the **sub_3** sub-structure of the **LoanInfo** structure.

Your IBE should now look like this (Figure 44):

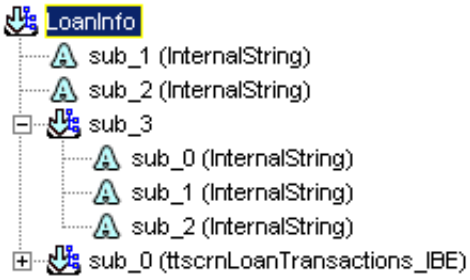

Figure 44. LoanInfo IBE Structure

Note: You can rearrange the structure's branches as needed, by dragging them to the appropriate location.

8 Now rename the IBE structure elements, so that they correspond to the names of the XML document elements (see "Example Output XML Document" on page 69). Enter the names specified in the following table, by right clicking each element, selecting **Rename**, entering the appropriate name and pressing the 'Enter' key.

Current Element Name	New Element Name
sub_1 under LoanInfo	AccountNumber
sub_2 under LoanInfo	Name
sub_3	Profile
sub_0 under Profile	BasePrincipal
sub_1 under Profile	Collateral
sub_2 under Profile	Term
sub_0	Transactions

Your IBE should now look like this (Figure 45):

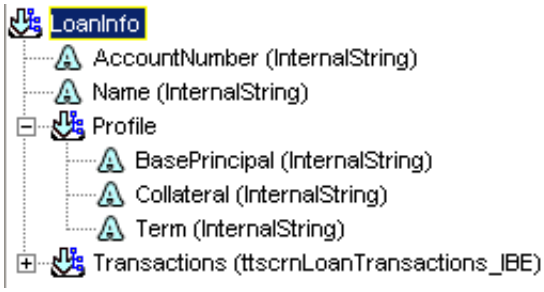

Figure 45. LoanInfo IBE Structure Partially Renamed

9 Expand the **Transactions** branch and notice that the elements are named **dfDate**, **dfAmount** and **dfType**, as defined in the table template and thus reflected in the Legacy XBE and the IBE copied from it.

Since the **Transactions** substructure was created from **ttscrnLoanTransactions_IBE**, the elements cannot be renamed here, but only in the **ttscrnLoanTransactions_IBE**.

Note: 'ttscrnLoanTransactions_IBE' is actually a Public data structure: other structures can use it, but any changes to the structure must be made directly in the 'ttscrnLoanTransactions_IBE'.
Similarly, any changes to 'ttscrnLoanTransactions_IBE' are automatically reflected in all the structures using it.

10 Select the **ttscrnLoanTransactions_IBE** in the **Structure Definition**, list and rename each element as follows:

Current Name	New Name
dfDate	Date
dfAmount	Amount
dfType	Type

11 Select **LoanInfo** again in the **Structure Definition** list and expand all branches. Your structure should now look like Figure 46.

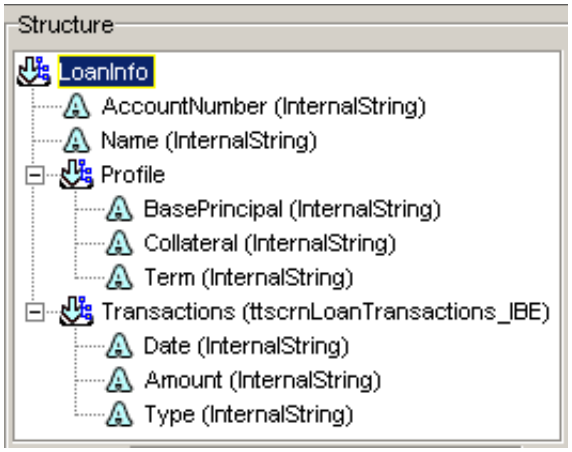

Figure 46. LoanInfo IBE Structure with Renamed Branches

12 Now you can create the XSD XBE from the LoanInfo IBE. With **LoanInfo** selected in the **Structure Definition** list, click on the **Copy to…** button.

The **Copy To** dialog box is displayed (Figure 47).

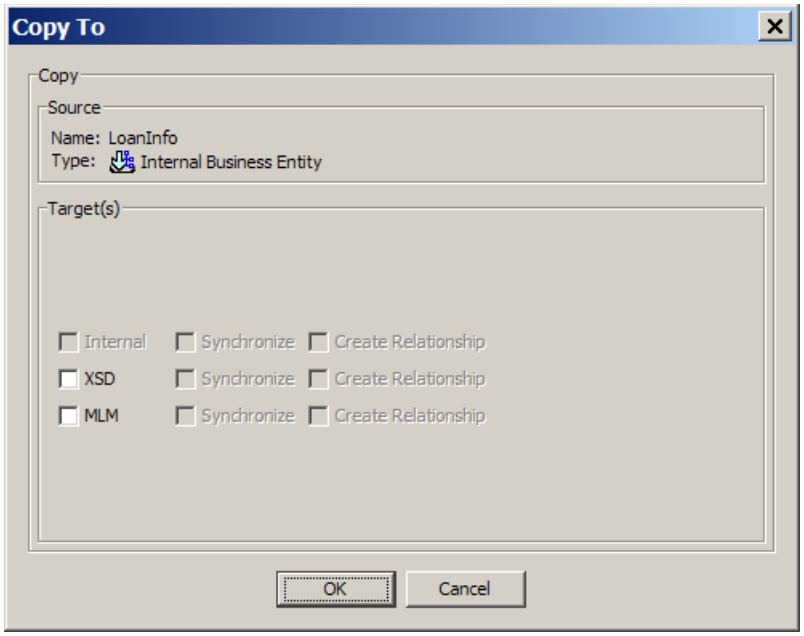

Figure 47. Copy To Dialog Box

13 Set the **XSD** check box (the **Synchronize** and **Create Relationship** boxes to its right are automatically set).

14 Click **OK**.

15 Select the **External Data Definitions** tab and then select the **XML/XSD** tab. Notice this tab now contains a new XSD XBE named **LoanInfo_XSD**.

16 Your XML document requirements are that the root node be named `LoanInformation`. Rename **LoanInfo_XSD** to `LoanInformation`.

17 Finally, we need to create an instance of the **LoanInfo** IBE that can be populated with data from the legacy system. Again, global variables are used to hold data within the map.
So far, you have defined global variables to hold simple data types, i.e. InternalStrings. Global variables can also be defined to hold complex data types, i.e. Business Entity structures.

To add a global variable, proceed as follows:

a Select the **Internal Data Model** tab and then select the **Global Variables** tab.

b Click **Add** to add a new global variable.
The **Choose a Type** dialog box (Figure 30) is displayed.

c Scroll down the list to choose **LoanInfo** and then click **OK**.
A new global variable (**gvLoanInfo**) is added to the **Instance** list.

18 You have finished defining all of the required Business Entities for now.

Close the **Business Entity Editor** by clicking on the **OK** button.

Defining a Method

Methods are logical groupings of screen transitions, navigational information, input and output variable definitions and data mappings. After Methods are defined, they are grouped together into one or more services that will execute the Method. Finally, the clients contact the relevant service and instruct it to execute the appropriate Methods. Methods are managed in the **Methods** tab.

When defining a Method, you do the following:

1. Add a new Method (by right-clicking the Tree's root and choosing **Add Method** from the pop-up menu).
2. Add a new step to the Method (see "Adding a Method Step" on page 80).
3. Configure the properties of the new step in the Properties view.
4. Link the new step to the other Method steps (see "Linking Method Steps" on page 82).
5. Repeat Step 2 to Step 4 as necessary, to include all required steps in the Method.
6. Finally, map the data used by the Method (see "Mapping the Data Used by the Method" on page 93).

Adding a Method Step

A Method is comprised of several Method steps. You can add steps to a Method either from the **Methods** Tree view or from the tool bar.

Adding a Method Step from the Methods Tree View

Proceed as follows:

1. Right click the **Steps** node in the Tree.
   A popup menu is displayed, listing all steps (Figure 48).

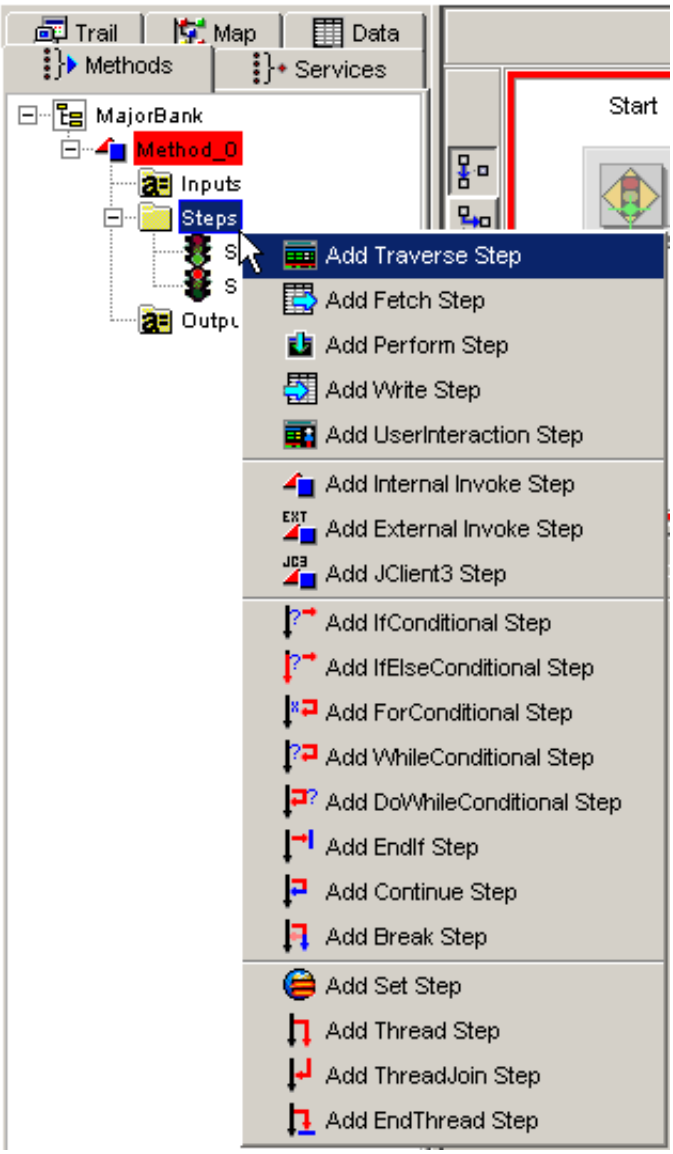

Figure 48. Method Steps Shortcut Menu

2 From the shortcut menu, select the type of step to be added.

Observe that the selected step is added both to the Presentation view and to the **Steps** node of the Methods Tree.

Adding a Method Step from the Method Steps Toolbar

Proceed as follows:

1 Locate the desired Method step icon in the tool bar:

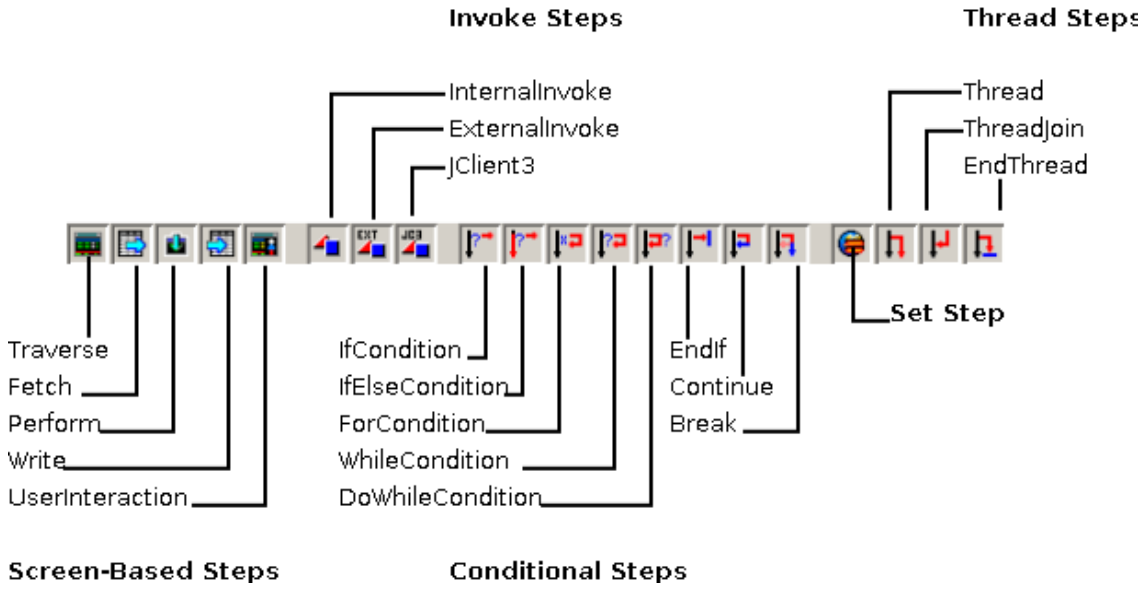

Figure 49. Method Step Toolbar

2 Left-click once on the icon, or drag and drop the icon into the Presentation view. Observe that the selected step is added both to the Presentation view and to the **Steps** node of the Methods Tree.

> **Note:** When a new step is created, it is added to the Tree view's **Steps** node, at the end of the list. To determine the actual order of the Method steps' execution, look at the graphical representation in the Presentation view.
>
> When screen(s) are removed from the map (due to changes in the 'Trail' and/or 'Map' tabs), any steps that rely on the removed screen(s) become invalid and must be corrected.

Linking Method Steps

The Method's steps are connected to each other manually, by drawing links between them. The following table describes the different types (modes) of links, as well as the Presentation view's display options that are included in the Link Modes tool bar

Button	Description

Next: The default link mode, which proceeds to the next step in the Method. This is the primary flow of the Method logic.

Branch: An alternative link mode, which diverges from a Conditional or Thread type of step.

- If the conditions specified in the step are true, the Method follows this **<branch>** link. When the Method reaches the end of the **<branch>** flow, it returns to the primary, **<next>** flow.
- If the conditions specified in the step are false, the Method skips the **<branch>** flow and follows the primary, **<next>** flow.

onFail: A backup link mode, used to handle situations in which a Method step fails and the host application is in an unknown state. An onFail link allows the service to proceed to another step that returns the legacy application to a known state.

For example, when a legacy application receives unexpected input, it might clear the screen and display an error. In this case, you can create an onFail link that returns the application to the main menu.

Auto Arrange: Aligns all connected Method steps in sequence. For Methods that do not contain conditional steps, the alignment is vertical.

Swap Content: Toggles between the Method flow view and the Presentation view of the highlighted step.

Note: The two views can also be toggled using the space key.

Multiple-Mode Links

A single link may contain up to three different logical link modes, one on top of the other. All types of steps may have “next” and “onFail” link modes, while Conditional and Thread steps may also have a “branch” link mode. Each link mode is indicated by a label.

When the mouse is moved over a label, both the label and the link associated with it are selected (i.e. bolded). When multiple edges appear on the same link, only one label can be selected at a time. For example, Figure 50 shows a multiple-edge link whose onFail label is selected.

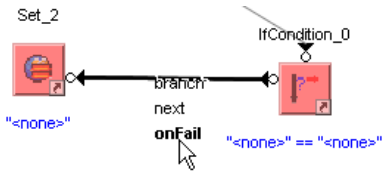

Figure 50. Multiple-Edge Link with onFail Label Selected

Double clicking the selected label (or its data mapping icon, if data mapping has already been configured for that link mode) opens its **Data Mapping Editor**, whose caption indicates the mode of the link and the names of the steps it connects. For example, double clicking the above onFail label (Figure 50) opens the **Data Mapping Editor** with the following caption (Figure 51):

Figure 51. Data Mapping Editor Caption

> **Note:** Dropping the link on a "blank" space or on an inappropriate destination step (e.g. trying to drag a "branch" link from a step that is not conditional) causes the linking operation to fail.

xWhen multiple modes appear on the same link, but no particular label is selected (Figure 52), double clicking the link itself automatically opens the **Data Mapping Editor** for the link mode with the highest precedence available, in the following order:

1. next
2. branch
3. onFail

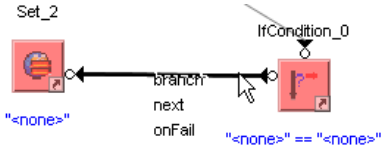

Figure 52. Selecting a Multiple-Mode Link without Choosing a Specific Mode

Adding Links

Link lines are drawn in the following manner:

1. In the Link Modes tool bar, select the link mode you wish to create (next , branch or onFail).
2. Click and hold the left mouse button over the "link arrow" of the origin step (see Figure 53), and drag a link to the destination step.

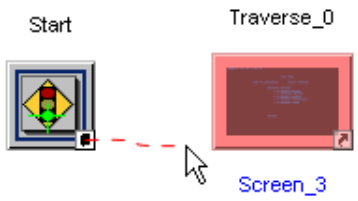

Figure 53. Linking Method Steps

> **Note:** Dropping the link on a "blank" space or on an inappropriate destination step (e.g. trying to drag a "branch" link from a step that is not conditional) causes the linking operation to fail

3 If there are multiple options available for the target of the link, a "Chose a..." dialog (e.g. **Choose a Destination** or **Choose a Template**) is displayed.

In this example, the **Choose a Destination** dialog is displayed for the Traverse step (Figure 54).

Select the appropriate destination step and click **OK**.

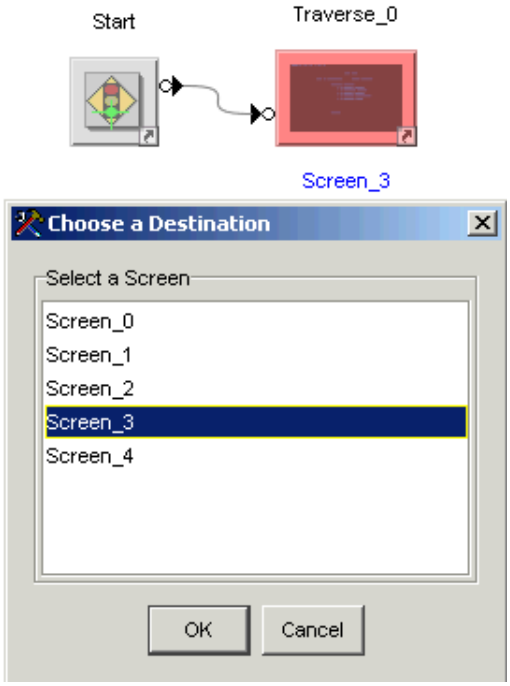

Figure 54. Choose a Destination Dialog Box

Once the two steps are properly linked, the following changes take place (Figure 55):

- The color of the destination step changes, in this case from red to green (for information on the color code, see "Methods Tree and Presentation View Color Code" on page 86).
- The mode of the link (next, branch or onFail) is indicated by a label.

