webMethods JI

Integration Guide

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-INTG-46-20260923

Table of Contents

About this Guide 5
Before You Begin 5
Organization 5
Formatting Conventions 6
Documentation Set 6
Viewing the Documentation Online 8

Chapter 1. The HTTP Gateway 9
About the HTTP Gateway Servlet. 9
HTTP Gateway Servlet Restrictions 10
General 10
Installing the HTTP Gateway Servlet 11
Tomcat 12
Servlet Initialization 14
The Properties File 15
Message Conventions 18
Setting Client Options 18
Specifying Client Options. 23
Message Encoding. 24
HTTP Request Messages 24
Request Message Request Line. 24
Request Message Headers. 25
Request Message Body 26
Response Message Status Line 27
Response Message Headers 27
Response Message Body. 27
Siebel XML Support 29
Siebel VBC (Virtual Business Component) XML Interface 29
Siebel VBC XML Formats 29
Initialization (Init) 30
Query. 30
Delete 32
Update 32
Pre-Insert. 33
Insert 34
Error Conditions 35
Processing the Siebel Requests 35
Property File Parameters. 35
Sessioning Support 39
Service Sessioning 39
HTTP Gateway Error Messages 40
Error Handling 40
Logging 40

Getting Debug Information 41
Scaling 42
Version Information 42

Chapter 2. The MQ Gateway 43
About the MQ Gateway 43
MQ Gateway Issues and Restrictions 45
MQSeries Configurations 46
MQSeries Message Conventions 46
MQSeries Client Interfaces 46
Setting Client Options 47
Using the ReplyTo Field 50
Ensuring Receipt of Proper Reply Message 50
The mqgw.cfg File 50
The JMS Format 55
Support for JMS and Non-JMS Clients 55
Sessioning Support 56
Service Sessioning 56
Starting and Stopping the MQ Gateway 58
Starting the MQ Gateway 58
Stopping the MQ Gateway 59
MQ Gateway Status 59
MQ Gateway Error Messages 60
Error Handling 60
Error Logging 61

Chapter 3. XML and XSLT Support in webMethods JI 63
XML and XSLT Support Overview 63
String Size Support 63
XSLT Processor 63
Invoking a Method and sending Input Data to the Service 63
Client Input Message - Specifying XML Parsing 64
Errors Encountered During Parsing 64
MapMaker Generated DTD 64
Parsing Rules 65
Overview 65
Configurable Parsing Options 66
Attribute Parsing Options 66
XSLT Support 68
XSLT Method Input Data as a String 69
XSLT Method Input Data as a URI 70
Returning XML Output to the Client 71
Logging 72
JAXP Warning Message 72

Chapter 4. Web Services 75
About Web Services 75
Web Service Architecture 77
Service Transport: HTTP 78
XML Messaging: SOAP 78
Service Description: WSDL 81
Service Discovery: UDDI 87
Developing a webMethods JI Web Service 91
Generating a WSDL in MapMaker 92
WSDL Generation — Overview 92
Generating the WSDL Code 92
Additional WSDL Settings in MapMaker 93
Setting the Default Path for Generated WSDLs 93
Setting the Default Base Path for SOAP Address Binding 94
Setting Up the SOAP Gateway 95
webMethods JI SOAP Gateway — Overview 95
SOAP Gateway Requirements 95
The SOAP Gateway Messaging Styles 95
SOAP Gateway Packaging 96
Initialization Parameters 97
SOAP Gateway Request URL 98
Determining the Properties File to be Used 98
Determining the webMethods JI Service Name 99
Query Parameter Support 100
SOAP Gateway Logging 100
Version Information 101
Configuring the Web Container 101
Testing your Generated Web Service 103
Publishing a Web Service to UDDI 105
Publishing your Web Service via the UDDI API 106
Publishing your Web Service to UDDI via Web-based Publishing 107

Chapter 5. MQSeries Integration 113
MQSeries 113
JMS Overview 113
Prerequisites 113
Using the JMS MQ Series Test Service Bean 114
Writing Non-host Service Beans 116
Example Custom Java Service 116

Chapter 6. Siebel 7 Integration 127
Siebel 7 127
Siebel eBusiness Applications 127
Siebel Applets 128
Siebel Business Components 128
Siebel Call Center 128
Siebel Tools 130

Installing Siebel Tools 131
Integrating webMethods JI with Siebel 132
Setting up the HTTP Gateway Servlet Environment 132
Setting up the Siebel Environment 133
Setting up the Siebel Web Client 133
Setting up the Siebel Data Base 134
Setting-up Siebel Tools 134
Integration using a Virtual Business Component 135
The Siebel EAi Environment 136
Development in Siebel Tools 137
Creating a New Project 138
Modifying an Existing Business Component. 139
Modifying a Business Component's Associated Applets 140
Creating a Virtual Business Component 141
Creating Fields in the Virtual Business Component 142
Setting the Virtual Business Component's User Properties 143
Linking the Modified Business Component to the New Virtual Business Component 144
Modifying an Existing Business Object. 145
Creating a New List Applet 146
Adding Fields to the New List Applet 148
Creating a Web Layout for the Applet 149
Adding the New Applet to the View 151
Compiling the Project 153
Testing the Software AG Integration with Siebel 154
Files Used for Integration 155
QueryDemoRet.xsl. 155
QueryDemoReq.xsl 155
MajorBankVBC.cfg 156
Integration Using Integration Objects 156
Runtime Architecture 157
Development in Siebel Tools 157
Creating an Internal Integration Object 158
Creating the External Integration Object 163
Compiling the Integration Objects. 165
Mapping the Data. 166
Creating the Workflow. 168
Testing the Workflow. 170
Files Used for Integration 171
MajorBankIO.cfg 171
DTD for Get_Checking Method 171

Glossary 173

Index 1

About this Guide

Welcome to the *webMethods JI Integration Guide*.

Before You Begin

This guide is intended for software developers intending to integrate other technologies with webMethods JI. In order to get the most out of this book, developers should have a working knowledge of the following:

- UNIX system administration.
- C programming language for developing webMethods JI clients and services.
- The location and communication requirements of existing legacy data and applications.

Organization

This guide is organized as follows:

- Chapter 1 - "The HTTP Gateway" on page 9
- Chapter 2 - "The MQ Gateway" on page 43
- Chapter 3 - "XML and XSLT Support in webMethods JI" on page 63
- Chapter 4 - "Web Services" on page 75
- Chapter 5 - "MQSeries Integration" on page 113
- Chapter 6 - "Siebel 7 Integration" on page 127
- “Glossary” on page 173
- “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. The HTTP Gateway

This document describes webMethods JI's HTTP Gateway. The HTTP Gateway servlet performs the following functions:

- It accepts an HTTP Servlet Request from any client application via an HTTP server such as a web server with a servlet engine, an application server, or any stand-alone system (e.g. servletrunner) that will accept an HTTP request, and passes it on to a servlet.

> **Note:** Unless otherwise noted, the entity that passes the HTTP request to the servlet and receives the HTTP response from the servlet, is referred to as the "servlet engine".

- Establishes a connection to the webMethods JI Environment Manager specified in the data stream with the connection properties specified in the data stream or set in property files.
- Passes the webMethods JI request embedded in the data stream on to webMethods JI (via the JClient3 library).
- Receives the webMethods JI response from the Environment Manager, extracts the XML data element from the response, and passes the resulting information back to the client application.

Assumptions

- This chapter assumes that the user is familiar with webMethods JI services. Services are discussed in this chapter only as they relate to, or are used by, the HTTP Gateway servlet. For more information on writing webMethods JI services, see the *webMethods JI User's Guide*.
- The implementation of the HTTP Gateway servlet requires the installation and configuration of a third-party servlet engine application. The HTTP Gateway servlet has been validated against the following servlet engines:
  - Apache Tomcat 9.0.109 (Tomcat 10 is unsupported in JI 4.6)

About the HTTP Gateway Servlet

The following figure shows the relationship between servlet clients, the servlet, and the webMethods JI environment:

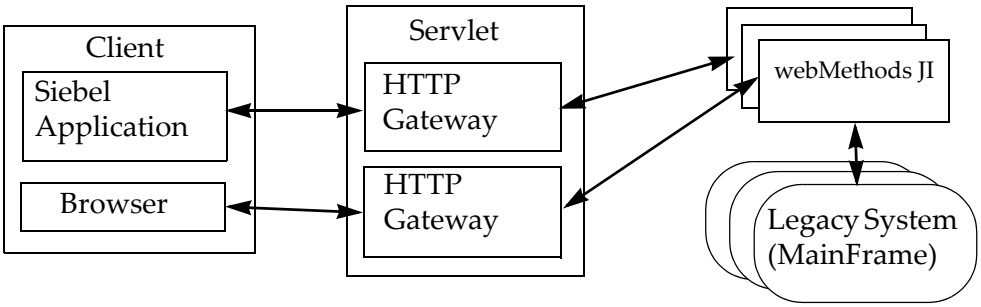


Figure 1. The relationship between servlet clients, the servlet, and the webMethods JI environment

The servlet receives a data stream, which includes the specification of all parameters necessary to execute a service/method on webMethods JI. There is a servlet properties file in which the default values for all parameters may be specified.

Clients send one or more request messages and each client request message elicits a single response message from the servlet. Client requests contain data that is needed for the servlet to interface with an webMethods JI Service. The protocol for requests and responses is HTTP.

HTTP Gateway Servlet Restrictions

General

The HTTP Gateway servlet supports only GET and POST HTTP request methods (see Message Encoding, or RFC 1945).

The following Content-Types have been validated:

- application/text
- application/xml
- application/x-www-form-urlencoded
- text/plain
- text/xml

Note: Multipart Content-Types are explicitly not supported.

Installing the HTTP Gateway Servlet

This chapter describes how to install the HTTP Gateway servlet into several environments. The HTTP Gateway servlet is distributed in a WAR (Web ARchive) file called "EAi.war". This file supports the Java Servlet standard and is actually a JAR file with a different extension.

The following shows the directory structure and contents of the file:

```
WEB-INF/
WEB-INF/classes/
WEB-INF/classes/*.class
WEB-INF/config/
WEB-INF/config/httpgateway.cfg
WEB-INF/lib/
WEB-INF/lib/jclient3.jar
WEB-INF/logs/
WEB-INF/web.xml
WEB-INF/error.xml
example.html
debug.html
```

File/Directory	Description
WEB-INF/classes	Contains the Java class files for the servlet.
WEB-INF/config	Contains a sample properties file for the servlet with comments for each of the possible properties (see the Properties File section).
WEB-INF/lib	Contains the JClient3 JAR file which the servlet requires. Servlet engines that support the servlet standard will automatically include any files in this directory in the CLASSPATH.
WEB-INF/logs	An empty directory. This is normally where the servlet log files will go.
WEB-INF/ web.xml	This file is used to configure the servlet initialization parameters for the servlet (see the *Servlet Initialization* section). Servlet engines that suport the servlet standard parse this file and pass the parameters to the servlet.