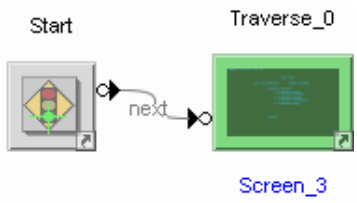

Figure 55. Linked Steps

Figure 56 shows an example Method, whose steps are all properly linked. This Method consists of three flows:

- The “next” flow, which is the primary flow.
- The “branch” flow, followed when the conditions specified in the **IfCondition_0** step are true.
- The “onFail” flow, followed when the **IfCondition_0** step cannot be executed.

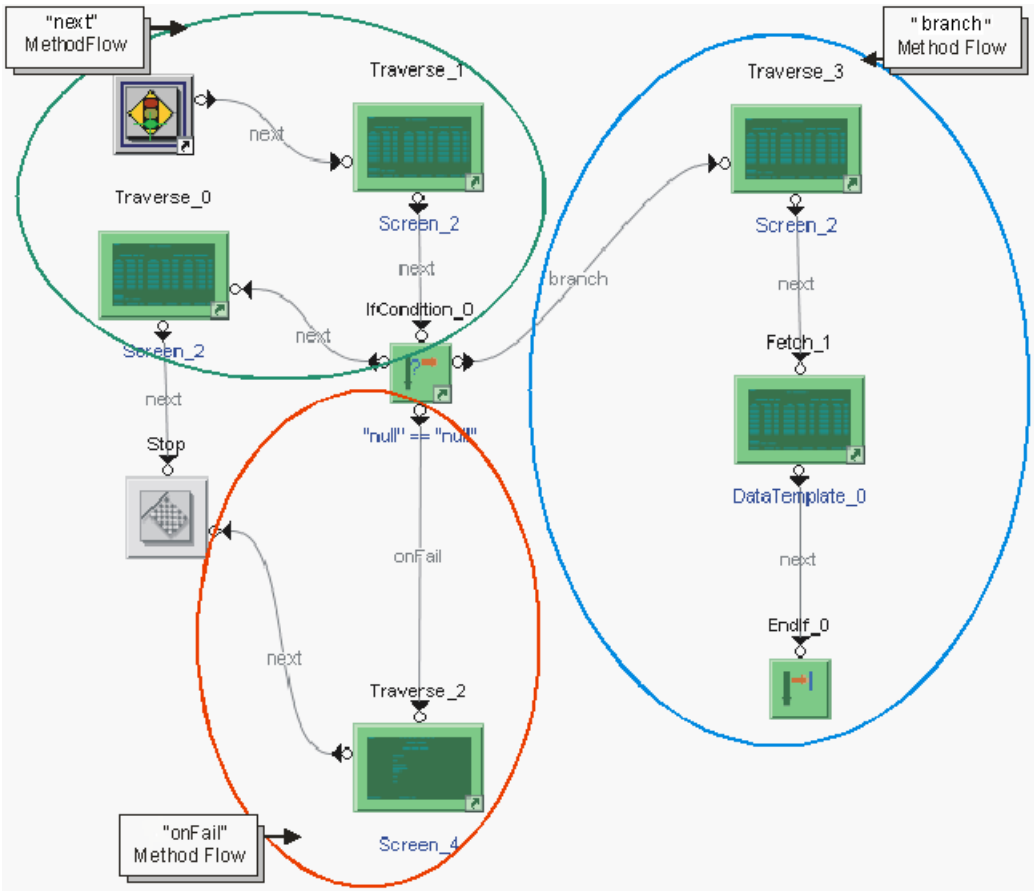

Figure 56. A Method with Three Flows: next, branch and onFail

Methods Tree and Presentation View Color Code

The Tree view and Presentation view provide a convenient visual indication of the Method’s validity, using the following color code:

- Red — some of the Method's components (steps, links etc.) are invalid. Examples of invalid items include:
  - Unlinked items.
  - Items without valid and/or required properties set. For example:
    - JClient3 steps whose host:port, service name, or Method name properties are not set.
    - Screen-based steps, such as Traverse, without a valid screen set.
- Yellow — some of the Method's components (steps, links etc.) are incomplete. An example incomplete item is a step that has no "Next" or "onFail" link.
- Green — the Method is properly configured.

When items are invalid or incomplete, they are colored red or yellow (respectively) in the following places:

- Tree view — highlighting the Method and its invalid steps.
- Presentation view — framing the invalid steps, as well as the whole pane.

In addition, the status bar at the bottom of the MapMaker window provides an error message that describes the problem (e.g. "Fetch_3 has no NEXT node...").

When the problems are fixed, the Tree view is no longer colored and the steps and frame of the Presentation view are colored green.

Searching for Objects in the Methods Tab

As you work in the **Methods** tab, you can easily search for a particular Method, input variable, output variable or step, by right-clicking the root of the Tree and choosing **Find...** from the pop-up menu.

The **Search for Object** dialog box is displayed, allowing you to enter the name of the object of interest (Figure 57).

Figure 57. Search for Object Dialog Box

> **Note:** The search is case-sensitive and requires the object's full name (no wild cards are allowed).

When the object is found, it is highlighted in the Tree view. Methods and steps are also highlighted in the Presentation view.

Defining Our First Method

The first Method you will define will log into the host application and advance us to the Main Menu screen. The Main Menu screen will become your Home screen or idle screen, i.e. the screen to which you will return after each transaction.

To define the Method:

1. Select the **Methods** tab.
2. Right-click the root of the Tree view (in this case, the **MajorBank** node) and select **Add Method** from the pop-up menu.
   A new Method is added to the Methods Tree.
3. Right-click the Method, select **Rename** and enter `mthLogon` as the name of the Method.
4. Click the "+" icon to expand the Method. Three sub-components are automatically included with the Method:
   - **Inputs**: Input variables receive input from the client. Input variables have two views of the data: internal and external.
   - **Steps**: Method steps represent the actual activity performed by the Method, such as screen traversal, inputting variable data into fields and extracting returned data from fields.
   - **Outputs**: Output variables send output back to the client. Just like input variables, output variables also have two views of the data: Internal and External.
5. Now, you will create a step to log on to the application. A single `Traverse` step will suffice for this Method. `Traverse` steps instruct the Method to traverse from the current screen to another screen.

   From the Steps toolbar, drag a `Traverse` step ( ) onto the Presentation view.

> **Note:** This step is currently red, indicating that its properties and links have not been set.

6. Link the new Traverse step (**Traverse_0**) to the **Start** step, by drawing a **Next** link from the Start step to the Traverse step. The **Choose a Destination** dialog box appears (Figure 58).

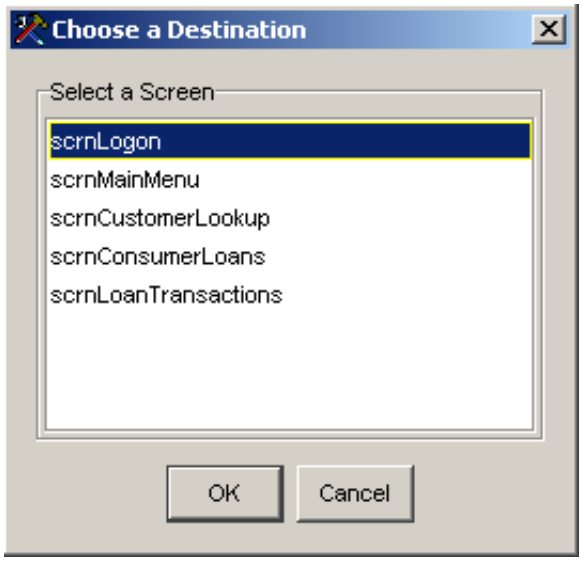

Figure 58. Choose a Destination Dialog Box

7 Select **scrnMainMenu** as the destination screen to which the **Traverse_0** step will navigate, and click **OK** to set the destination.

> **Note:** The color of the Method, the 'Traverse_0' step and the Presentation view frame changes from red to yellow. This indicates that the properties have been set (i.e. a destination screen has been selected), but that a link is missing.

8 Draw a `Next` link from the **Traverse_0** step to the **Stop** step.

> **Note:** The color of the Method, the 'Traverse_0' step and the Presentation view frame changes from yellow to green, indicating that the method is now fully configured.

This Method is now complete.

9 Create another Method, this time to log off:

Right-click the **MajorBank** node and select **Add Method** from the pop-up menu. A new Method is added to the Methods Tree.

10 Right-click the new Method, select **Rename** from the pop-up menu and enter `mthLogoff` as the name of the Method.

11 From the Steps toolbar, drag a `Traverse` step ( ) onto the Presentation view.

12 Link the new step (**Traverse_1**) to the **Start** step. Draw a `Next` link from the **Start** step to the **Traverse** step.

The **Choose a Destination** dialog box appears (Figure 58).

13 Select **scrnLogon** as the destination screen to which **Traverse_1** will navigate, and click **OK** to set the destination.

14 Draw a "next" link from **Traverse_1** to the **Stop** step.

This Method is now complete.

Add Method - mthGetLoanInfo

Follow this process:

1 Add a new Method by right-clicking the **MajorBank** node and selecting **Add Method** from the pop-up menu.

2 Right-click the Method, select **Rename**, and enter `mthGetLoanInfo` as the name of the Method.

3 Click on the "+" to expand the **mthGetLoanInfo** Method.

4 Create an input variable as follows:

   a Right click on the **Inputs** node, select **Add Input** and rename the new input variable (**InVar_0**) `LoanAccountNumber`.

   b Input variables have two views of the data: an *internal* view and an *external* view. The internal view is described by an Internal Business Entity (IBE), and the external view is described by an External Business Entity (XBE).

   For this **LoanAccountNumber** input variable, the internal view of the data will be **AccountNumber_IBE**, which you have created from the **AccountNumber** XSD XBE via the 'Copy to' operation (see Step 5 on page 72).

   The external view of the data (i.e. the input from the client) will be an XML document, represented by the above **AccountNumber** XSD XBE that you have created earlier (see "Creating Business Entities" on page 71).

   To define the internal and external views of the data, select the

LoanAccountNumber input variable in the Tree, thereby displaying its properties in the **Properties** view.

c In the **Input Type Selection** section, set the **Internal Type** property to **AccountNumber_IBE** (Figure 59).

d In the same section, set the **External Type** property to the **AccountNumber** XSD.

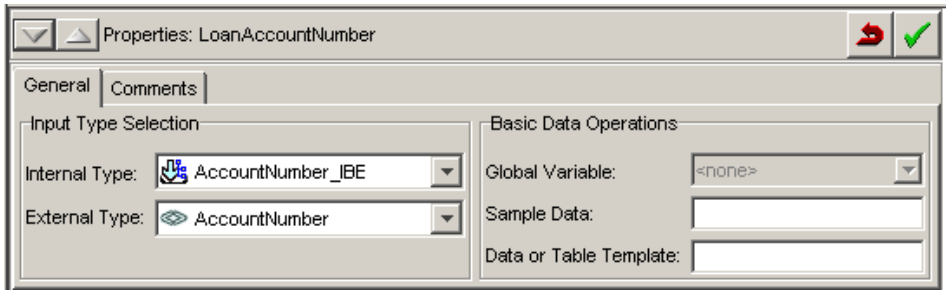

Figure 59. LoanAccountNumber Input Variable Properties

e Accept the changes to the input variable properties by clicking on the `Apply` button ✓.

5 Create an Output variable as follows:

a Right click on the **Outputs** node, select **Add Output** and rename the new output variable (**OutputVar_0**) to `LoanInfo`.

b Output variables, like input variables, have two views of the data: an internal view, described by an IBE; and an external view, described by an XBE.

For this output variable, the output and input will be represented by two BEs which you have already created earlier: the internal data view will be the **LoanInfo** IBE (see Step 6on page 74); and the output to the client (i.e. the external data view) will be an XML document, represented by the **LoanInformation** XSD XBE (see Step 12 on page 78).

To define the internal and external views of the data, select the **LoanInfo**

output variable in the Tree, thereby displaying its properties in the **Properties** view.

c In the **Output Type Selection** section, set the **Internal Type** property to **LoanInfo** (Figure 60).

d In the same section, set the **External Type** property to **LoanInformation**.

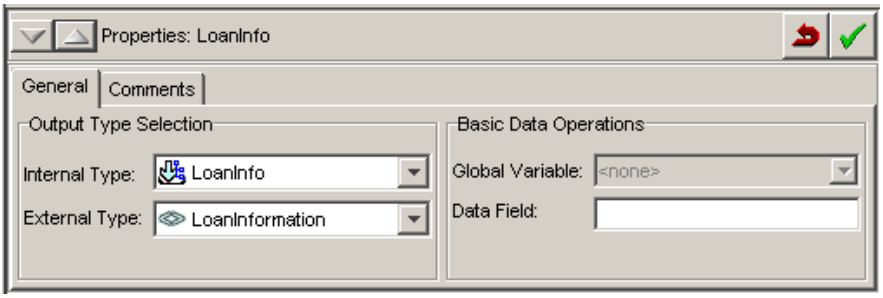

Figure 60. LoanInfo Output Variable Properties

e Accept the changes to the output variable properties by clicking on the **Apply** button .

6 Now you will add steps that will navigate through the loan screens and fetch the data and table templates you have defined.

Add a `Traverse` step to the Method as follows:

a From the Steps toolbar, drag a `Traverse` step ( ) onto the Presentation view.

b Draw a `Next` link from the **Start** step to the **Traverse_2** step.
The **Choose a Destination** dialog is displayed.

c Choose **scrnConsumerLoans** and click **OK**.

7 Add a `Fetch` step to the Method as follows:

a From the Steps toolbar, drag a `Fetch` step ( ) onto the Presentation view.

b Draw a `Next` link from the **Traverse_2** step to the **Fetch_0** step.
The **dtLoanDetails** data template is automatically selected by the **Fetch_0** step, as it is the only template defined on the **scrnConsumerLoans** screen.

8 Add a `Perform` step to the Method as follows:

a From the Steps toolbar, drag a `Perform` step ( ) onto the Presentation view.

b Draw a `Next` link from the **Fetch_0** step to the **Perform_0** step.
The **toscrnLoanTransactions** action is automatically selected by the **Perform_0** step, as it is the only action defined on the **scrnConsumerLoans** screen.

9 Add another `Fetch` step to the Method as follows:

a From the Steps toolbar, drag a `Fetch` step ( ) onto the Presentation view.

b Draw a `Next` link from the **Perform_0** step to this new **Fetch_1** step. This time, the **Choose a Template** dialog is displayed, allowing you to choose the data template or table template to be fetched.

> **Note:** In this case, the 'Choose a Template' dialog is displayed since scrnLoanTransactions (as opposed to scrnConsumerLoans) has more than one template defined.

c Select **ttscrnLoanTransactions** and click on the **OK** button.

10 Complete the Method flow by drawing a `Next` link from **Fetch_1** to **Stop**. The Method is now colored green, indicating it is properly defined (Figure 61).

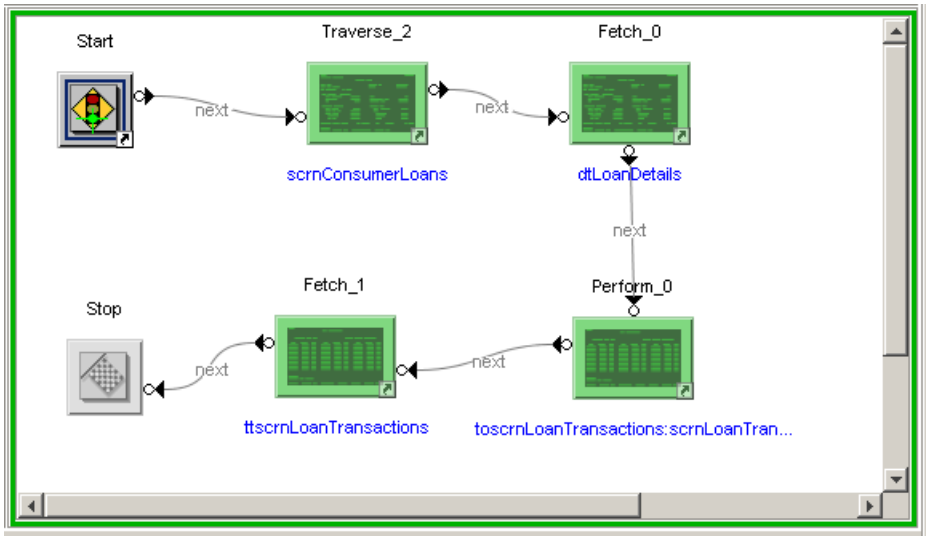

Figure 61. The mthGetLoanInfo Method

Mapping the Data Used by the Method

Now that the flow of the Method has been defined, the last step is to map the data used by the Method. The **mthGetLoanInfo** Method requires the following:

- Receiving data from the input variable (**LoanAccountNumber**).
- Storing the data obtained from the host application in global variables (**gvAccountNumber** and **gvLoanInfo**).

> **Note:** Data can also be stored in Method variables, which are specific to individual methods and inaccessible from other methods in a map. For more information, see
> Chapter 8, "Methods" in the webMethods JI User Guide.

- Sending data back to the client via the output variable (**LoanInfo**).

Data mapping allows you to define where data is stored or retrieved by various operations within the Method. Data mappings are defined on the links between the Method steps.

For this **mthGetLoanInfo** Method, we need to define mappings on the following links:

- The link between the **Start** step and the **Traverse_2** step, where you store the account number in the **gvAccountNumber** global variable.
- The link between the **Fetch_0** step and the **Perform_0** step, where you store the loan details (the account number, name, base principal, collateral and term) in the **gvLoanInfo** global variable.
- The link between the **Fetch_1** step and the **Stop** step, where you store the loan transactions data (the date, amount and type) in the **gvLoanInfo** global variable; and then map **gvLoanInfo** to the **LoanInfo** output variable.

To define data mapping on a link, double click it to display its **Data Mapping Editor**.

> **Note:** There is no need to map data from Variables to a target, since the **Traverse** step is not a consumer of data.

xStoring the Account Number in the gvAccountNumber Global Variable

Follow this process:

1. Double-click on the **next** link between the **Start** step and the **Traverse_2** step. The **Data Mapping Editor** is displayed.

   The **Data Mapping Editor** allows you to do the following:

   - In the **Standard** tab:
     - Map data from the **Source** to **Variables**.
     - Map data from **Variables** to the **Target** (if the step is a consumer of data).
   - In the **Variables** tab:
     - Map data from a **Source** variable to a **Target** variable.