File/Directory	Description
WEB-INF/ error.xml	The default error template used when XML error output is enabled. (See *Enabling XML Output* section)
example.html	An HTML page that runs the servlet.
debug.html	An HTML page that runs the servlet and causes it to return debug information.

In the following sections, <machine-name> in a URL reference indicates the DNS name or IP address of the machine on which the servlet engine is running.

Tomcat

This section shows how to install the HTTP Gateway servlet into the Tomcat environment. In this example, Tomcat is installed on a Windows machine in the directory D:\jakarta-tomcat.

Note: The setting for the JAVA_HOME environment variable used in Tomcat startup is derived from the javacmd property in the JI envmgr.cfg file. The value of the general JAVA_HOME environment variable is not used.

1. Either manually or automatically extract the contents of the EAi.war file as follows:

   Automatically:

   Copy the EAi.war file into the `webapps` directory, then restart Tomcat.

   The EAi.war file is automatically extracted to the `webapps` directory.

   ```
   D:
   cd jakarta-tomcat\webapps
   copy ..\..\lib\EAi.war .
   ```

   Manually:

   Copy the EAi.war file into the `webapps` directory, create an `EAi` subdirectory and manually extract the contents of EAi.war into it.

   ```
   d:
   cd \jakarta-tomcat\webapps
   copy ..\..\lib\EAi.war .
   mkdir EAi
   ```

```
cd EAi
jar -xvf ..\EAi.war
```

2 With a text editor, edit the web.xml file (in the `WEB-INF` directory) and set the servlet initialization properties as required (see the Servlet Initialization section for more details). The properties to be edited are in the <init-param> sections. The following example shows the <init-param> sections that pertain to the servlet:

> **Note:** Some of these parameters are initially "commented out". Any parameters that are changed must be "uncommented" prior to running the servlet. (Comments begin with "<!--" and comments end with "-->".)

```
<!-- The HTTP Gateway Servlet installation directory. -->
<!--
<init-param>
<param-name>installDir</param-name>
<param-value></param-value>
</init-param>
-->
<!-- The path to the HTTP Gateway Servlet properties file. -->
<init-param>
<param-name>propertiesFile</param-name>
<param-value>httpgateway.cfg</param?value>
</init-param>
<!-- The logging level for the HTTP Gateway Servlet -->
<init-param>
<param-name>logLevel</param-name>
<param-value>0</param-value>
</init-param>
<!-- The log directory for the HTTP Gateway Servlet. This value must be an
absolute directory path (e.g. /http/logs). If this value is not set, log
files go to the WEB-INF/logs directory.
-->
<!--
        <init-param>
            <param-name>logDir</param-name>
            <param-value></param-value>
        </init-param>
-->
```

3 Edit the WEB-INF\config\httpgateway.cfg file as necessary (see the Properties File section for more details).

4. Re-start Tomcat.
5. The URL to run the servlet through Tomcat is:
   http://<machine-name>:<tomcat-port>/EAi/httpgateway

Servlet Initialization

When the servlet is first instantiated by a servlet engine, an initialization procedure is run. This procedure does the following:

1. Reads the initialization parameters (InitParameters) set by the servlet engine and passed to the servlet. These parameters are usually set in a configuration file for the servlet engine. The possible parameters are:

propertiesFile	The path to the properties file for the servlet. If just a file name (with no path information) is given for this property, the file is assumed to be in the "<installDir>/WEB-INF/config" directory. If this property is not given, the servlet will look for a file called "httpgateway.cfg" in the "<installDir>/WEB-INF/config" directory.
installDir	The absolute path of the installation directory for the servlet. This parameter may also be set in the properties file. Normally, this parameter is automatically determined by the servlet, therefore it does not need to be set by the user, although depending upon the servlet engine, this parameter may need to be set.
logLevel	The initial value of the log level to be used by the servlet. This parameter may also be set in the properties file. The possible values are 0 through 3. If this parameter is left undefined (in both the web.xml file and the properties file), no logging will occur. Refer to the "HTTP Gateway Error Messages" on page 40 section for more information.

The servlet checks if the file pointed to by the propertiesFile property exists and if it does not, a message is logged to the servlet engine (e.g. Tomcat) log file. The servlet also checks if the directory pointed to by the installDir is actually a directory and if it is not, a message is logged to the servlet engine log file. In any case, the servlet continues to run.

2. Reads and processes data found in the properties file.
3. Sets up and initializes logging. If a log file name is not set in the properties file, the log file name will be “httpgateway.log”. If the installDir property is not set, no logging will take place.

The Properties File

The HTTP Gateway properties file contains configurable properties that govern the servlet behavior. The path to this file is determined by one of the following, in order of precedence:

1. The <propertiesFile> servlet property (see Servlet Initialization).
2. Look for the file httpgateway.cfg located in the <installDir>/config/ directory.

Note: The properties file is optional. If there is no file, default values are used.

The properties included in this file are:

Property	Description
installDir	The absolute path of the installation directory for the servlet. This parameter may also be set in the properties file. Normally, this parameter is automatically determined by the servlet, therefore it does not need to be set by the user, although depending upon the servlet engine, this parameter may need to be set. **Note:** *This property is normally set once, usually during Servlet initialization. Once set, it is set for the life of the Servlet instance and cannot be dynamically changed*.

Property	Description
logLevel	The initial value of the log level to be used by the servlet. This parameter may also be set in the properties file. The possible values are 0 through 3. If this parameter is left undefined (in both the web.xml file and the properties file), no logging will occur. Refer to the "HTTP Gateway Error Messages" on page 40 section for more information. **Note:** This property is normally set once, usually during Servlet initialization. Once set, it is set for the life of the Servlet instance and cannot be dynamically changed.
jclient3Timeout	In seconds. (Default = 60)
jclient3Slope	The rate (in seconds) at which to increase delay between connection retries. (Default = 2)
jclient3InitialDelay	The initial delay (in seconds) between connection retries. (Default = 0)
jclient3MaxRetries	The maximum number of times to retry connections for this JClient3 service. (Default = 10)
useEncryption	Specifies if the data between the HTTP Gateway servlet and webMethods JI should be encrypted (default NO).
cipherSuite	Specifies the cipher suite to be used if encryption is enabled. See the JClient3 documentation for more information.
environmentManager	Client-specified value for the webMethods JI Environment Manager.
serviceName	Client-specified value for the webMethods JI service name.

Property	Description
methodName	Client-specified value for the webMethods JI method name.
sessionName	Client-specified value for the service session name.
stopSession	Stops the client-specified session.
SiebelMode	When set to “Y” the servlet looks for Siebel information in the XML data that is received. Default is set to No. See the section “Message Conventions” on page 18 for more details.
xmlStart	The text string that indicates the start of XML data in the body of the HTTP request. The default value is “<?xml”. This parameter is useful for the situation where the client sends XML data in the body of the HTTP request, but the data does not start with “<?xml” and the data is not encoded as name/value pairs.
xslIn	See EA_XSL_IN description in the webMethods JI Client Options section.
xslInURI	See EA_XSL_IN_URI description in the webMethods JI Client Options section.
xslOut	See EA_XSL_OUT description in the webMethods JI Client Options section.
xslOutURI	See EA_XSL_OUT_URI description in the webMethods JI Client Options section
xmlErrorOutput	Specifies if error messages should be in XML format (default NO). See the *Error Messages* section for more details.

Property	Description
xmlErrorSendNormal	If set to “Y”, XML error messages will be sent with an HTTP status code of “200 OK”. The default action (No) is to send XML error messages with an HTTP status code of 500.
xmlErrorTemplate	If XML error output is enabled, this specifies the XML template for the error message. See the *Error Messages* section for more details.
xmlErrorTemplateURI	If XML error output is enabled, this specifies the XML template URI for the error message. See the *Error Messages* section for more details.

Message Conventions

Typically, an HTTP client will perform the following steps when communicating with the HTTP Gateway servlet:

1. Connect to the servlet, using the designated port number, IP Address, and URL.
2. Send the Request message.
3. Read back the Response message.

HTTP clients will indicate the webMethods JI environment manager, service name, method name, and other options by setting the appropriate client options in the request message.

Setting Client Options

HTTP clients will set webMethods JI client options to specify values for environment manager, service name, method name, session name, etc that govern how the HTTP Gateway servlet will handle the request. These client options are also included in the file reference by the URL and/or the configuration file for the servlet (see the Properties File section for more information). See the section on Specifying Client Options for the precedence used to set webMethods JI client options. Client options that are not specified in the request message will result in the value from the configuration file being used.

The following are the webMethods JI client options:

Property	Description
EA_ENVMGR	The IP address and port number of the environment manager to connect to. If this option is not specified, the `environmentManager` property from the configuration file will be used.
EA_SERVICE	The name of the webMethods JI service to open. If this option is not specified, the `serviceName` property from the configuration file will be used.
EA_METHOD	The name of the method to invoke. If this option is not specified, the `methodName` property from the configuration file will be used.
EA_SESSION	This will be used to indicate if the client wishes to open a connection to a session-based webMethods JI service. The value of this field will be the session name, or blank if no session is desired. See the section entitled "Sessioning Support" on page 39 for more information.
EA_STOP_SESSION	This is used to end a connection to a session-based webMethods JI service. The value of this field is Y to enable the EA_STOP_SESSION option, or N to disable it.
EA_TIMEOUT	In seconds.
EA_SLOPE	The rate (in seconds) at which to increase delay between connection retries.
EA_INITIAL_DELAY	The initial delay (in seconds) between connection retries.
EA_MAX_RETRIES	The maximum number of times to retry connections for this JClient3 service.