   In this case, all you need to do is map the input variable **LoanAccountNumber**, which is the **Source** containing the account number, into the global variable **gvAccountNumber**.

> **Note:** There is no need to map data from Variables to a target, since the **Traverse** step is not a consumer of data.

2. In the **Source** list, expand the **LoanAccountNumber** input variable and select the **_TEXT** node. This is the node containing the account number from the

XML document (see "Creating an XSD XBE to Represent the Input XML Document" on page 71).

3 In the **Variables** list, select **gvAccountNumber**.

> **Note:** Once you select two object that can be mapped to each other, the **Create Data Map** button is enabled.

4 Click the **Create Data Map** button, to create a mapping between the two selected objects (Figure 62).

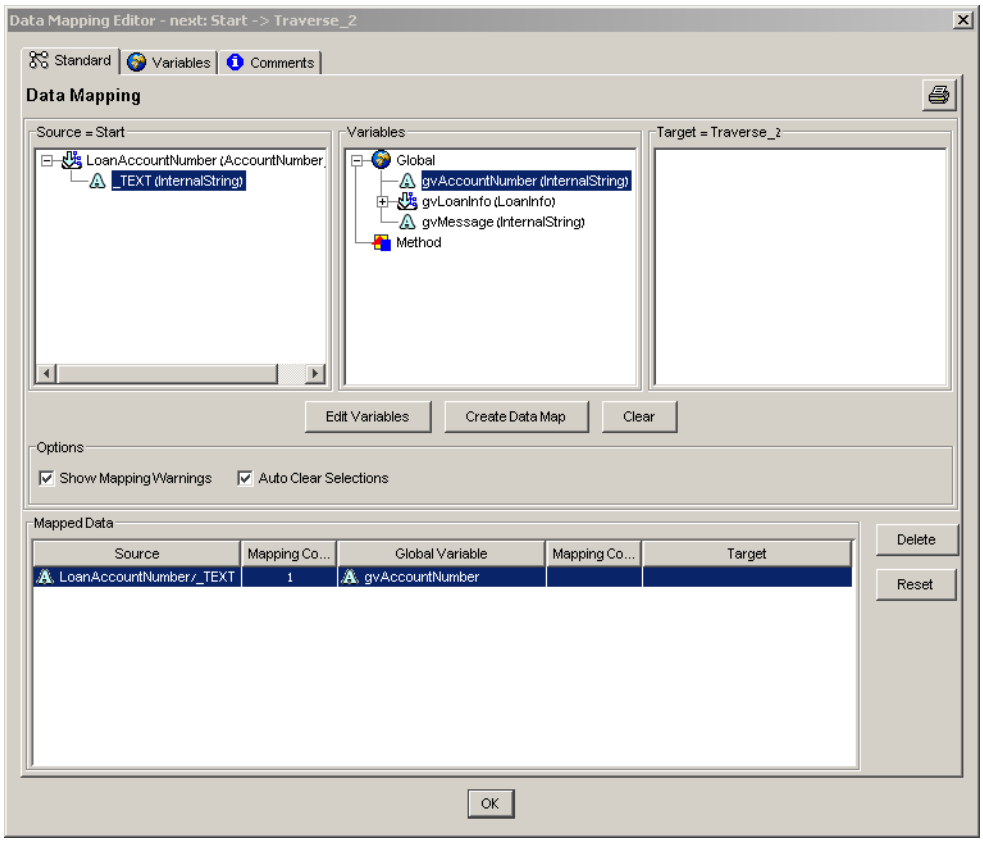

Figure 62. Data Mapping Editor > Mapping the _TEXT Source to the gvAccountNumber Global Variable

When this link is executed, the account number from the XML document will be copied to the global variable **gvAccountNumber**.

5 Close the **Data Mapping Editor** by clicking on the **OK** button.

A data mapping icon is added to the link between the **Start** step and the **Traverse_2** step (Figure 63).

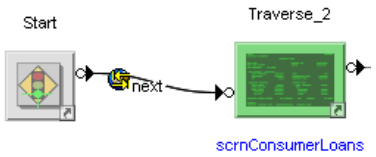

Figure 63. A Data Mapping Icon on a Link Between Two Steps

Storing the Loan Details in the gvLoanInfo Global Variable

The next data mapping that needs to be defined is on the link from the **Fetch_0** step to the **Perform_0** step. The **Fetch_0** step returns legacy system data that need to be stored in a global variable. In this case, the data contained in the `dtLoanInfo` data template (name, account number, etc.) need to be stored in the `gvLoanInfo` global variable.

`gvLoanInfo` can hold a complex data structure, since its type is the LoanInfo IBE (see Step 17 on page 79), which represents the output XML document (see "Creating an IBE Structure Representing the LoanInformation Output XML Document" on page 74).

Proceed as follows:

6 Double-click on the **next** link between the first **Fetch_0** step and the **Perform_0** step.

The **Data Mapping Editor** is displayed.

7 In the **Data Mapping Editor**, expand the following entities:

- In the **Source** list, expand **dtLoanDetails**.
- In the **Variables** list, expand **gvLoanInfo**.
- Under **gvLoanInfo**, expand **Profile**.

8 Select **dfName** under **dtLoanDetails** and **Name** under **gvLoanInfo**, and click on the **Create Data Map** button to create the data mapping.

9 Repeat the instructions in Step 8 to create data mappings between the following pairs of sources and global variables:

Source (dtLoanDetails)	Global Variable (gvLoanInfo)
dfAccountNumber	AccountNumber
dfBasePrincipal	BasePrincipal
dfCollateral	Collateral
dfTerm	Term

The data mapping between the entities is now complete (Figure 64).

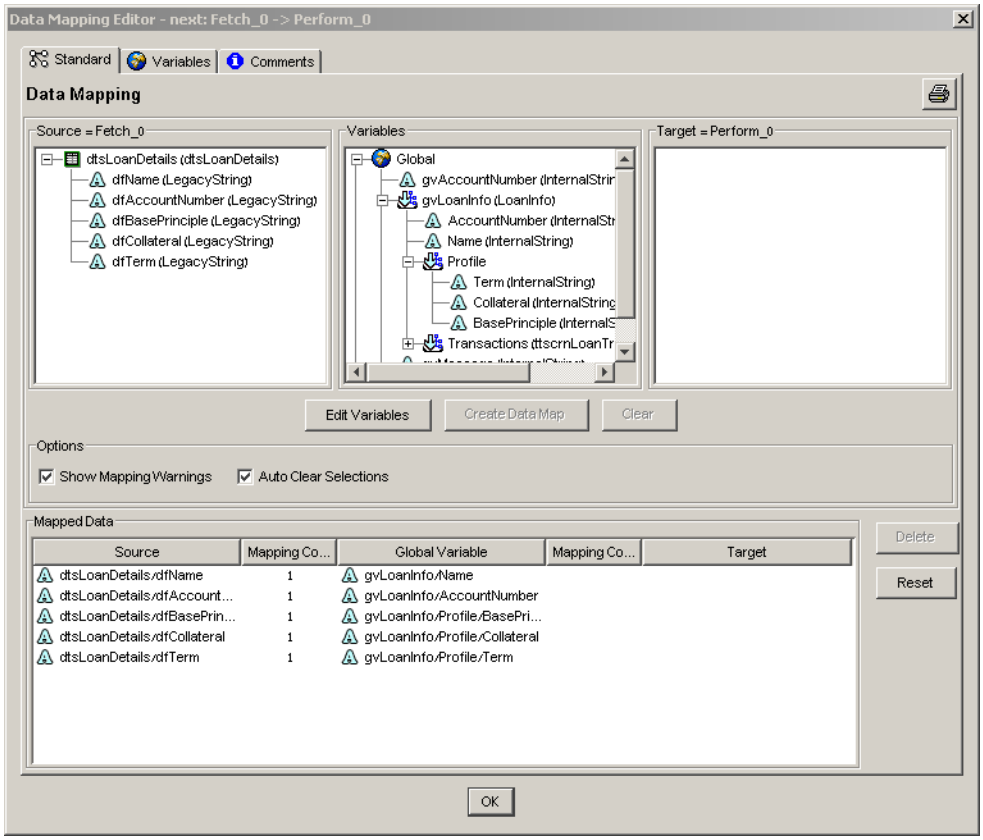

Figure 64. Data Mapping Editor > Mapping the dfLoanDetails Source to the corresponding gvLoanInfo Global Variables

10 Close the **Data Mapping Editor** by clicking on **OK**.
A data mapping icon is added to the **next** link between the **Fetch_0** step and the **Perform_0** step.

Storing the Loan Transactions Data in the gvLoanInfo Global Variable

The final mapping for this Method is on the link from the **Fetch_1** step to the **Stop** step. Proceed as follows:

1 Double-click on the **next** link between the **Fetch_1** step to the **Stop** step.
The **Data Mapping Editor** is displayed.

> **Note:** This time, the 'Source', 'Variables' and 'Target' lists all have contents. This is because the 'Fetch_1' step (i.e. the Source) is producing data, and the 'Stop' step (i.e. the Target) is consuming data.

2 Select **ttscrnLoanTransactions** in the **Source** list.

3 In the **Variables** list, expand **gvLoanInfo** and select **Transactions**.

4 Click on the **Create Data Map** button to create the mapping.

This mapping operation copies the **dfDate**, **dfAmount** and **dfType** elements of **ttscrnLoanTransactions** to the corresponding **Date**, **Amount** and **Type** elements of **Transactions**. This mapping is possible because the **Transaction** substructure of **gvLoanInfo** was created from the **ttscrnLoanTransactions_IBE**, which in turn was created from **ttscrnLoanTransactions** using a 'Copy to' operation. The 'Copy to' operation creates pre-defined mappings between the two objects (Figure 65).

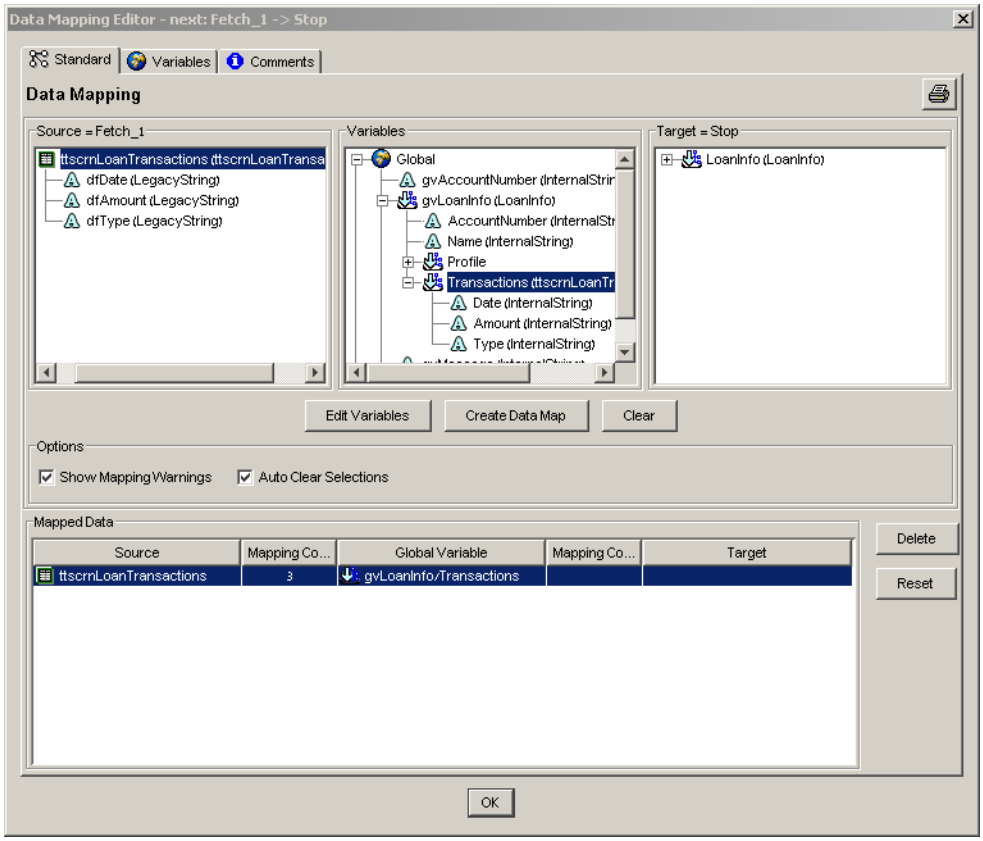

Figure 65. Data Mapping Editor > Mapping the ttLoanTransactions Source to the Transactions Global Variable

Mapping gvLoanInfo to the LoanInfo Output Variable

Finally, you need to map **gvLoanInfo** into the output variable **LoanInfo**. Proceed as follows:

5 Select **gvLoanInfo** in the **Variables** list and **LoanInfo** in the **Target** list.

6 Click on the **Create Data Map** button.

This mapping operation copies each element from **gvLoanInfo** into the corresponding element of the output variable **LoanInfo**. This mapping is possible because the internal types of both **gvLoanInfo**'s and **LoanInfo** are set to the **LoanInfo** IBE (see Figure 66).

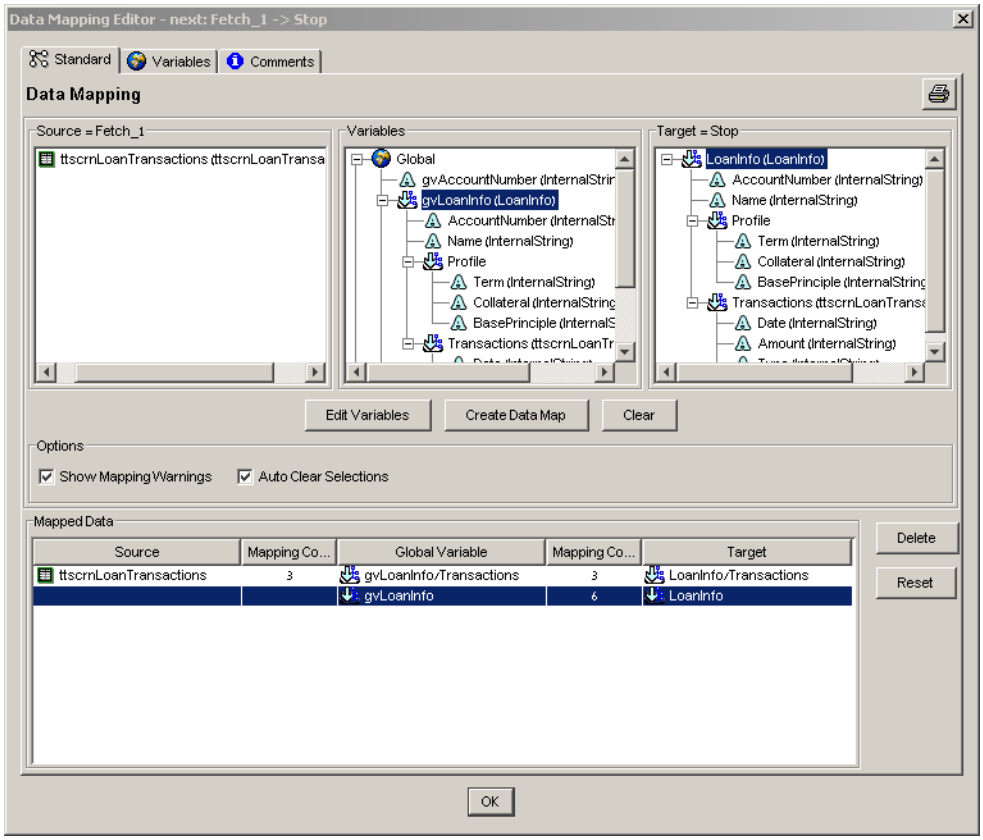

Figure 66. Data Mapping Editor > Mapping the gvLoanInfo Global Variable to the LoanInfo Target

7 Close the **Data Mapping Editor** by clicking on the **OK** button.

You have just completed the definition of the **mthGetLoanInfo** Method.

Defining the Service

Next, we will create a service and include our Methods in the new service.

Create the service

1 Click on the **Services** tab to access the **Services** panel.

2 In the **Services** panel, right-click the **MajorBank** node and select **Add Service**.

3 Right-click the new service, select **Rename** and enter `MajorBankSvc` as the name of the service.

4 **Add Methods to the service**

5 Right-click the **MajorBankSvc** service and select **Add Method to Service**. The **Choose a Method** dialog box opens (Figure 67).

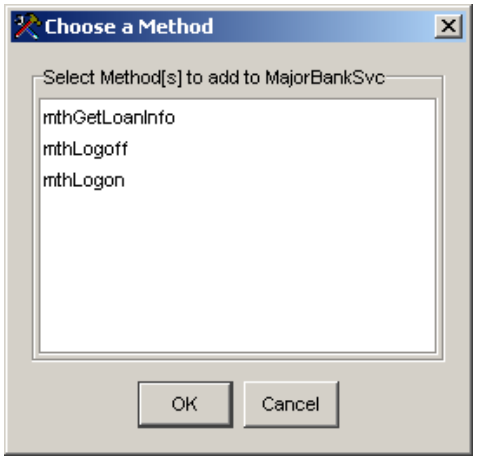

Figure 67. Choose a Method Dialog Box

6 Select all three Methods (multi-select by holding down the **Shift** key while selecting) and click **OK**.

The three Methods are added to the **MajorBankSvc** node of the Tree.

Set the Service Properties

The service **Properties** view allows you set a home screen, so that after each client request webMethods JI will return to the specified screen. In our case, we should return to the main menu screen (`scrnMainMenu`) after each client request.

You can also specify Methods to be run when the service initializes and finalizes. The following steps define `mthLogon` as the Initialize Method and `mthLogoff` as the Finalize Method.

1 In the **Services** tab, select the **MajorBankSvc** node to display the service properties in the **Properties** view (Figure 68).

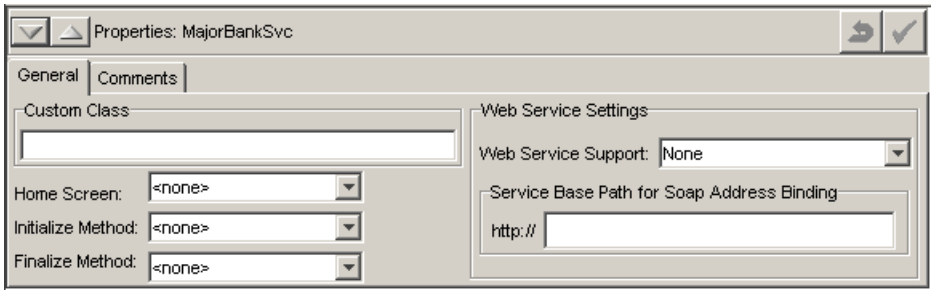

Figure 68. Service Properties Pane

2 In the **Properties** pane, make the following selections:

Property	Selection
Home Screen	`scrnMainMenu`

Property	Selection
Initialize Method	`mthLogon`
Finalize Method	`mthLogoff`

3 Click **Apply** ✓.

Save the Map • Select **File** > **Save**.