Property	Description
EA_USE_ENCRYPTION	This is used to indicate (Y/N) if the data between the HTTP servlet thread and webMethods JI should be encrypted. If this option is not set, the `useEncryption` property from the configuration file will be used.
EA_CIPHER_SUITE	Specifies which cipher suite is to be used. Only valid if encryption is enabled.
EA_CLIENTID	Used by the client to identify itself; beneficial for logging purposes. This value is used when setting the client name property for the thread. (If the EA_CLIENTID option is not set, the client name property defaults to `HTTPGatewayServletClient_<id>`, where `<id>` is the thread ID of the servlet thread.) The client name will appear in the Client Name column of the System Monitor's Details and Clients views.
EA_REMOTE_PATERM_ID	Remote PATerm identifier. See the Protocol Agents chapter in the *webMethods JI User's Guide* for more information.
EA_XSL_IN	This is used to specify a XSL style sheet as string data. This style sheet is used to transform input in EA_XML fields at the webMethods JI server. It defines the XSL data in string format. If this option and the next option are not specified, the `xslIn` or `xslInURI` property from the properties file will be used. If none of these options are set, there will be no translation applied to the input XML data.

Property	Description
EA_XSL_IN_URI	This is used to specify a XSL style sheet as a URI. This style sheet is used to transform input in EA_XML fields at the webMethods JI server. It defines a reference to the XSL data via a URI. If this option and the previous option are not specified, the `xslIn` or `xslInURI` property from the properties file will be used. If none of these options are set, there will be no translation applied to the input XML data.
EA_XSL_OUT	This is used to specify a XSL style sheet as string data. This style sheet is used to transform output data at the webMethods JI server (only if "service output in XML format" is enabled). It defines XSL data in string format. If this option and the next option are not specified, the `xslOut` or `xslOutURI` property from the properties file will be used. If none of these options are set, there will be no translation applied to the output XML data.
EA_XSL_OUT_URI	This is used to specify a XSL style sheet as a URI. This style sheet is used to transform output data at the webMethods JI server (only if "service output in XML format" is enabled). It defines a reference to the XSL data via a URI. If this option and the previous option are not specified, the `xslOut` or `xslOutURI` property from the properties file will be used. If none of these options are set, there will be no translation applied to the output XML data.

Property	Description
EA_XML_START	The text string that indicates the start of XML data in the body of the HTTP request. The default value is "`<?xml`". This is useful for a situation in which the client sends XML data in the HTTP request's body, but the data does not start with "`<?xml`" and the data is not encoded as name/value pairs.
EA_XML_ERROR_OUTPUT	Specifies if error messages should be in XML format (default NO). See the *Error Messages* section for more details.
EA_XML_ ERROR_SEND_NORMAL	If set to "Y", XML error messages will be sent with an HTTP status code of "200 OK". The default action (No) is to send XML error messages with an HTTP status code of 500.
EA_XML_ERROR_TEMPLA TE	If XML error output is enabled, this specifies the XML template for the error message. See the *Error Messages* section for more details.
EA_XML_ERROR_TEMPLA TE_URI	If XML error output is enabled, this specifies the XML template URI for the error message. See the *Error Messages* section for more details.
versionMode	This property tells the Gateway which operational mode to use if a given request does not specify the mode. The legal values are "4_0" [JI 4.x mode] or "3_x" [JI 3.5 compatibility mode][default].
xmlInVar	If set, this property specifies the name of the method input variable for which the body XML should be mapped. In this case, EA_XML is over-ridden.

Property	Description
xmlOutVar	If set, this property specifies the name of the method output variable which will contain the response XML to be returned to the client. In this case, EA_XML is not used to contain the return XML.
EA_VERSION	Tells the gateway how to process a request. Legal values are “4_0”, and “3_x”. • If the value is “3_x”, the gateway operates exactly as it did under JI 3.5. • If the value is “4_0”, the gateway operates under the JI 4.x rules of processing.
EA_INVAR	When the EA_VERSION is set to “4_0”, this parameter provides the name of the method input variable that should be populated with the XML data in the body of the HTTP request. There is no default value, unless specified in the gateway configuration file.
EA_OUTVAR	When the EA_VERSION is set to “4_0”, this parameter provides the name of the method output variable that will contain the XML to be returned to the requester. There is no default value, unless specified in the gateway configuration file.

Specifying Client Options

HTTP client developers have the choice of specifying client options. Specifying client options is done in any and all of five ways:

- In the message body. This method only applies to POST type requests, as GET type requests have no message body.
- In the arguments of the URL portion of the request line within the request message.
- In the URI portion of the request line within the request message. The reference in the path specified in the URL is assumed to be a reference to a properties file that is to be used for this request.

- In the user-defined properties in the message header.
- The servlet properties file processed when the servlet is initialized.

The order shown here is the precedence (highest to lowest) that is used to specify the value of client options if an option is set in more than one way. That is, a value set in the message body of the request will be used regardless of what value was set in by any other means. The specifics of each method of specifying client options are discussed in the next section: Message Encoding.

Message Encoding

HTTP messages use standard ASCII encoding. There are two types of HTTP messages: request messages (sent from the client to the servlet) and response messages (sent from servlet back to client).

HTTP Request Messages

HTTP request messages contain the sections:

- Request line
- Header(s)
- Blank line with CR/LF
- Message body (POST method only)

Request Message Request Line

The request line is composed of the following elements:

```
<request method> <requested URL> <HTTP-version>
```

Request Method

As previously stated, the HTTP Gateway servlet will only support two request methods:

- GET type requests (also referred as “Simple-Requests”) have no message body. All message data will be contained entirely within request line and headers.
- POST type requests (also referred to as “Full-Requests”) may contain input data within the message body as well as within the request line and headers.