Generating the Service and Testing the Client Code

The next step is to generate the service code. Proceed as follows:

1 Select **Tools** > **Generate** > **Generate All Code**.
The **Generate Code** dialog box is displayed (Figure 69).

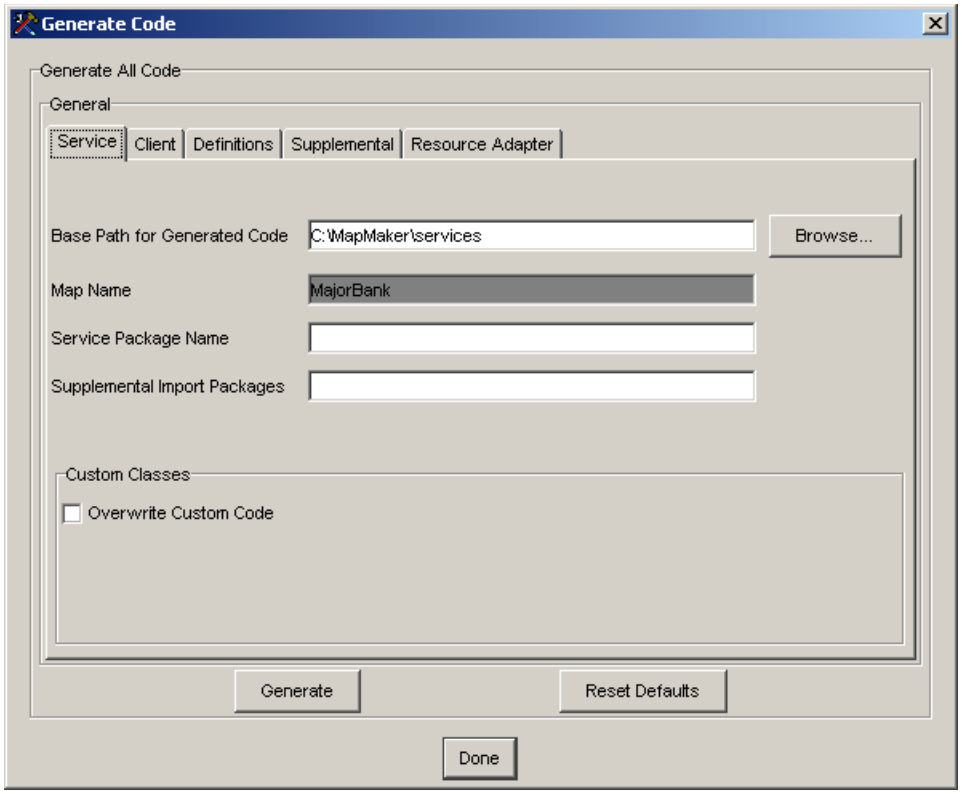

Figure 69. Generate Code Dialog Box

The **Generate Code** dialog box offers the following options:

Option	Description
Base Path for Generated Code	The root directory (and drive letter, if applicable) where the generated code is saved. When the code is generated, the actual location of the code is determined based on this directory, plus the directory structure specified in the **Service Package Name** option (described below). For example, if the base path is: *C:\MapMaker\services* and the **Service Package Name** is: *com.jacada.MajorBankSvc*, the actual location of the generated code is: `C:\MapMaker\services\com\jacada\MajorBankSvc.`

Option	Description
Map Name	The name of the map file. This field is completed automatically based on the name of the map (e.g. MajorBank), and cannot be edited.
Service Package Name	The name of the package to which the generated files belong. A Java package is a collection of java class files that are grouped together to form a logical set of classes. For the purposes of this tutorial, enter `com.jacada.MajorBankSvc` as the package name.
Supplemental Import Packages	The packages that are imported into the service. To import additional packages, include them here using a semicolon (`;`), colon (`:`) or comma (,) to delineate each package.
Overwrite Custom Code	When checked, any previously generated and modified custom code is overwritten with the new stub code.

2. Enter the required information described in the above table, and click the **Generate** button.

   If you have checked **Overwrite Custom Code**, an alert message is displayed. Click **OK** to Continue.
3. The code is automatically generated and compiled, as indicated by a **Progress...** message.

   Next, a **Success!!** message is displayed, indicating that the operation has been successfully completed.

   Click **OK** to close the **Success!!** message box.
4. Click **Done** to close the **Generate Code** dialog box.

Starting Your webMethods JI Server Environment

The webMethods JI runtime server environment consists of three different components that must be started prior to testing your generated service:

- Resource Database
- Resource Server

- Environment Managers

For a detailed description of the webMethods JI runtime environment, see the *webMethods JI User Guide*.

The webMethods JI Environment is started using the `ea_start` command. Execute the following at your command prompt:

```
cd <JI_install_dir>\bin
ea_start
```

> **Note:** These instructions assume that the EA_ENV environment variable is set to the location of your webMethods JI configuration files, and that the correct parameters are set in the environment.ccf file. For more information on ea_start, see the webMethods JI Installation and Configuration Guide or the webMethods JI User Guide.

Deploying the Service

The Service code generated, as described in "Generating the Service and Testing the Client Code" on page 101, is compiled as a `jar` file. This `jar` file can either be deployed to your webMethods JI Resource Database, or stored on your local file system.

Deploying the service in the database provides a number of benefits, including a central location for `jar` and `map` file storage and cross-platform compatibility (as you will see when we configure the service, storing the service on the file system rather than in the database requires identifying the location of the service in a platform-specific manner). Therefore, if you intend to use load balancing between Environment Managers in both UNIX and Windows environments, you must store the service in the database.

> **Note:** In order to deploy to your database, your Resource Database and your Resource Server must be running. For instructions on starting these components of the webMethods JI server environment, see "Starting Your webMethods JI Server Environment" on page 103.

To deploy the service, proceed as follows:

1. Select **Tools** > **Deploy Map**.
   The **Deploy to Resource Server** dialog box opens (Figure 70).

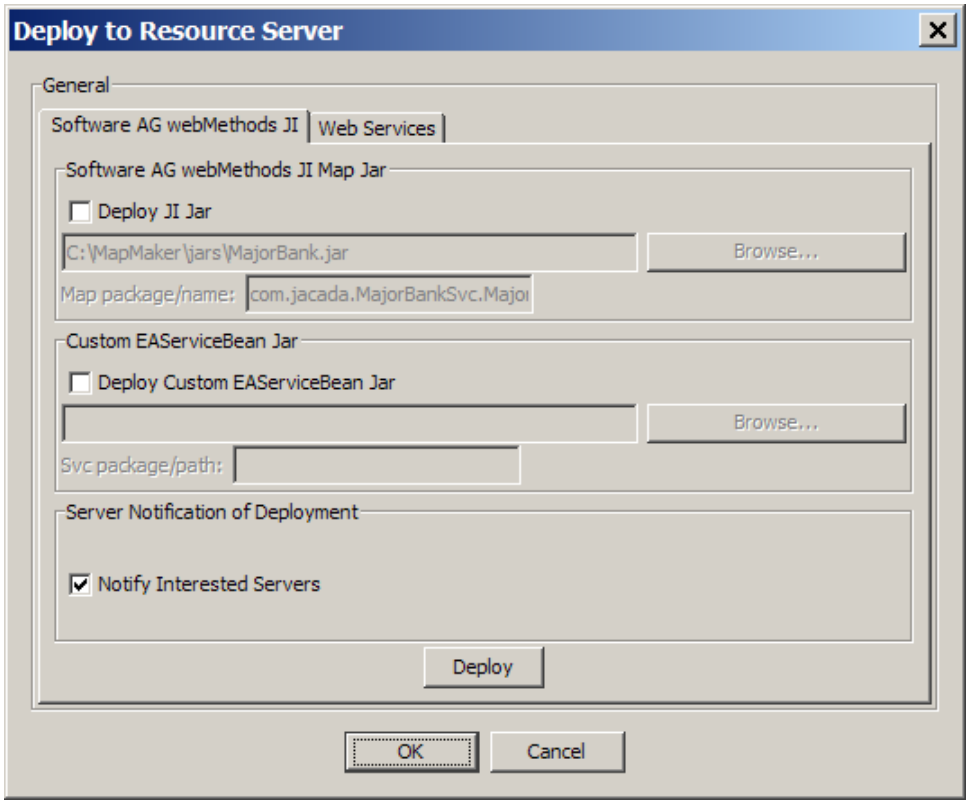

Figure 70. Deploy to Resource Server Dialog Box

2 Set the **Deploy webMethods JI Jar** check box and click **Deploy**.

Depending on the settings in the **Servers** tab of the MapMaker **Properties** dialog box, the **Choose a Resource Server** dialog box may be displayed, requiring that you select a Resource Server from the list of all **Resource Servers** located using multicasting.

3 Next, the **Choose a Database** dialog box is displayed.

Select a Resource Database from the list and click **OK**.

4 If this service had already been deployed before, the **webMethods JI Map Jar** dialog box is displayed. A drop-down list includes the version numbers you can apply to the service when it is stored in the Resource Database.

Select the latest number available and click **OK**.

5 An **Information** message is displayed, indicating that the service has been successfully deployed.

Click **OK** to close the message.

6 Click **OK** to close the **Deploy to Resource Server** dialog box.

Confirming the Service Configuration

Next, we need to verify the service configuration. This configuration is done automatically when a service is deployed to the Resource Server database. It is good practice to confirm that the configuration was done correctly and completely, even though the default settings are the desired settings.

Service configuration is performed in two places in the Configuration Manager:

- Service Masters set the options for services throughout an entire environment.
- Service Details are then used to override some of the Service Master settings for specific Environment Manager Configurations (a Configuration is a subset of a Group and represents the load balancing parameters for one or more Environment Managers).

Starting the Configuration Manager

To start the Configuration Manager, proceed as follows:

1 Enter the following in a DOS prompt:

```
cd <JI_install_dir>\bin
ea_cfgmgr
```

Note: If Windows shortcuts were added to the Start Menu during the webMethods JI installation, the Configuration Manager can be started by selecting Start>Programs>webMethods JI 4.5>ConfigurationManager.

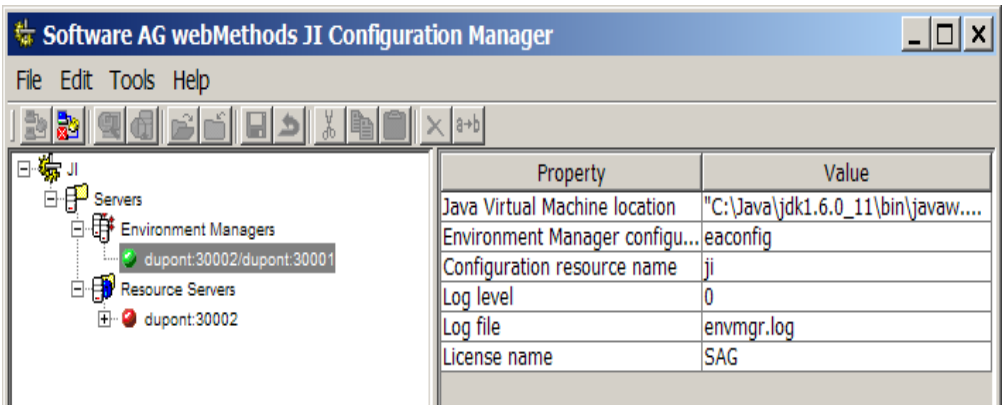

Figure 71. The Configuration Manager

About the Configuration Manager User Interface

The Configuration Manager interface consists of two primary panes or views:

- The Tree view
- The Property view

Tree View

The left pane of the Configuration Manager interface is the Tree view. The Tree view is a graphical representation of hierarchical information, consisting of "root" components that can be expanded to expose underlying sub-components (also called "nodes" or "branches"). These sub-components often can be expanded as well, to reveal their own sub-components, creating a hierarchical tree format. A "+/-" symbol before each component in the tree identifies if the node can be expanded to reveal additional, lower-level nodes. A "+" indicates that the node can be expanded and a "-" indicates that the node is already expanded. The absence of a "+" or "-" indicates that the node cannot be expanded.

Property View

The right pane of the Configuration Manager interface is the Property view. This view is used to display the properties and values for various configuration parameters. For example, when you are editing a Resource Database, the Property view displays all of the properties associated with the database object and their associated values. You can edit the values of each record in this view.

Locating Your Resource Server

Follow this process:

1 The first step to configure the service is to locate your Resource Server.
   - If your Resource Server is on the local machine or on another machine within the local sub-net, webMethods JI's multicast technology automatically locates the Resource Server. In this case, your Resource Server will be listed as subordinate to the Resource Server node. Connect to the Resource Server by highlighting it and selecting the **Connect** button ( ).
   - If your Resource Server is not on the local sub-net, you need to use the **Locate** dialog box to locate the server. Highlight the Resource Server node and select **File** > **Locate** and enter the host name and the port of the Resource Server.

> **Note:** When you use this Method to locate a Resource Server, you will be automatically connected to the Resource Server.

2 After locating and connecting to a Resource Server, the next step is to open the database being served by the server. The resources served by this server,

such as Resource Databases and licensing, are listed as subordinate to the Resource Server.

Highlight the appropriate Resource Database and select **File > Open datasource**. The Configuration Manager's Tree view will look like Figure 72.

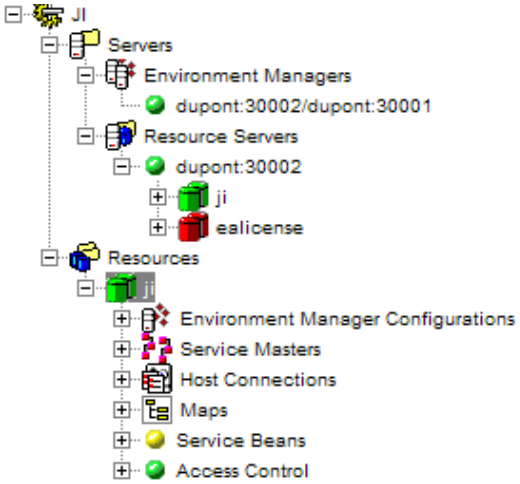

Figure 72. The Configuration Manager Tree

3 Next, we will verify that the information for the Service Master Record that was created for this service is correct. Expand the **Service Masters** node of the Resource Database, and highlight the Service Master Record **MajorBankSvc**.

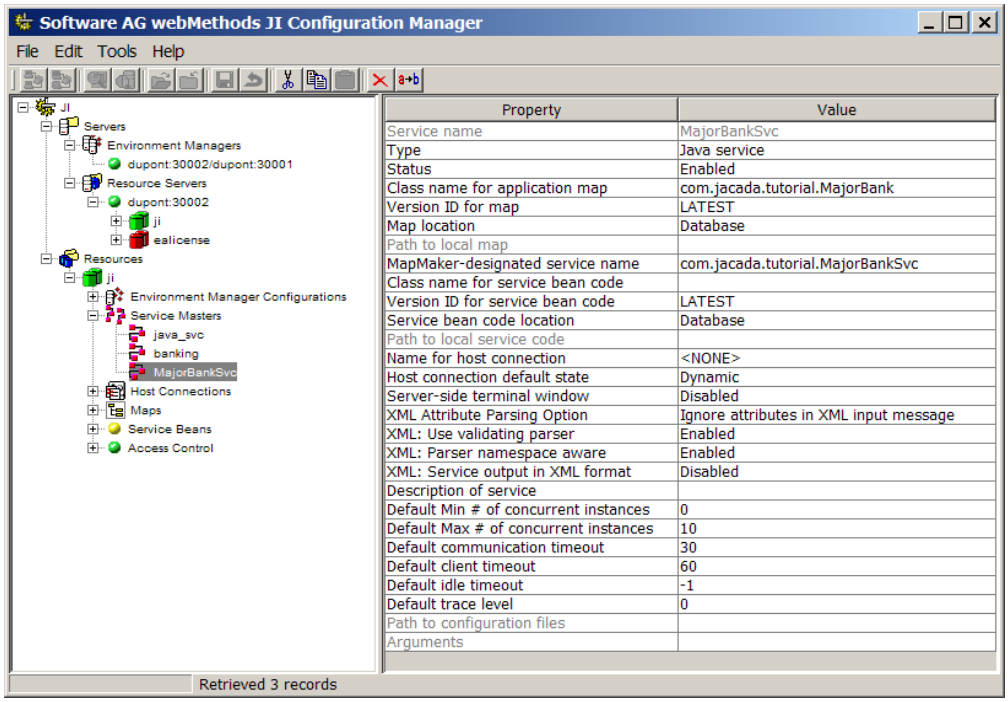

Figure 73. Expanded Service Masters Node

Confirm the following information for the Service Master and make any necessary changes:

Note: Items marked with an asterisk (*) must be changed to the appropriate value.

Property	Value
Service name	`MajorBankSvc` (completed by default)
Type	`Java service`
Status	`Enabled`

Property	Value
Class name for application map	The class name of the map that was created with MapMaker. For this tutorial, this should be set to `com.jacada.MajorBankSvc.MajorBank` **Note:** This is the full qualified map name that includes the package name (com.jacada.MajorBankSvc). This name can also be found by looking at the deployed maps under the maps node in the Configuration Manager.
Version ID for map	`LATEST`
Map location	`Database`
Path to local map	This field is not accessible when Database is selected as the map location.
MapMaker-designated service name	The class name of the service that was generated in MapMaker. For this tutorial, this should be set to: `com.jacada.MajorBankSvc.MajorBankSvc.` **Note:** This is the full qualified service name. It is the name of the service as defined in MapMaker and the package name assigned in the service code.
Class name for service bean code	Leave blank
Version ID for service bean code	LATEST
Service bean code location	Database
Path to local service code	This field is not accessible when Database is selected as the service code location.