Requested URL

The form of the requested URL field is:

```
<servlet realm/zone>/[property file reference]/[?query]
```

where:

- <servlet realm/zone> is the portion is used by the web server to locate the servlet.
- <property file reference> references a file located relative to the root of the servlet.
- <query> is Optional data in the form “name1=value1&name2=value2…”.

Property File Reference

The URI file defined by the <property file reference> will contain webMethods JI properties (service, method etc) to be used by the current request. Properties in this file override the servlet properties file, but not properties defined with other options noted below. The file must be placed in or under the <Install Dir>/WEB-INF directory, where <Install Dir> is the directory in which the servlet is installed. If the property file does not exist, an error will be returned in the response to the request.

An example URL with a property file reference:

```
http://mymachine.mycompany.com/EAi/httpgateway/props.cfg
```

Query

The query portion of the URL can specify client options (see Client Options) or input data for an webMethods JI service method in the form of simple name value pairs. To pass XML formatted data to the service method, use the name “EA_XML” and set the value to the XML data. Note that special characters need to be encoded per the standard URL encoding conventions (spaces converted to the “+” character and other special characters and 8 bit values converted to their hexadecimal equivalent prefaced with the “%” character).

An example URL with optional data:

```
http://mymachine.mycompany.com/EAi/httpgateway?param1=value1
&param2=value2
```

Request Message Headers

The form for request header lines is:

```
Name: <value>
```

Note that a colon, a single white space and finally the value follow the name. Header lines can be used to specify client options (see Client Options).

Request Message Body

For POST type requests, message data and optionally client options may be embedded in the message body portion of the HTTP request.

The body can be in one of two formats:

- XML data
- name/value pairs

> **Note:** The servlet only recognizes these 2 formats for the body of the request, therefore the Content-Type of the request is only relevant as described in the following sections.

The request's body may contain a mixture of these two types of data, but all name/value pairs must be first, followed by the XML data. Any name/value pair after the XML data is ignored.

XML Data Format

This format is only recognized if the Content-Type of the request is not "application/x-www-form-urlencoded". The XML data format consists of just the XML data. XML data is recognized by looking for the string specified by the xmlStart property (see the Properties File section for more details). If XML data is detected, it is sent to the method in a field called "EA_XML".

Name/Value Pairs Format

This format is as follows:

```
<name>=<value>[<delimiter>]...[more name/value pairs]
```

The value of <delimiter> depends on the Content-Type of the request:

Content-Type	Delimiter
application/x-www-form-urlencoded	Ampersand ('&')
All other types	Semi-colon (';')

An example:

```
EA_SERVICE=MyService;EA_METHOD=MyMethod
```

```
HTTP Response Messages
```

HTTP response messages contain the following sections:

- Status Line
- Header(s)
- Blank line with CR/LF
- Message body

Response Message Status Line

The status line is composed of the following elements:

```
<HTTP-version> <Status-code> <Status code description>
```

Each element of the status line follows the conventions specified in the HTTP/1.0 protocol definition (RFC 1945). A successfully processed request returns a response status code of “200 OK”. An unsuccessfully processed request returns a status code of “500 Internal Server Error” and the response message body contains the error message.

Response Message Headers

No webMethods JI specific information will be contained in the response header lines.

Response Message Body

Data returned from a successful service method invoke, or exception data from an unsuccessful service method invoke, is returned in the response message body. If the method invoke is successful, the data is returned as either XML, or plain text, depending on whether the “XML: Service output in XML format” option is set for the service. If the option is set, the servlet will extract the EA_XML field from the message that was returned from the method invoke.

If the option is not set, the data in the message returned from the method invoke is sent as plain text in the following format:

```
Element 0:
     Name = <name>, Value = <value>
.
.
.
Element N:
     Name = <name>, Value = <value>
```

Error Messages

Error message format depends on which servlet engine being used. Typically, the message will contain some sort of HTML formatting, followed by the error message from the servlet.

The following example is an error message returned by Tomcat:

```
<h1>Error: 500</h1>


<h2>Location: /httpgateway/servlet/httpgateway/dummy.cfg</h2>URL defined
configuration file not found (D:\jakarta-tomcat\webapps\httpgateway\WEB-
INF\dummy.cfg).
```

The following sections explain how to cause the servlet to send error messages in XML format.

Enabling XML Output

To cause the servlet to send error messages to the client in XML format, set either the EA_XML_ERROR_OUTPUT client option or the xmlErrorOutput property to "Y". This will cause the servlet to send the error message embedded in XML in the response message body.

Specifying the XML Error Message Format To specify the format of the XML error message, the servlet will parse a user defined error template (see the following section for the format of the template).

To specify the template, use one of the following client options or property file properties:

- EA_XML_ERROR_TEMPLATE
- EA_XML_ERROR_TEMLATE_URI
- xmlErrorTemplate
- xmlErrorTemplateURI

The xmlErrorTemplate and EA_XML_ERROR_TEMPLATE values directly specify the template. The xmlErrorTemplateURI and EA_XML_ERROR_TEMPLATE_URI values specify a URI reference to the template.

The URI can be in either a normal URI format (e.g. "http://<file path>") or it can be the full path to a file on the machine (e.g. "/templates/error.xml"). The servlet installer file contains a default file (error.xml) that is installed in the WEB-INF directory. If none of the template parameters are set, this file is used as the template to format error messages.

XML Error Template Format The XML error template is simply well-formed XML with the string @ERRORTEXT inserted where the error message should be placed.

For example, if the template was as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<error>
@ERRORTEXT
</error>
The error message the client receives (using the example error from above)
would be:
<?xml version="1.0" encoding="ISO-8859-1"?>
<error>
URL defined configuration file not found
   (D:\jakarta-tomcat\webapps\httpgateway\WEB-INF\dummy.cfg)
</error>
```

Siebel XML Support

Siebel VBC (Virtual Business Component) XML Interface

The webMethods JI HTTP Gateway Servlet interacts with, and provides an interface to, the Siebel XML Adapter. The Siebel XML Adapter is a general-purpose Siebel adapter that uses HTTP to send XML documents between the Siebel application and external applications. The Siebel XML Adapter sends a standard set of XML messages to the webMethods JI HTTP Gateway Servlet to handle initialization, query, delete, update, pre-insert, and insert Siebel messaging functions. webMethods JI responds with a specific XML format for each of these operations.

Siebel VBC XML Formats

A VBC within Siebel is defined with a set of fields. These fields must be supported by the external system, in this case the webMethods JI. The VBC XML integration supports a defined set of operations (init, query, delete, update, pre-insert and insert) along with a defined set of XML requests and replies. The following sections describe the formats for each operation and include XML examples. In the following examples, the buscomp tag defines the name of the VBC as defined in Siebel and remote-source defines the name of the external data source.

For the XML examples, the VBC name (or buscomp) is “JCDA FINCORP Account VBC NC”, and the remote-source is “webMethods JI”.

The VBC for the examples used in this chapter was defined with the following fields: Account Number, Amount, Date, Description, and Detail.

Initialization (Init)

When an applet within Siebel is to be displayed, and the content of the displayed data is represented by a VBC, the VBC will send an initialization message to the webMethods JI HTTP Gateway Servlet requesting the fields that are supported. webMethods JI then replies with a set of data fields that it supports. These data fields must match what is defined in the VBC though additional data fields can be included.

The request XML, or the message to the webMethods JI has the following format.

```
<?Siebel-Property-Set EscapeNames="true" ?>
<siebel-xmlext-fields-req>
      <buscomp id="1">
            JCDA FINCORP Account VBC NC
      </buscomp>
      <remote-source>
            webMethods JI
      </remote-source>
</siebel-xmlext-fields-req>
```

The response to this request includes the names of the fields that will be returned by the webMethods JI. The webMethods JI must support, as a minimum, the fields defined within the VBC. The returned field names must also match the field names exactly (case sensitive) as they are defined within the VBC.

Therefore, the response to the init request for this example is:

```
<siebel-xmlext-fields-ret>
      <support field="Account Number"/>
      <support field="Date"/>
      <support field="Description"/>
      <support field="Detail"/>
      <support field="Amount"/>
</siebel-xmlext-fields-ret>
```

Additional fields can be returned, but they are ignored by the VBC.

Query

The query request follows the init request. This is a request for the external system to return the data to be handled by the VBC. A single “row” of data or multiple “rows” can be returned. The query will contain match tags that

represent the “key” for the query. If multiple “key” fields are defined within the VBC then multiple match tags will be generated within the XML. If multiple match tags exist they should be “And-ed” together to form the query.

```
<?Siebel-Property-Set EscapeNames="true" ?>
<siebel-xmlext-query-req>
      <buscomp id="1">
            JCDA FINCORP Account VBC NC
      </buscomp>
      <remote-source>
            webMethods JI
      </remote-source>
      <match field="Account Number">
            0585-234780
      </match>
</siebel-xmlext-query-req>
```

The response to the query is the row(s) of data to be displayed by the VBC.

In this example the following XML message was returned.

```
<siebel-xmlext-query-ret>
      <row>
            <value field="Account Number">1234</value>
            <value field="Amount">123.00</value>
            <value field="Date">12/12/01</value>
            <value field="Description">Check</value>
            <value field="Detail">101</value>
      </row>

      <row>
            <value field="Account Number">1234</value>
            <value field="Amount">123.00</value>
            <value field="Date">12/12/01</value>
            <value field="Description">Check</value>
            <value field="Detail">101</value>
      </row>

      <row>
            <value field="Account Number">1234</value>
            <value field="Amount">123.00</value>
            <value field="Date">12/12/01</value>
            <value field="Description">Check</value>
            <value field="Detail">101</value>
      </row>
```

```
</siebel-xmlext-query-ret>
```

Delete

The delete operation occurs when a user within Siebel selects a row within the VBC and deletes that row. The following XML request is sent to the webMethods JI. All fields that contain values are included in the XML request to identify the row (record) to be deleted in the webMethods JI. Only one row (record) can be deleted at one time.

```
<?Siebel-Property-Set EscapeNames="true" ?>
<siebel-xmlext-delete-req>

      <buscomp id="1">
            JCDA FINCORP Account VBC NC
      </buscomp>

      <remote-source>
            webMethods JI
      </remote-source>

      <row>
            <value field="Account Number">1234</value>
            <value field="Detail">101</value>
            <value field="Amount">25.00</value>
            <value field="Date">12/12/01</value>
            <value field="Description">ATM Fee</value>
      </row>

</siebel-xmlext-delete-req>
 The response to a delete request is an empty reply.
<siebel-xmlext-delete-ret>
</siebel-xmlext-delete-ret>
```

Update

When a user within Siebel makes a change to a data field within a record represented by a VBC, an update request is generated.The update request contains tags for all fields, and an attribute to identify if the value of the field changed. In this example the description field was changed by the user and the following XML request is generated.