Property	Value
Name for host connection	Leave blank (set to <None>)
*Host connection default state	`Pooled`
*Server-side terminal window	Creates a window on the server machine that displays an active Presentation view including all of the interactions between the service and the legacy host. To create a server-side terminal window, select `Enabled`.
XML Attribute Parsing Option	Ignore attributes in XML input message.
XML: Use validating parser	Enabled
XML: Parser name space aware	Enabled
XML: Service output in XML format	Disabled
*Description of service	`Major Bank Tutorial`
*Default Minimum number of concurrent instances	1
Default Maximum number of concurrent instances	10
Default communication timeout	30
Default client timeout	60
Default idle timeout	-1

Property	Value
*Default trace level	The default level on which information for this service is logged to the log file. Valid entries are `0` through `9`, with `0` representing no logging and `9` representing the most verbose logging. This value can be overridden on the command line at startup. Set this property to 9.
Path to configuration files	Not applicable.
Arguments	Not applicable.

4 If you have made changes, right-click **MajorBankSvc** and select **Save and Apply** from the shortcut menu.

   You may now close the Configuration Manager.

Testing the Service

To test the service, we will use a Java client that is automatically generated when the service code is generated. The client code, `MajorBankSvc_jclient3.java`, is located in the *clients* sub-directory of the directory in which the service code was generated. For example, in this tutorial we have designated the following directories:

- The **Base Path for Generated Code**, set in the **Properties** dialog box's **Default Directories** tab, is *C:\MapMaker\services* (see "Designating the Tutorial Directories" on page 38).
- The **Service Package Name**, specified in the **Generate Code** dialog box, is *com.jacada.MajorBankSvc* (see Figure 69).

Accordingly, the clients will be saved to the following directory: *C:\MapMaker\services\com\jacada\MajorBankSvc\clients.*

Testing the service consists of the following operations:

1 "Editing the Java Client Code (MajorBankSvc_jclient3.java)" on page 113
2 "Compiling the Java Client Code" on page 113
3 "Running the Test Client Code" on page 114

Editing the Java Client Code (MajorBankSvc_jclient3.java)

Follow this process:

1. Open the `MajorBankSvc_jclient3.java` file in a text editor such as Notepad.
2. Look for a section titled:

   `Map buildMap(String eaMethod) {`

   This section contains the statements that build the input data.
   Notice that this section contains the XML document to be sent from the client (i.e. <AccountNumber></AccountNumber>).
   To pass an account number in the XML document, insert 1234567898 into the XML document (i.e. <AccountNumber>1234567898</AccountNumber>).
3. Next, you will see an `if` statement for `mthGetLoanInfo`.

   The following `map.put` statement may be contained within comments, depending on the options that are set for test client generation.

   Remove the /* preceding and the */ following the map.put statement if needed.

   The completed section should look something like this:

```
// Create a Map object
Map map = new OrderedMap();

// Build the XML for the AccountNumber Business Entity
String AccountNumber_XML =
 "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" +
 "<AccountNumber>1234567898</AccountNumber>";

if (eaMethod.equals("mthGetLoanInfo")) {
    map.put("LoanAccountNumber", AccountNumber_XML);
}
```

4. Save the file.

Compiling the Java Client Code

Follow this process:

1. Create a batch file containing the following command:

```
<jdk_dir>\bin\javac -classpath .;<JI_install_dir>\lib\jclient3.jar
MajorBankSvc_jclient3.java
```

Note: <jdk_dir> represents the path (and drive letter, if applicable) to your JDK installation, and <JI_install_dir> represents the path (and drive letter, if applicable) to your webMethods JI installation.

For example, in Windows the following command may be used:

```
C:\jdk\bin\javac -classpath .;C:\JI\lib\jclient3.jar
MajorBankSvc_jclient3.java
```

2 Save the file as `compile.bat` in the `clients` sub-directory of the service code generation directory (e.g. `C:\MapMaker\services\com\jacada\MajorBankSvc\clients`).

3 At the command prompt, run the compile.bat file from the service code generation directory.

Running the Test Client Code

Follow this process:

1 Create a batch file containing the following command:

```
<jdk_dir>\bin\java -classpath .;<JI_install_dir>\lib\jclient3.jar
MajorBankSvc_jclient3
```

Note: <jdk_dir> represents the path (and drive letter, if applicable) to your JDK installation, <JI_install_dir> represents the path (and drive letter, if applicable) to your webMethods JI installation.

For example, in Windows the following command may be used:

```
C:\jdk\bin\java -classpath .;C:\JI\lib\jclient3.jar MajorBankSvc_jclient3
```

2 Save the file as run.bat in the clients sub-directory of the directory in which the service code was generated (e.g. *C:\MapMaker\services\com\jacada\MajorBankSvc\clients*).

3 At the command prompt, run the `run.bat` file from the service code generation directory.

Note: If the deployed service contains a User Interaction step, the “green screen” appears when the relevant Method contained in the run.bat file is run. You can position your command prompt window and the “green screen” window of the host application so that both can be viewed. This way, when you run the test client, you will see the host application being driven by your Method. The Method can be run several times if desired, so that you can observe the host application being navigated via the “green screen” window.

The service output consists of a single, long string, representing the XML document:

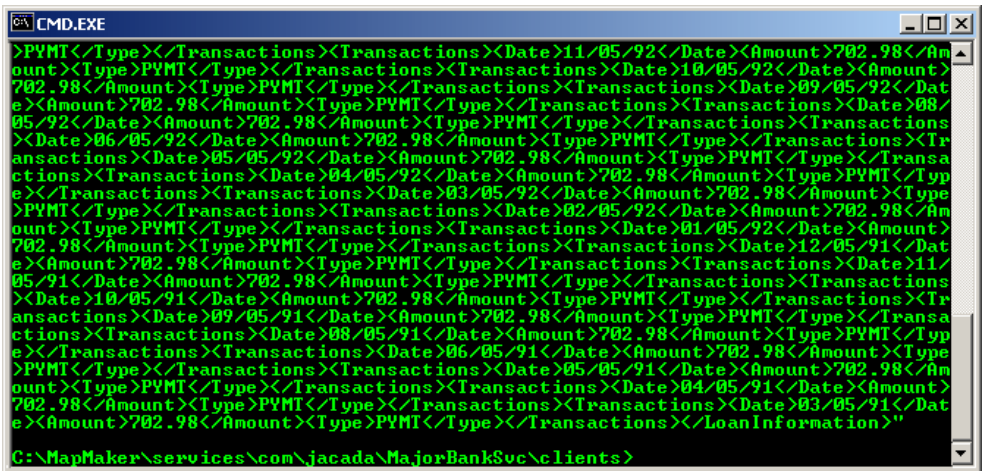

Figure 74. Service Output in XML

The following example shows a formatted output of the `mthGetLoanInfo` method.

Example 7 - 1 Formatted Output of the mthGetLoanInfo method

```
<LoanInformation>
    <AccountNumber>1234567898</AccountNumber>
    <Name>PAUL AGEE</Name>
    <Profile>
        <BasePrincipal>50000.00</BasePrincipal>
        <Collateral>75000.00</Collateral>
        <Term>120</Term>
    </Profile>
    <Transactions>
        <Date>12/08/97</Date>
        <Amount>702.98</Amount>
        <Type>PYMT</Type>
```

```
    </Transactions>
    <Transactions>
        <Date>05/05/97</Date>
        <Amount>770.30</Amount>
        <Type>LCHG</Type>
    </Transactions>
</LoanInformation>"
```

Summary of this Chapter

You have now completed the steps to successfully create and test Methods and services using MapMaker and webMethods JI.

In this chapter you have learned how to:

- Trail an application by recording the screens required to implement a given transaction requirement within the host application.
- Map the input fields to use data passed in from a client.
- Define the data to be returned from individual screens.
- Create Methods to perform the required transactions. This includes:
  - Creating data structures using the Business Entity Editor.
  - Creating data mappings within the Method using the Data Mapping Editor
- Create and deploy a service containing the defined Methods.

The techniques you have used in the chapter are the basic techniques to you will use to create additional transactions and functionality within MapMaker and webMethods JI.

New Terms Introduced in this Chapter

Fields: Fields within a host screen. During the mapping process, every group of related characters is marked and saved by MapMaker as a field.

Tags: In MapMaker, tags are used to distinguish between different instances of screens.

Actions: During trail recording, every action performed on the host is marked and saved by MapMaker.

Snapshots: Each screen navigated through during trail recording is saved by MapMaker as a snapshot. This is regardless of whether the screen is new or not.

Data Template: A logical representation of non-repeating data fields on the legacy screen. Data templates are defined in MapMaker and are used to extract data from the legacy application. Similar to a table template, used to define repeating data fields.

Table Template: A logical representation of the area on a legacy screen that contains repeating Data Fields. Table templates are defined in MapMaker and are used to extract data from the legacy application. Similar to a Data Template, used to define non-repeating Data Fields.

Global Actions: Actions defined and assigned in MapMaker, which are used as a default mechanism that will always return the application from an unknown state to a known state.

Global Variables: Variables defined in MapMaker, which are used to define variable inputs that will be used in the webMethods JI service. Global variables are added to a map and assigned to Action components in the map. For example, the text representing the user ID and password can be replaced with variables, which are used to fill the fields for the purpose of navigating to the next screen.

Internal Business Entities: Deal with data that is used internally within the JI Service. Internal Business Entities are also used for the creation of Global Variables. Also known as Internal Data Types.

External Business Entities: Deal with data that is sent to or received from an entity which is external to the JI Service, such as a legacy screen and a client. Also known as External Data Types.

Data Mappings: A means for instructing a Method, at any given time, to transfer data from one data type to another. Within the flow of a Method, there are two or more instances where data mapping takes place. Data mapping is defined in MapMaker's Data Mapping Editor.

Initialize Method: The Method which is triggered automatically upon entering the application.

Finalize Method: The Method which is triggered automatically upon leaving the application.

Code Generation: The process of generating Service and Client code in MapMaker.

Code Deployment: The process of deploying a webMethods JI service.

webMethods JI Configuration Manager: A tool provided with webMethods JI that is used to configure components in the webMethods JI environment.

Chapter 6. Advanced MapMaker Techniques

About this Chapter

This chapter will introduce you to some of the advanced features and techniques available within MapMaker. In this chapter you will learn how to reuse Methods that you have already created.

You will be introduced to advanced table techniques, such as record lookup. You will also learn how to use conditional logic steps to control the flow of a Method based on client and/or host data values. And finally, you will learn how to use XPath statements to access and manipulate data structures.

The workflow described in this chapter guides you through the following advanced procedures:

- "Advanced Table Techniques" - Provide the steps necessary to use table templates and data templates for account lookup operations.
- Reusing a Previously Created Method - Provides the steps necessary to make use of a previously defined Method.
- Conditional Logic Operations - Provide the steps necessary to use conditional logic to control the execution flow of a Method.
- XPath - Describes how to use XPath statements to manipulate or query data structures.

Note: For this section of the tutorial, we will continue using the map that we created in the previous chapters. But first, you need to save the existing MajorBank.map using a new name. In the event of an error, this map will serve as a backup, preserving the Methods and services created earlier in the tutorial. Once saved, close the map, and proceed to the next section.

Advanced Table Techniques

You will now create a new Method that uses a social security number to access a screen containing account numbers for the various accounts a customer may have. Using this screen, you will automatically obtain the loan account number and get the related loan information for that account. Since these new screens are not part of the current map, a new trail containing them must be created.

Record a New Trail

To record a new trail:

1 Open the original `MajorBank.map` file in MapMaker.

> **Note:** Trail recording will not begin until we have already navigated to the 'Main Menu' screen. While the trail could be started at the logon screen, this technique can be used to isolate new host functionally within its own trail.

2 In the Tree view, select the **Trail** tab.

3 Select **File** > **Connect** from the menu, or click on the `Connect Host` button .

   The **Connect to Host** dialog is displayed (Figure 19).

4 Select **MajorBank** from the **Select Host** drop-down list and click **OK**.

   The opening, logon screen of the `MajorBankHost.map` file is displayed in the MapMaker Presentation view (Figure 20).

5 To log into the application, type `jacada` in the **User id** field and press the **Tab** key to advance to the **Password** field.

6 In the **Password** field, enter `jacada` and press the **Enter** key.
   This will advance the tutorial banking application to the **Main Menu** screen (Figure 21).

7 Select **Map** > **Start Trail** or click on the `Start Trail` button .

   A new Trail (**Trail_1**) is added to the Tree.

8 At the prompt in the **Main Menu**, type **4** and press the `Enter` key to advance to the **Retail Customer Information** screen.

9 At the **SOC SEC #** prompt, type `987654321` and press **Enter**.

   The account information for the social security number entered is displayed.

10 Press `PF9` to return to the **Main Menu** screen.

11 This is the final screen in our second trail. Stop the trail recording mode by selecting **Map** > **Stop Trail** or clicking on the `Stop Trail` button .

12 Disconnect from the host by selecting **File** > **Disconnect** from the menu, or clicking on the `Disconnect Host` button .

Reconcile New Screens

To reconcile new screens:

1 Update the map by selecting **Map** > **Update Map** from the menu, or clicking on the `Update Map` button .

   A dialog box is displayed, stating that the map has been updated.

2 Click **OK**.

3 Switch to the Map panel by clicking the **Map** tab in the Tree view.

   Observe that two new screens (**Screen_0** and **Screen_1**) have been added to the Map.

4 Rename **Screen_0** by right clicking it, selecting **Rename** and entering `scrnCustomerAccountsLookup` as the new name.

5 Rename **Screen_1** to `scrnCustomerAccounts` by following the same procedure.

6 Add a global variable representing the Social Security Number (SSN) as follows:

   a Open the **Business Entity Editor** by clicking on the button.

   b In the **Global Variables** tab, click the **Add** button to display the **Choose a Type** dialog box.

   c Select **InternalString** and click **OK**.
   A new global variable (**gvString**) is added to the **Instance** list of the **Global Variables** tab.

   d Rename the new global variable by right-clicking on it, selecting **Rename** from the pop-up menu and entering the name `gvSSN`.

   e Close the **Business Entity Editor** by clicking on the **OK** button.

7 In the Tree view, expand the **scrnCustomerAccountsLookup** node and then expand its **Actions** node.

   This node includes a single action, **tocrnCustomerAccounts**.

8 Select the **toscrnCustomerAccounts** action to display its properties in the **Properties** view.

9 In the **Properties** view, select the **Action Input** tab.

   This tab allows you to replace the recorded SSN (**987654321**) with the global variable you have created (**gvSSN**).

10 In the **Data Prefix** field, clear the recorded SSN (**987654321***)*.

11 In the **Keystroke Input Operations** section, highlight **At 251: <987654321>**.

12 Set the **Data Source** field to gvSSN using one of the following Methods:

   - Manually enter `gvSSN`, *Or*
   - Click on the ellipsis button to display the **Choose a Data Source** dialog box, select **gvSSN** from the **Global Variables** list and click **OK**.

13 Click the **Apply** button to apply the changes.

Note: For a detailed description of the procedure linking a global variable to an Action, review the section “Linking Global Variables to Actions” on page 60.

Define the Table Template

To define the table template:

1. Select the **Data** tab and highlight the **scrnCustomerAccounts** screen in the Tree view.

   The corresponding snapshot (**Retail Customer Information**) is displayed in the Presentation view.

2. Create a table template consisting of the **ACCOUNT** and **TYPE** columns as follows:

   a. Select the table by clicking in the upper left corner under the **ACCOUNT** column, and dragging to enclose the table in a red box.
   The box should extend down to the line just above the **COMMAND** line, as shown in Figure 75.

   b. When the red box is positioned properly, right-click and select **Add Table Template**.

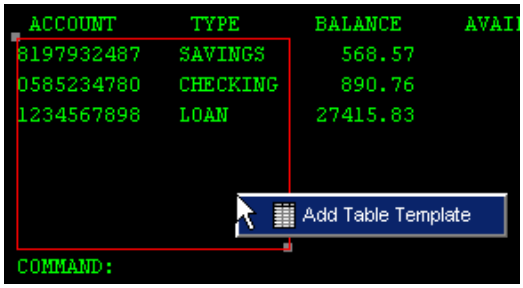

Figure 75. Defining the Customer Accounts Table Template

A new table template (**ttscrnCustomerAccounts**) is added to the **scrnCustomerAccounts** screen.

3. In the Presentation view, add the data fields to the new table template as follows:

   a. Double click the first value in the **ACCOUNT** column, and then right-click it and select **Add DataField**.
   A new data field (**DataField_0**) is added to the table template in the Tree view.

   b. Double click the first value in the **TYPE** column, and then right-click it and select **Add DataField**.

A new data field (**DataField_1**) is added to the table template in the Tree view.

c Rename the new data fields `dfAccount` and `dfType`, respectively.

Store Values into Global Variables

The values of the **ACCOUNT** and **TYPE** fields need to be stored as each row is read. This will allow the loan account number to be read from this screen and used for input on the **Customer Lookup** screen (**scrnCustomerLookup**). This requires an additional, account-type global variable.

1 Add a global variable representing the account type as follows:

a Open the **Business Entity Editor** by clicking the **Edit Business Entities** button.

b In the **Global Variables** tab click the **Add** button, to display the **Choose a Type** dialog box.

c Select **InternalString** and click **OK**.
A new global variable (**gvString**) is added to the **Instance** list of the **Global Variables** tab.

d Rename the new global variable by right-clicking on it, selecting **Rename** from the pop-up menu and entering the name `gvAccountType`.

e Close the **Business Entity Editor** by clicking on the **OK** button.

f Update the **dfAccount** data field to use the **gvAccountNumber** global variable as follows:

g In the **Data** tab's Tree view, select the **dfAccount** data field (under **scrnCustomerAccounts**' **ttscrnCustomerAccounts** node), to display its properties in the **Properties** view.

h In the **Update Data Source** property, choose one of the following:
- Manually enter `gvAccountNumber`,
Or
- Click on the ellipsis button to display the **Choose a Data Source** dialog box, select **gvAccountNumber** from the **Global Variables** list and click **OK**.

i Click the Apply button to apply the changes.

2 Update the **dfType** data field to use the **gvAccountType** global variable as follows:

a In the Tree view, select the **dfAccount** data field to display its properties in the **Properties** view.

b In the **Update Data Source** property, choose one of the following:
- Manually enter `gvAccountType`, *Or*
- Click on the ellipsis button to display the **Choose a Data Source**

dialog box, select **gvAccountType** from the **Global Variables** list and click **OK**.

c Click the Apply button to apply the changes.

Define the Table Template Properties

To define the table template properties:

1 In the **Data** tab, expand the **scrnCustomerAccounts** screen in the Tree view.

2 Select the **ttscrnCustomerAccounts** table template to display its properties in the **Properties** view.

3 Set the properties of the **Table Record Controls** tab as shown in Figure 76:

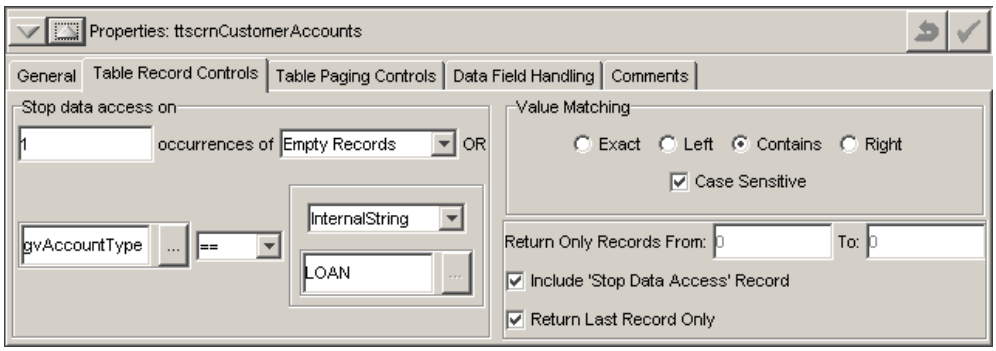

Figure 76. Setting Table Record Properties —Table Record Controls Tab

a In the bottom left property of the **Stop data access on** section, choose one of the following:
- Manually enter `gvAccountType`, *Or*
- Click on the ellipsis button ... to display the **Choose a Data Source** dialog box, select **gvAccountType** from the **Global Variables** list and click **OK**.

b Make sure **= =** is selected in the operator drop-down list.

c Set the data type drop-down list on the right to **InternalString**.

d In the bottom right property, manually enter the string `LOAN`.

e In the **Value Matching** section, make sure **Contains** and **Case Sensitive** are set.

f In the bottom right section, set the **Include 'Stop Data Access' Record** and **Return Last Record Only** check boxes.

g Click the **Apply** button to apply the changes.

4 Select the **Data Field Handling** tab and set the **Trim Options** as shown in Figure 77:

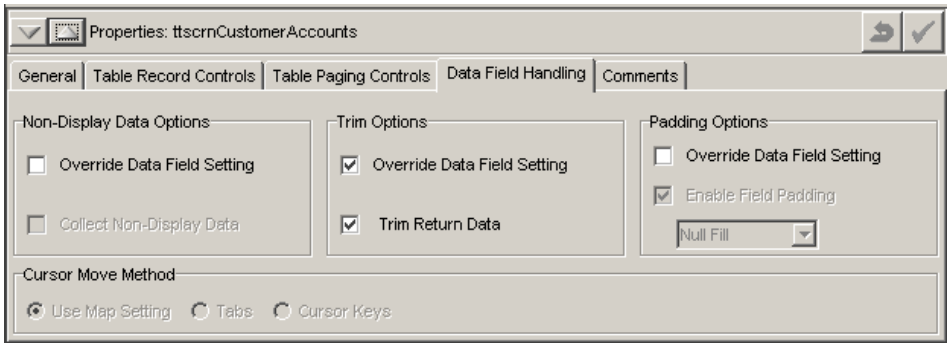

Figure 77. Setting Table Record Properties — Data Field Handling Tab

a Under **Trim Options**, check **Override Data Field Setting** and **Trim Return Data**.
These settings will trim leading and trailing spaces from the fields fetched by the table template.

b Click the `Apply` button to apply the changes.

Define a Method

You will now define a new Method named `mthConvertLoanTypes`. This Method will do the following:

1. Accept a social security number as input via an XML document.
2. Access the **Customer Accounts** screen and locate the loan account number for the given social security number.
3. Once the loan account number has been found, the Method will call (invoke) the existing **mthGetLoanInfo** Method, in order to obtain the loan details and transactions.
4. Finally, using conditional steps and XPath, the new Method will investigate each loan transaction and change the transaction type codes from their abbreviated forms ('PYMT' and 'LCHG') to their full forms ('PAYMENT' and 'LATE CHARGE', respectively).

When finished, your Method should look something like this (Figure 78):

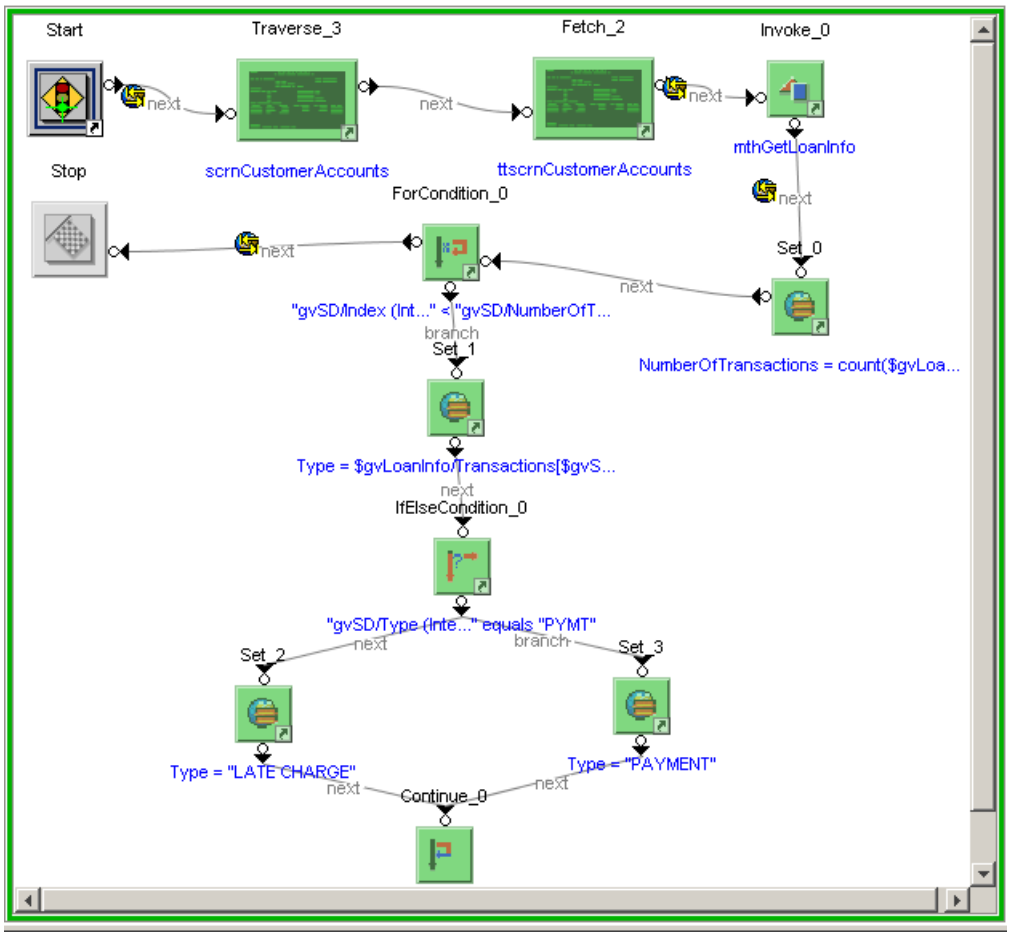

Figure 78. The mthConvertLoanTypes Method

Define New Business Entities

To define new buisiness entities:

1 The first item you need to define is an XSD XBE for the XML document containing the Social Security Number (SSN). Again, you will use a very simple XML document to pass in the SSN:

```
<SSN></SSN>
```

Proceed as follows:

a Open the **Business Entity Editor**.

b Select the **External Data Definitions** tab and its **XML/XSD** tab.

c Click the **Add** button.
A new XSD XBE (**XSD_0**) is added to the **Structure Definition** list.

d Right-click **XSD_0**, select **Rename** and enter the name `SSN`.

e Now create an IBE that represents the **SSN** XSD XBE:

f With **SSN** selected, click the **Copy to** button to display the **Copy to** dialog box (Figure 43).

g Since **Internal** is selected by default in the **Target(s)** list, just click **OK** to add the new IBE.
The **Internal Data Definitions** tab (under the **Internal Data Model** tab) now includes a new IBE named **SSN_IBE**.

2 This Method also needs a place to store the following information:

- The current transaction type.
- The number of loan transactions returned.
- An index for indexing (iterating) the collection of transactions.

You could very easily set up three global variables to store this information, but in this case we will illustrate another way to handle this requirement. As has already been illustrated by the **gvLoanInfo** global variable, a global variable can hold a data structure (see Step 17 on page 79). Therefore, instead of having multiple global variables for every separate piece of data that needs to be stored, you can create a single data structure to hold all separate pieces of data, and then assign that structure to one global variable.

To create the new data structure, proceed as follows:

a In the **Business Entity Editor**, select the **Internal Data Definitions** tab under the **Internal Data Model** tab.

b Add a new IBE by clicking the **Add** button.
The new IBE (**IBE_0**) is added to the **Structure Definition** list.

c Rename the new IBE by right-clicking it, selecting **Rename** and entering the name `SupportingData`.

d In the **Structure Definition** list, make sure **SupportingData** is selected.

e In the **Structure** pane, click the **New InternalNumber** button twice, and then click the **New InternalString** button once.

Three new elements (**sub_0**, **sub_1** and **sub_2**, respectively) are added to the **SupportingData** IBE.

f Rename **sub_0** to `NumberOfTransactions`, **sub_1** to `Index` and **sub_2** to `Type` (Figure 79).

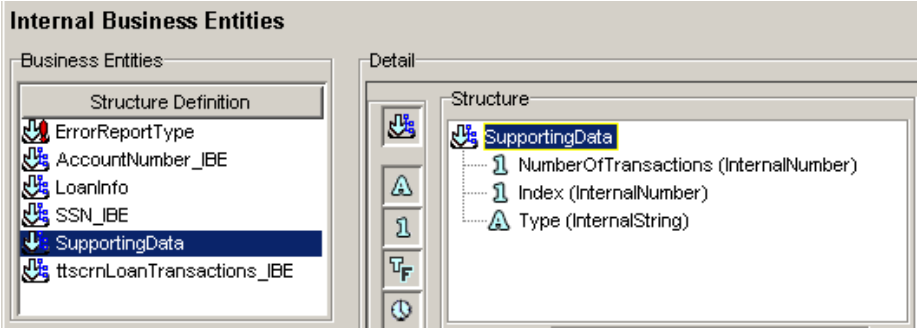

Figure 79. The SupportingData IBE Data Structure

g Since indexing works with integer values, you need to set the output format of **NumberOfTransactions** and **Index** to return only the integer part of the number stored.
To do this, select **NumberOfTransactions** (in the **Structure** section), right-click and select **Edit Format**.
The **Edit Format** dialog box is presented.

h Select the **Output Format Editor** tab, enter a single zero and click **OK** (Figure 80).

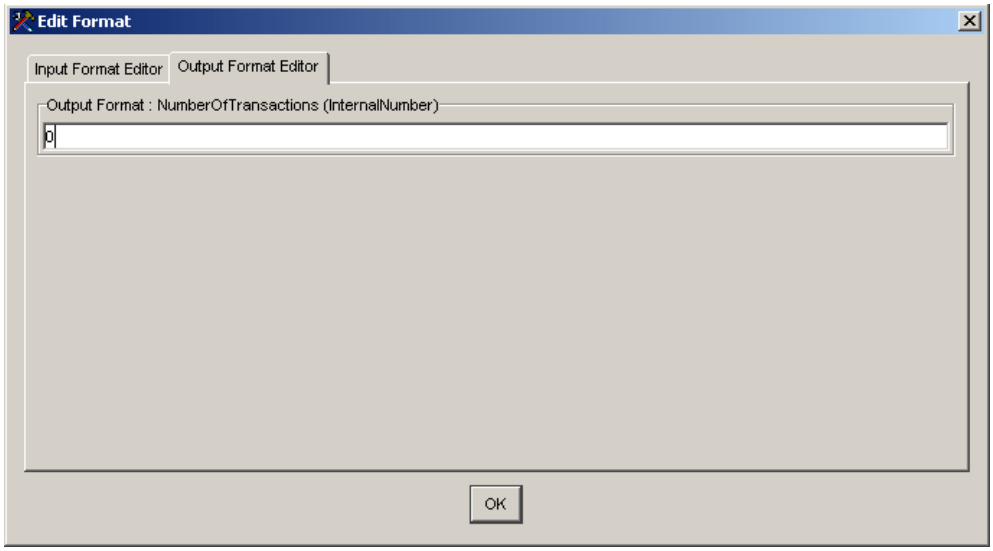

Figure 80. Edit Format Dialog Box — Output Format Editor

i Repeat Step g and Step h for the **Index** element.

Note: The 'Edit Format' dialog box can be used to format numbers in many different ways. For example, to represent negative numbers using ()'s instead of a negative sign, you can use an output format of (###.00).

3 Now that the **SupportingData** IBE has been defined, you can create a single global variable that can hold all three values: **NumberofTransactions**, **Index** and **Type**. Proceed as follows:

   a In the **Global Variables** tab, click the **Add** button to display the **Choose a Type** dialog box.

   b Scroll down the **Select Type** list and choose **Supporting Data**.

   c Click **OK**.
   A new global variable (**gvSupportingData**) is added to the **Instance** list.

   d Rename the new global variable `gvSD` (Figure 81).

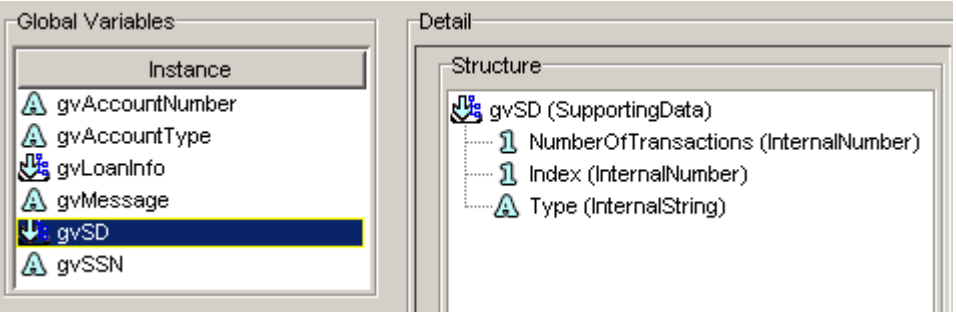

Figure 81. The gvSD Global Variable Structure

4 Click **OK** to close the **Business Entity Editor**.

Define the Input and Output Variables of the New Method

To define the input and output variables of the new method:

1 Add the new Method as follows:

   a In the **Methods** tab, right-click on the **MajorBank** node and select **Add Method**.
   A new Method (**Method_0**) is added to the Tree.

   b Rename the new Method `mthConvertLoanTypes`.

2 Expand the **mthConvertLoanTypes** branch by clicking on the "**+**" sign.

3 Add a new input variable to represent the SSN as follows:

a Right-click on the **Inputs** sub-branch and select **Add Input**.
A new input variable (**InVar_0**) is added to the Tree.

b Rename the new input variable `SSN`.

c In the **SSN** input variable's **Properties** view, set the **Internal Type** to `SSN_IBE` and the **External Type** to `SSN` (Figure 82).

d Click the **Apply** button to apply the changes.

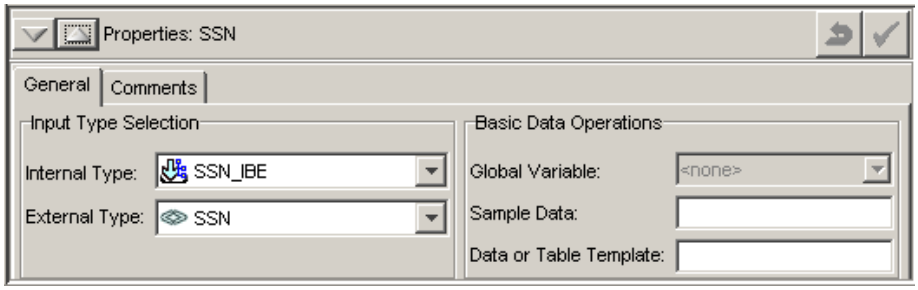

Figure 82. SSN Input Variable Properties View

4 Add a new output variable to represent the loan information as follows:

a Right-click on the **Outputs** sub-branch and select **Add Output**.

b Rename the output variable to `LoanInfo`.

c In the **Properties** view, set the **Internal Type** to `LoanInfo` and the **External Type** to `LoanInformation` (Figure 83).

d Click the **Apply** button to apply the changes.

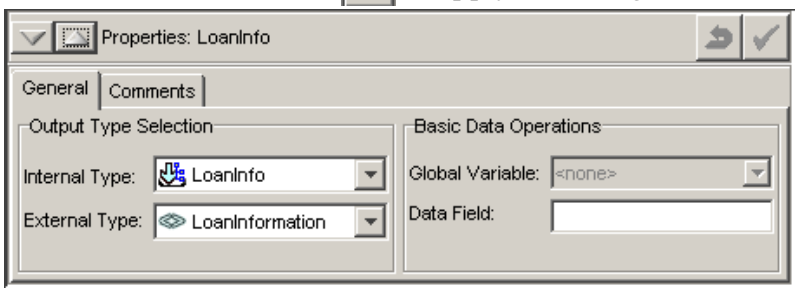

Figure 83. LoanInfo Output Variable Properties View

Create the Method steps

To create the method steps:

1 Drag a `Traverse` step ( ) to the Methods canvas.

2 Draw a `next` link from the **Start** step to the **Traverse_3** step.
The **Choose a Destination** dialog box is displayed.

3 In the **Select a Screen** list, select **scrnCustomerAccounts** and click **OK**.

4 Drag a `Fetch` step ( ) to the Methods canvas.

5 Draw a `next` link from the **Traverse_3** step to the **Fetch_2** step.

The table template **ttscrnCustomerAccounts** is automatically selected, as it is the only template defined on screen **scrnCustomerAccounts**.

6 Drag an `Internal Invoke` step ( ) to the Methods canvas.

7 Draw a `next` link from the **Fetch_2** step to the **Invoke_0** step.

The **Choose a Method** dialog box is displayed.

8 In the **Select a Method** list, select **mthGetLoanInfo** and click **OK**.

9 Drag a `Set` step ( ) to the Method canvas.

10 Draw a `next` link from the **Invoke_0** step to the **Set_0** step.

11 A **Set** step allows you to obtain the number of loan transactions returned from the **mthGetLoanInfo** Method, using the XPath `count` function.

With **Set_0** selected to display its properties in **Properties** view, set the **Assignment** section of the **General** tab as shown in Figure 84:

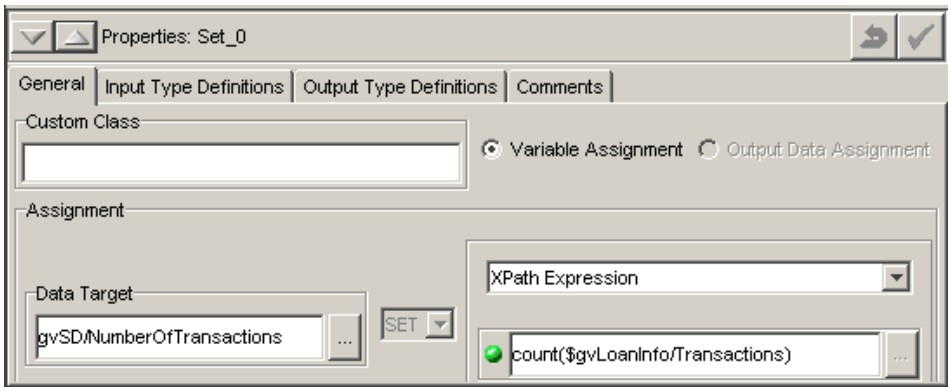

Figure 84. Set_0 Step Properties View — General tab: Assignment Settings

a Set the **Data Target** field in one of the following ways:
- Enter `gvSD/NumberofTra actions`, *Or*
- Click on the ellipsis button ... to display the **Choose a Data Source** dialog box.
In the **Global Variables** list, expand the **gvSD** data structure to select its **NumberofTransactions** element and click **OK**.

b In the right-hand data type drop-down list, select **XPath Expression**.
The XPath `count` function allows you to count the number of `Transaction` nodes in the `gvLoanInfo` structure. To refer to a global variable in an XPath statement, simply precede the reference with a dollar sign (e.g. `$gvLoanInfo`). The result of the XPath expression is stored into

the `NumberOfTransactions` element contained within the `gvSD` structure (see Figure 81).

c In the bottom right field, enter the following expression:
`count($gvLoanInfo/Transactions)`

d Click the **Apply** button to apply the changes.

> **Note:** MapMaker evaluates all XPath expressions for proper syntax and to ensure the validity of any references to global variables. The status of the expression is indicated by a colored ball icon that appears on its left.
> - A red ball indicates a "hard" error that must be corrected.
> - A yellow ball indicates that a possible error was detected but that MapMaker will accept the statement as written.
> - A green ball indicates that the XPath expression is valid.
> Use the XPath Evaluator check your XPath expressions, as described in the webMethods JI User Guide.

12 Drag a `ForCondition` step ( ) to the Method canvas.

13 Draw a `next` link from the **Set_0** step to the **ForCondition_0** step.

14 Draw a `next` link from the **ForCondition_0** step to the **Stop** step.

15 The **ForCondition** step allows you to do the following:

- Increment or decrement an index.
- Specify the condition under which the ForCondition continues to loop.

Select **ForCondition_0** to display its properties in the **Properties** view.

16 Select the **Initializer** tab.

This tab allows you to set the `Index` element of the `gvSD` global variable structure to zero. Set the **Initializer** properties as shown in Figure 85:

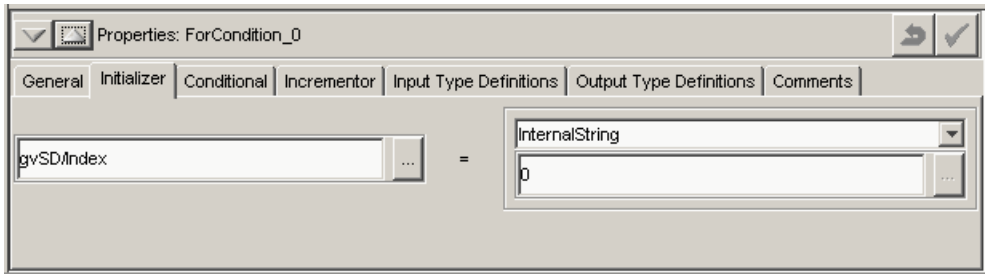

Figure 85. ForCondition Method Step Properties - Initializer Tab

a Set the left property using one of the following:
- Enter `gvSD/Index`, *Or*
- Click on the ellipsis button ... to display the **Choose a Data Source** dialog box.
In the **Global Variables** list, expand the **gvSD** data structure to select its **Index** element and click **OK**.

b Set the data type drop-down list (on the right) to **InternalString**.

In the right property, enter `0`.

c Click on the `Apply` button to apply the changes.

17 Next, select the **Conditional** tab.

This tab allows you to create a conditional expression that causes the `ForCondition` step to loop as long as the value of `Index` is less than the `NumberOfTransactions`.

Set the **Conditional** properties as shown in Figure 86:

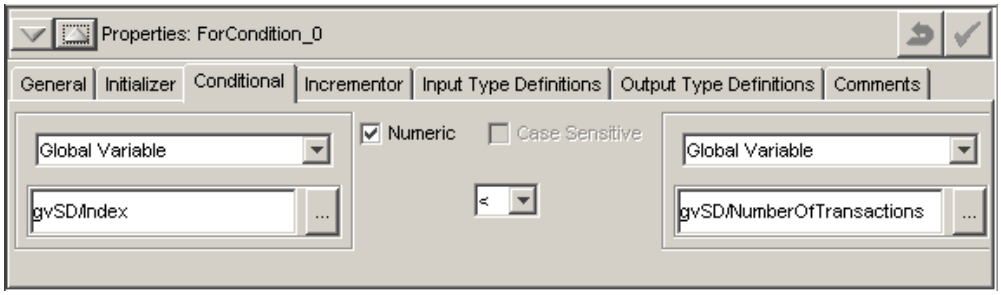

Figure 86. ForCondition Method Step Properties - Conditional Tab

a Set the left data type drop-down list to **Global Variable**.

b In the bottom left property, choose one of the following:
- Enter `gvSD/Index`, *Or*
- Click on the ellipsis button to display the **Choose a Data Source** dialog box.
In the **Global Variables** list, expand the **gvSD** data structure to select its **Index** element and click **OK**.

c Set the right object type drop-down list to **Global Variable**.

d In the bottom right property, choose one of the following:
- Enter `gvSD/NumberOfTransaction`, *Or*
- Click on the ellipsis button to display the **Choose a Data Source** dialog box.
In the **Global Variables** list, expand the **gvSD** data structure to select its **NumberOfTransactions** element and click **OK**.

> **Note:** Now that valid data types are selected on the left hand and right hand sides of the condition, the relevant operators are enabled.

e Since you are comparing numbers here, check the **Numeric** box.

f Set the middle operator drop-down list to `<`.

g Click the **Apply** button to apply the changes.

18 Finally, access the **Incrementor** tab to set the increment properties of the `ForCondition` step. In this example, you need to increment the `Index` by 1 each time the `ForCondition` step executes a loop instance.

Set the **Incrementor** properties as shown in Figure 87:

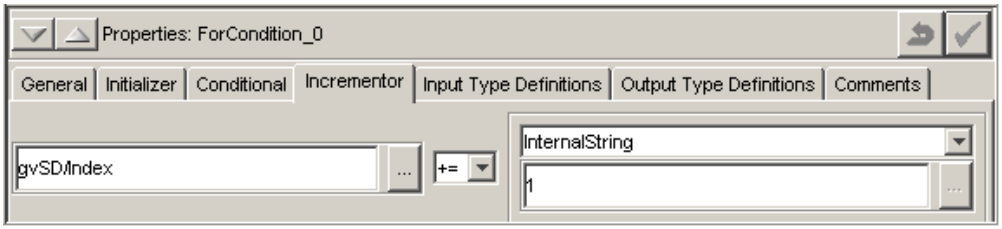

Figure 87. ForCondition Method Step Properties - Incrementor Tab

a In the left property, choose one of the following:
- Enter `gvSD/Index`,
Or
- Click on the ellipsis button ... to display the **Choose a Data Source** dialog box.

In the **Global Variables** list, expand the **gvSD** data structure to select its **Index** element and click **OK**.

b From the data type drop-down list on the right, choose **InternalString**.

c In the bottom right property, enter `1`.

d Make sure **+=** is selected in the middle operator.

e Click the **Apply** button to apply the changes.

19 Drag a `Set` step ( ) to the Methods canvas.

This **Set_1** step will be used to obtain the transaction type of the current loan transaction, as identified by the Index.

> **Note:** An XPath expression will be used to access the proper transaction. This XPath expression references gvLoanInfo (i.e. $gvLoanInfo) and gvSD (i.e. $gvSD).
>
> In addition, since XPath array indexing is 1-based, it requires adding 1 to your Index value. MapMaker structures, being Java-based, are-0 based for indexing.

20 In the Link Modes toolbar (on the left side of the Methods canvas), click on the `Branch` button to create a `branch` link.

21 Draw a `branch` link from **ForCondition_0** to **Set_1**.

22 In the Link Modes toolbar, reset the link mode back to `next` by clicking on the `Next` button .

23 Select **Set_1** to display its properties in the **Properties** view.

24 In the **General** tab of the **Set_1** step **Properties** view, set the **Assignment** section as shown in Figure 88:

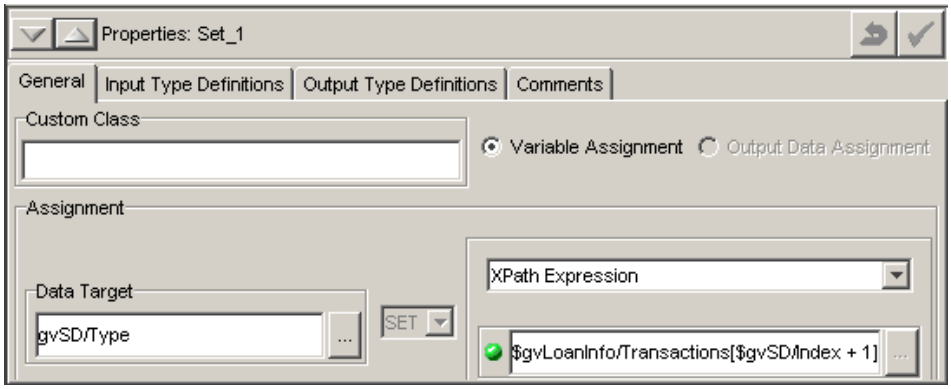

Figure 88. Set_1 Step Properties View — Assignment Section

a Set the **Data Target** property using one of the following:
- Enter `gvSD/Type`, *Or*
- Click on the ellipsis button ... to display the **Choose a Data Source** dialog box.
In the **Global Variables** list, expand the **gvSD** data structure to select its **Type** element and click **OK**.

b From the right data type drop-down list, choose **XPath Expression**.

c In the bottom right property, enter the following XPath expression:
`$gvLoanInfo/Transactions[$gvSD/Index + 1]/Type`

d Click the **Apply** button to apply the changes.

> **Note:** A green ball appears to the left of the XPath expression, indicating the validity of its syntax and its references to global variables.

25 Drag an `IfElseCondition` step ( ) to the Methods canvas.

This step allows you to determine the transaction type, by comparing the value that **Set_1** stores in **gvSD/Type** to the string **PYMT**:

- If **gvSD/Type** contains the string **PYMT**, the `branch` link of **IfElseCondition** is executed.
- If **gvSD/Type** does not contain the string **PYMT**, the `next` link of **IfElseCondition** is executed.

26 Draw a `next` link from **Set_1** to **IfElseCondition**.

27 Set the properties of this **IfElseCondition** step as shown in Figure 89:

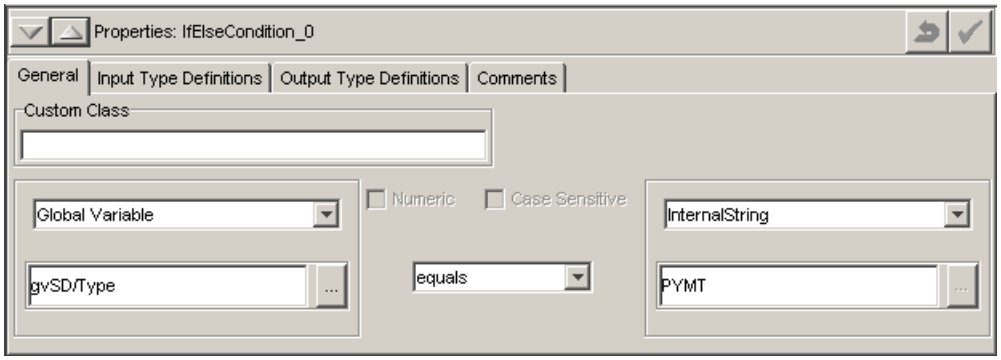

Figure 89. IfElseCondition Step Properties View — Properties Tab

a From the left data type drop-down list, select **Global Variable**.

b Set the bottom left property using one of the following:
- Enter `gvSD/Type`, *Or*
- Click on the ellipsis button ... to display the **Choose a Data Source** dialog box.
In the **Global Variables** list, expand the **gvSD** data structure to select its **Type** element and click **OK**.

c From the right data type drop-down list, select **InternalString**.

d In the bottom right property, enter `PYMT`.

> **Note:** Once valid data types are selected on the left hand and right hand sides of the condition, the relevant operators are enabled.

From the middle drop-down list, select **equals**.

Click the **Apply** button ✓ to apply the changes.

28 Drag a `Set` step ( ) to the Methods canvas.

This step, **Set_2**, is executed when the loan transaction Type is `LCHG`, and replaces it with the string `LATE CHARGE`.

29 Draw a `next` link from the **IfElseCondition_0** to **Set_2**.

30 Set the properties of this **Set_2** step as follows (Figure 90):

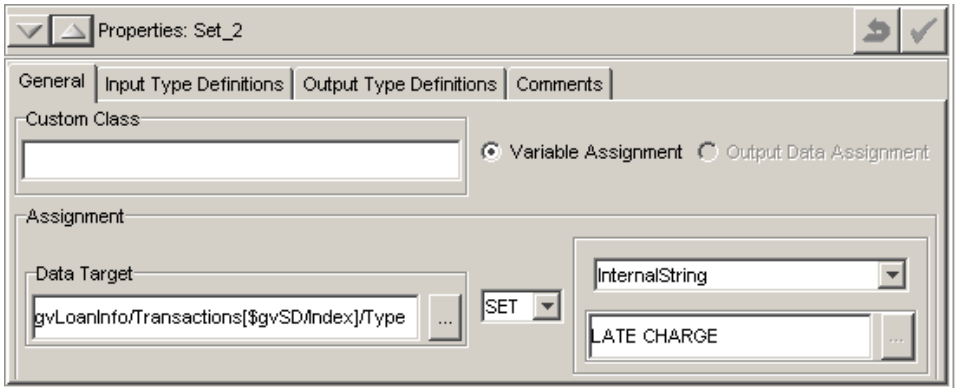

Figure 90. Set_2 Step Properties View — General Tab

a In the **Data Target** property, enter the following:
`gvLoanInfo/Transactions[$gvSD/Index]/Type`

b From the right data type drop-down list, choose **InternalString**.

c In the bottom right property, enter `LATE CHARGE`.

d Click the **Apply** button to apply the changes.

> **Note:** A dollar sign ($) is not needed to reference the global variable containing the element to be set. However, if other global variables are used to identify the element (e.g. [$gvSD/Index]), they must be preceded by a dollar sign.

31 Drag a `Continue` step ( ) to the Methods canvas.
The **Continue** step causes the **ForCondition** step to execute (loop) again.

32 Draw a `next` link from **Set_2** to **Continue_0**.

33 Drag a `Set` step ( ) to the Methods canvas.
This step, **Set_3**, is used when the loan transaction type is `PYMT`, and replaces it with the string `PAYMENT`.

34 In the Link Modes toolbar, click on the `Branch` button to create a `branch` link.

35 Draw a `branch` link from **IfElseCondition_0** to **Set_3**.

36 Reset the link mode to `next`.

37 Draw a `next` link from **Set_3** to **Continue**.

38 Set the **Assignment** properties for this **Set_3** step as shown in Figure 91:

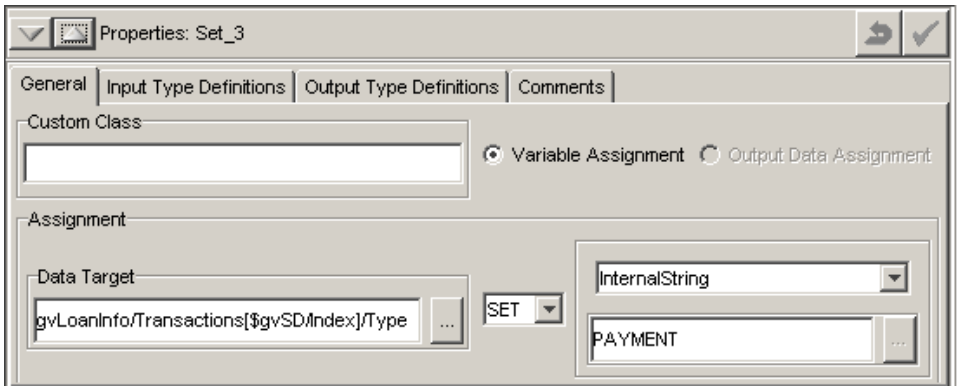

Figure 91. Set_3 Step Properties View — Assignment Section Properties

a In the **Data Target** property, enter the following:
`gvLoanInfo/Transactions[$gvSD/Index]/Type`

b From the right data type drop-down list, choose **InternalString**.

c In the bottom right property, enter `PAYMENT`.

d Click the **Apply** button to apply the changes.

> **Note:** At this point, the Tree view is no longer colored and the steps and frame of the Presentation view are colored green, indicating that the Method is properly configured.

Define the Data Mappings for the Method

To define the data mappings for the method:

1. Map the **SSN** from the input variable to the **gvSSN** global variable as follows:
   a. Double-click on the **next** link between **Start** and **Traverse_3**.
   The **Data Mapping Editor** is displayed.
   b. In the **Source** list, expand **SSN** and select **_TEXT**.
   c. In the **Global Variables** list, select **gvSSN**.
   d. Click on the **Create Data Map** button.
   e. Close the **Data Mapping Editor** by clicking on the **OK** button.
   A new data mapping icon is added to the **next** link.
2. Map the account number contained in the **gvAccountNumber** global variable into the **LoanAccountNumber** input variable structure (defined for the **mthGetLoanInfo** Method) as follows:
   a. Double-click on the **next** link between **Fetch_2** and **Invoke_0**.
   The **Data Mapping Editor** is displayed.
   b. In the **Global Variables** list, select **gvAccountNumber**.
   c. In the **Target** list, expand **LoanAccountNumber** and select **#TEXT**.
   d. Click on the **Create Data Map** button.
   e. Close the **Data Mapping Editor** by clicking on the **OK** button.
   A new data mapping icon is added to the **next** link.
3. Map the output of the **mthGetLoanInfo** Method into the **LoanInfo** structure held by the **gvLoanInfo** global variable as follows:

a Double-click on the **next** link between **Invoke_0** and **Set_0**.
The **Data Mapping Editor** is displayed.

b In the **Source** list, select **LoanInfo**.

c In the **Global Variables** list, select **gvLoanInfo**.

d Click on the **Create Data Map** button.

e Close the **Data Mapping Editor** by clicking on the **OK** button.
A new data mapping icon is added to the **next** link.

4 Map the **LoanInfo** structure held by **gvLoanInfo** into the **LoanInfo** output variable of this Method as follows:

a Double-click on the **Next** link between **ForCondition_0** and **Stop**.
The **Data Mapping Editor** is displayed.

b In the **Global Variables** list, select **gvLoanInfo**.

c In the **Target** list, select **LoanInfo**.

d Click on the **Create Data Map** button.

e Close the **Data Mapping Editor** by clicking on the **OK** button.
A new data mapping icon is added to the **next** link.

Add the New Method to the Service

To add the method to the service:

1 Select the **Services** tab.

2 Right-click on `MajorBankSvc` and select **Add Method to Service**.
The **Choose a Method** dialog box is displayed.

3 Select **mthConvertLoanTypes** and click **OK**.

4 Select **File** > **Save**.

Generate, Test and Deploy the Service

To generate, test, and deploy the service:

1 Generate the service (select **Tools** > **Generate** > **Generate All Code....**) — see "Generating the Service and Testing the Client Code" on page 101.

2 Deploy the service — see "Deploying the Service" on page 104.

3 Test the service — see "Testing the Service" on page 112.

Note: When editing the *MajorBankSvc_jclient3.java* file, use `987654321` (instead of 1234567898) as the Account Number.

The service output consists of two long strings, representing the XML output of the two methods: mthConvertLoanTypes and the mthGetLoanInfo it invoked. The following is a formatted example of the `mthConvertLoanTypes` method output.

Example 8 - 2 Formatted Output of the mthConvertLoanTypes method

```
<LoanInformation>
    <AccountNumber>1234567898</AccountNumber>
    <Name>PAUL AGEE</Name>
    <Profile>
        <BasePrincipal>50000.00</BasePrincipal>
        <Collateral>75000.00</Collateral>
        <Term>120</Term>
    </Profile>
    <Transactions>
        <Date>12/08/97</Date>
        <Amount>702.98</Amount>
        <Type>PAYMENT</Type>
    </Transactions>
    <Transactions>
   <Date>05/05/97</Date>
        <Amount>770.30</Amount>
        <Type>LATE CHARGE</Type>
</Transactions>
</LoanInformation>"
```

Note: The output of the 'mthConvertLoanTypes' method is identical to the output of the 'mthGetLoanInfo' method (see "Formatted Output of the mthGetLoanInfo method" on page 115), except that the abbreviated transaction type codes ('PYMT' and 'LCHG') have been replaced with their full forms ('PAYMENT' and 'LATE CHARGE', respectively).

Appendix A. Additional Tutorial Information

Tutorial Navigation Tips

For the *MajorBankHost.map* used in this tutorial, all Function keys noted at the bottom of the screens work as noted on the screen except for F1=HELP. F1 does not work.

When an "invalid" request is made, MapPlayer may advance to another screen by using a default "first" action it finds. Thus, pressing F7 to advance past the last page of transactions may advance to the look up screen for example.

Customer Accounts for this Tutorial

The valid accounts are (all account lookup screens use Enter to advance to a details screens):

Customer Information

Paul Agee 987654321

Kate Bancroft 123456789

Loan Accounts

Paul Agee 1234567898

Kate Bancroft 1234567899

Savings Accounts

Paul Agee 8197932487

Kate Brancroft 5681816851

Checking Accounts

Customer Information

Paul Agee 0585234780

Kate Brancroft 6023781200

Locating Accounts

On the Loan Lookup screen accounts can be located using a Name lookup. If, instead of entering an account number, you fill in the **Last Name**, **First Initial** and **State** fields, then you will reach a screen containing a list of accounts. There are only two names that work and both access the same loan locator screens. The name lookup takes the last name, first initial and state.

Valid values are:

```
Last Name: agee
First Initial: p
State: ca
```

-OR-

```
Last Name: brancroft
First Initial: k
State: ca
```

> **Note:** Of the listed accounts, only the ones for P AGEE in CA and K BRANCROFT in CA are valid.

Place an X next to either of these records and press the ENTER key to access the loan details screen.

Glossary

ACL	Access Control List
Actions	Used by webMethods JI Java services to navigate from one legacy screen to the next on the legacy application. An action includes all data that was input by the user during trail recording in MapMaker, along with the AID key or Action that caused the screen transition.
Action Key	A key sequence that performs an action in the legacy application. Action keys are valid for the Telnet protocol and are similar in concept to AID keys.
Applet	A program written in the Java programming language that is accessed from a Web browser.
Application	A Java program run as a stand-alone program.
API	API - Application Programming Interface. The library of C or Java functions callable from UNIX and Windows programs. Used to develop webMethods JI clients and services.
AID Key	The Attention Identifier Key (AID). A single key on the keyboard that, when pressed by the user, performs an action in the legacy application. Typical AID keys include the Enter and PF keys, although the legacy application may change their usage or use other AID keys. AID keys are valid for the TN3270 and TN5250 protocols.
Browser	A program that allows users to access information on a Web server. Also known as a Web browser.
CGI	Common Gateway Interface. A standard method for external gateway programs to interface with Web servers.

Character Encoding

The format or encoding of a language-set character. Character encodings are usually 1, 2, 3, or 4 bytes. Unicode is an example of a 2-byte character encoding. Other examples are ASCII, EBCDIC, and UTF-8.

Character Mode

Character mode describes the functionality in webMethods JI that communicates with character-based applications over the Telnet protocol.

Client

In webMethods JI, client refers to one of two items:

- A Runtime version of an application developed in Java that uses webMethods JI client APIs to communicate with webMethods JI services.
- Also refers to a third-party software application that interfaces with webMethods JI services and functions similarly to an application developed with a webMethods JI client API.

Client Functions

The C or Java functions used to allow clients to connect to services, execute service methods, and input and extract data.

Content Pane

A content pane, also called a panel, is a GUI component that acts as a container for various GUI components. A content pane is basically a window that other GUI objects, such as buttons and text fields, are placed on.

Cookie

A general mechanism used by Web servers to both store and retrieve information on the client side of the connection.

Custom Classes

Classes that are used to “extend” or customize service code that was generated in MapMaker.

Data Field

A data field is an individual field on the legacy screen that is added to either a data template or a table template. Data fields are used in conjunction with output variables to extract data from the legacy screen.

Data Mapping

Refers to the mapping of data in the flow of a method. Every point in a method where data is sent to or retrieved from an external source requires data mapping. Data mapping is defined in the Data Mapping Editor.

Data Stream The flow, or stream, of information between computer programs. Data on the data stream is represented using "character encoding" and is transferred using a mutually agreed upon protocol.

Data Template A data template is a logical representation of non-repeating data fields on the legacy screen. Data templates are defined in MapMaker and are used in conjunction with output variables to extract data from the legacy application. Similar to table template, used to define repeating data fields.

Data Typing Refers to the creation and definition of data types. Data types are defined and maintained in MapMaker's Business Entity Editor.

DBCS Double-Byte Character Set.

DLL Dynamically Linked Library. A library of function calls used in Windows environments.

DOM Document Object Model (aka "random access" protocol for XML) – an XML parser that converts the XML document into a collection of objects, which can then be manipulated in any way you choose.

DTD Document Type Definition for XML – an optional part of the XML document prolog that specifies the kinds of tags that can be included in an XML document and the valid arrangement of those tags.

EAServiceBean The interface between webMethods JI Java service code and the JService that manages the service in the webMethods JI server environment. The EAServiceBean can be extended or customized to change the interface if required.

ECS Extended Character Support.

EIS Enterprise Information System - an application providing information of critical importance to the day-to-day planning and/or operation of a business. EISs are generally run on larger platforms, such as mainframes or minicomputers. EISs provide the information infrastructure for an enterprise. Examples of EISs include enterprise resource planning systems, mainframe transaction processing systems, relational database management systems, and other legacy information systems.

Enterprise System A system involved in an organization's critical business processes. Typically, enterprise systems are large and complex, use database management systems (DBMSs), and run on mainframes or minicomputers.

Formatted Fields Fields on the legacy application that have special formatting characteristics. MapMaker identifies all such fields on the legacy screen and uses the field layout to match screens (unless the fields are disabled and Tags are used to identify screens).

GBBasic A Java package, *com.jacada.mapstudio.GBBasic*, that is included with webMethods JI and can be used to extend or customize Java service code that was generated in MapMaker.

GUI Graphical User Interface. An application that allows users to interface with computer programs in a graphical environment. In webMethods JI, MapMaker, the Configuration Manager, and the System Monitor are all Graphical User Interfaces.

HTML HyperText Markup Language, a format used to create Web documents.

Hos **t**A computer machine where applications reside. In webMethods JI, hosts can be the machine on which components of the webMethods JI environment are running, the machine on which the telnet, TN3270, or TN5250 server reside, or the machine on which the legacy applications reside.

IBE Internal Business Entity. Refers to data types that are used internally by the webMethods JI Service. A global variable may be defined of an IBE data type.

IDE Integrated Development Environment. Refers to a graphical development tool that uses standard GUI components to facilitate application development.

Jacada Integrator See webMethods JI.

Java An object-oriented, platform-independent programming language developed by Oracle.

Java services webMethods JI services developed using the Java programming language. Java services are generated from the MapMaker graphical development interface (GUI) and can be customized using the custom service classes included with webMethods JI.

JClient3 The Java Client Library version 3 is an improved version of the JClient, which allows webMethods JI clients to be developed using JDK 17.

JDBC Java DataBase Connectivity.

JDK Java Development Kit. The development environment for the Java programming language. Includes a Java Runtime Environment.

JRE Java Runtime Environment. The minimum environment required to run Java applications. This is a combination of a JVM along with the core classes and files required to run Java applications.

JVM Java Virtual Machine. A Java interpreter that converts Java code into executable code.

Legacy data Data residing on a mainframe platform.

Legacy host The machine or host on which legacy applications reside.

Map The logical representation of the screens, fields, data input and AID keys that make up the user interaction with a legacy application. Maps are created in MapMaker. Note that webMethods JI's use of the term Map is distinct from the java.util.Map that is included with Java.

MapMaker The graphical development interface provided with webMethods JI for the purpose of developing Java services. Used to record trails, maps, data and table templates, create methods and services, and then generate and optionally deploy the services into the webMethods JI server environment.

Methods Object-oriented entity of one or more functions. A collection of methods, initialization code, and events make up a service.

MLM Map-List-Map. Refers to an XBE data type used for communication with a client, such as a Java, C, or VB Client.

Multicasting A connectionless IP networking communication in which applications on the IP network broadcast information over a well-known socket.

Multithread-safe See thread-safe.

Multithreaded See threaded.

Offset Refers to the location of data on the legacy screen. The offset is determined using the following format: For an 80 column screen, the offset is (column # - 1) + 80 x (row # -1). For example, the first column of the second row is position 80:
(1 - 1) + 80 x (2-1).

Package A collection of java classes that are grouped together to form a logical combination of classes.

Presentation Space A representation of the communications between the legacy host to the webMethods JI environment, including the screen and field information from the legacy application.

Protocol Agent A software interface that governs the procedures used to exchange information between physically remote entities such as computer systems.The Protocol Agent governs the format of the messages, the generation of checking information, and the flow control, as well as the actions to take in the event of errors.

Proxy Server Special instances of Resource Servers that allow multicasting communication to take place over multiple sub-nets.

Resources The components of webMethods JI that are managed by the Resource Server. These components include Resource Databases and license files.

Resource Database A database that is used in the webMethods JI environment to store environment and service configuration, along with service code and maps.

Resource Serve **r**Manages the communication between environment managers and the webMethods JI resources.

RMI Remote Method Invocation. A standard from Oracle that allows distributed Java objects to communicate with each other over TCP/IP networks.

Screen The screen, as contained in the data stream, that is coming from the legacy host.

Screen Mapping The process of marking the physical boundaries of, and defining the screen components found on, an external application screen. The components include tags, fields, areas, repeating areas, and repeating fields.

Service A collection of methods which answer requests from a client.

SOCKS SOCKS is a firewall proxy protocol.

System Monitor The tool used to monitor the webMethods JI System. It allows you to monitor, log, and view real-time activity for all or selected Environment Managers, JClusters and JServices, clients, and services.

Table Template A Table Template is a logical representation of the area on a legacy screen that contains repeating Data Fields. Table templates are defined in MapMaker and are used in conjunction with output variables to extract data from the legacy application. Similar to Data Template, used to define non-repeating Data Fields.

Tag A user-defined component of the host application screen that most often serves as a label for fields. You can define any static screen text as a tag. MapMaker can identify external application screens by the tags that are defined for them.

Telnet A TCP/IP-based terminal emulation protocol. Requires a Telnet server. Telnet is also used to describe the Character Mode functionality within webMethods JI.

Terminfo UNIX terminal information database. See the UNIX terminfo(4) man page.

Thread-safe Also multithread-safe (MT-safe). A description of a function or library that may be called in a threaded environment without any additional coding.

Thread A single flow of control within a process or address space. Programs using two or more threads are referred to as threaded or multi-threaded.

Threaded Also multithreaded. A form of multi-tasking that uses multiple independent execution threads.

TN3270 An implementation of the telnet protocol that is used to communicate between TCP/IP networks and IBM mainframe applications that use IBM 3270 terminals. A TN3270 server is required for connection from the TCP/IP network to the mainframe.

TN5250 An implementation of the telnet protocol that is used to communicate between TCP/IP networks and IBM AS400 applications that use IBM 5250 terminals. A TN5250 server is required for connection from the TCP/IP network to the AS400.

Trail The linear path of all screens encountered during navigation through a host application. MapMaker records trails during host application interaction.

Unicode A universal character code.

Web A network of computers based on the client-server model. The Web uses Web browsers to access information from a Web server. A Web can be a part of the World Wide Web or can be a part of a separate network, also know as an “Intranet”.

Web browser A program that allows users to access information on a Web server.

webMethods JI	Consists of one or more Environment Managers, Resource Servers, resources including the Resource Database, and webMethods JI clients and services.
XBE	eXternal Business Entity. Refers to data types that are used for the webMethods JI Service to send and receive data to and from an external source. Such external sources are Legacy Screens and Clients.
XML	eXtensible Markup Language – a text-based markup language that is fast becoming the standard for data interchange on the web.
XSD	XML Schema Definition. Specifies how to formally describe the elements in an XML document. One of the XBE data types supported by MapMaker is XML/XSD.

Index

A

Action 57
Action Key 143
Actions 143
    linking to Global Variables 60
AID Key 143
API 143
Applet 143
Application Programming Interface 143

B

Browser 143
Business Entity 70
    Copy to operation 72
    ErrorReportType 75
    external 70
    internal 71

C

CGI 143, 144
Character Encoding 144
Character Mode 144
Client 144
    compiling 112
    functions 144
    generating 101
Configuration Manager 106
Cookie 144
Copy to operation 72
Create Relations 73

D

Data Field 144
Data Mapping 94, 144
Data Mapping Editor 94
Data Modeling 17
Data Panel 31
Data Stream 145
Data Structure 70
    public 78
Data Template 17, 61
    data field 17
Data Templates 145
Data Typing 145
DBCS 145
DLL 145
Document Object Model (DOM) 145
Document Type Definition (DTD) 145
Documentation 7
    Viewing on-line 9
DOM 145
DTD 145

E

EAServiceBean 145
ECS 145
Enterprise System 146
Environment
    starting 103
Extensible Markup Language (XML) 151
External Business Entity 70
    Legacy 70
    MLM 70
    XML/XSD 70

F

Fields
    Formatted 146

Formatted Fields 146
Formatting Conventions 7

G

GBBasic 146
Global Variables 56
GUI 146

H

Host 146
Host Connection
    defining 40
HTML 146

I

IBE 146
IDE 146
Input variable 88, 90
Internal Business Entity (IBE) 71
isArray 76

J

Java
    runtime environment (JRE) 147
    virtual machine (JVM) 147
Java Client Library 12
Java Development Kit (JDK) 147
Java Runtime Environment (JRE) 147
Java Services 15, 147
Java Test Client
    compiling 112
    generating 101
Java Virtual Machine (JVM) 147
JClient 12, 147
JDBC 147
JDK 147
JRE 147
JVM 147

L

Legacy Data 147
Legacy External Business Entity
    definition 70
Legacy Host 147
Link Modes
    Auto Arrange 83
    Branch 83
    Next 83
    onFail 83
    Swap Content 83

M

Map 147
    updating 53
Map Panel 31
MapMaker 15, 148
MapPlayer 23
Method
    defining 80, 88
Method Steps
    adding and inserting method steps 80
    Break 20
    Continue 20, 137
    DoWhileCondition 20
    EndIf 20
    EndThread 20
    External Invoke 19
    Fetch 18
    ForCondition 19, 132
    IfCondition 19
    IfElseCondition 19, 135
    Internal Invoke 19
    JClient3 19

linking method steps 82
Perform 18
Set 20, 131
Thread 20
ThreadJoin 20
Traverse 18
UserInteraction 19
WhileCondition 20
Write 18
Methods 148
Methods Panel 31
Middleware 11
MLM External Business Entity
definition 70
Multicast 148
Multithreaded 148
Multithread-safe 148

O

Offset 148
onFail Processing 83
Output variable 88, 91

P

Package 148
Presentation Space 148
Presentation view 30
Properties view 30
comments tab 34
Protocol Agent 148
Proxy Server 148

R

Resource Database 104, 149
Starting 103
Resource Manager 149
Resource Server 107
Starting 103
Resources 149
RMI 149

S

Screen 149
mapping 149
Server Environment 12
Service 15, 20, 149
Configuring 106
Defining 99
Deploying 104
Generating 101
Testing 112
Service Panel 31
SOCKS 149
Status bar 87
Synchronization 73
System Monitor 149

T

Table Template 17, 63, 149
data field 17
Tag 55, 149
Telnet 150
Terminfo 150
Test Client
compiling 112
generating 101
Thread 150
Threaded 150
Thread-safe 150
TN3270 150
TN5250 150
Trail 150
Trail Panel 31

U

Unicode 150

W

Web 150
Web Browser 150
webMethods JI
    service 15
webMethods JI environment 147, 151

X

XBE 151
XML 151
XML/XSD External Business Entity
    definition 70
XPath 134
    count function 131
    validation 132