```
<?Siebel-Property-Set EscapeNames="true" ?>
<siebel-xmlext-update-req>
```

```
    <buscomp id="1">
          JCDA FINCORP Account VBC NC
    </buscomp>
    <remote-source>
          webMethods JI
    </remote-source>
    <row>
          <value changed="false"
                field="Account Number">1234</value>
          <value changed="false"
                field="Detail">101</value>
          <value changed="false"
                field="Amount">123.00</value>
          <value changed="false"
                field="Date">12/12/01</value>
          <value changed="true"
                field="Description">Overdraft Fee</value>
    </row>
</siebel-xmlext-update-req>
```

The response to the update request can be an empty reply or the reply can contain updates to fields that may have been changed by the external system based on the changes within Siebel. For example, if the description is changed by the Siebel user to “Overdraft Fee” the external system may automatically set the Amount field to $25.00.

```
<siebel-xmlext-update-ret>
    <row>
        <value field="Amount">25.00</value>
    </row>
</siebel-xmlext-update-ret>
 An empty response would not contain the row definitions.
```

Pre-Insert

When a user within Siebel creates a new record represented by a VBC, the VBC sends a pre-insert message to the webMethods JI HTTP Gateway servlet. The pre-insert is used so that the webMethods JI can set default values in the new record being created. It is important that the webMethods JI only supplies default values and does not create a new record at this point. If the user decides to cancel the “New Record” request after the pre-insert request is sent from Siebel no follow up message is sent to the webMethods JI.

The format of the pre-insert XML request is:

```
<?Siebel-Property-Set EscapeNames="true" ?>
```

```
<siebel-xmlext-preinsert-req>
    <buscomp id="1">
        JCDA FINCORP Account VBC NC
    </buscomp>
    <remote-source>
        webMethods JI
    </remote-source>
</siebel-xmlext-preinsert-req>
```

The format of the reply is a list of default field values.

```
<siebel-xmlext-preinsert-ret>
    <row>
        <value field="Amount">25.00</value>
    </row>
</siebel-xmlext-preinsert-ret>
```

If there are no default values the external system can reply with an empty reply message (no row definitions).

Insert

An insert request is generated by the VBC when the user within Siebel commits the record. The following XML request will be generated:

```
<?Siebel-Property-Set EscapeNames="true" ?>
<siebel-xmlext-insert-req>
    <buscomp id="1">
        JCDA FINCORP Account VBC NC
    </buscomp>
    <remote-source>
        webMethods JI
    </remote-source>
    <row>
        <value field="Detail">120</value>
        <value field="Account Number">1234</value>
        <value field="Amount">1000.00</value>
        <value field="Date">12/12/02</value>
        <value field="Description">Check</value>
    </row>
</siebel-xmlext-insert-req>
```

The response to the insert request is an empty reply.

```
<siebel-xmlext-insert-ret>
```

```
</siebel-xmlext-insert-ret>
```

Error Conditions

If an error occurs in the external system an error reply is returned. All tags within the status message are optional. The format of the status message is:

```
<siebel-xmlext-status>
    <status-code>120</status-code>
    <error-field>Description</error-field>
    <error-text>Description cannot be ERROR</error-text>
</siebel-xmlext-status>
```

Processing the Siebel Requests

The HTTP Gateway Servlet processes requests from Siebel by using property files that contain all the required information that is necessary to process a request (init, query, etc.). The property file is specified by the "property file reference" portion of the URL in the HTTP request that is received by the servlet. An example URL:

```
http://machine.company.com/EAi/httpgateway/config/Siebel.cfg
```

In this example, the property file reference is "config/Siebel.cfg". The servlet would look for the file "Siebel.cfg" in the directory "config" underneath the directory in which the servlet is installed. The servlet processes any property file that is referenced in the URL.

Siebel properties may be set in the *httpgateway.cfg* file. However, the properties set in the httpgateway.cfg file should be considered default or static values, to be used only if no properties were defined in the "property file reference" portion of the URL in the HTTP request. When a property file is referenced in the URL of the request, the values in that file override the values set in the *httpgateway.cfg* file.

Property File Parameters

The property file contains parameters that enable the servlet to map a Siebel request (init, query, etc.) to webMethods JI methods or default replies. Each request has its own set of parameters.

> **Note:** When defining the parameters in the property file, the "****" in the parameter names as shown in the table should be replaced by the name of the request from the following list:

- Init

- Query
- Delete
- Update
- PreInsert
- Insert

The parameters are:

Parameter	Description
Siebel****Service	The webMethods JI service to be opened.This parameter is optional. If it is not defined, either the EA_SERVICE (client option) or the serviceName (properties) parameter will define the service.
Siebel****Method	The webMethods JI method to be invoked. If this parameter is defined, the Siebel****Error parameter must also be defined and the Siebel****DefaultResponse parameter should not be defined.
Siebel****DefaultResponse	The response to be sent back to Siebel. This parameter should *not* be defined if the Siebel****Method parameter has been defined.
Siebel****DefaultResponseURI	Same as above, except the parameter value is the URI of the response.
Siebel****Error	If the method invoke fails, this is the XML that is sent back to Siebel. This parameter must be defined if the Siebel****Method parameter has been defined.
Siebel****ErrorURI	Same as above, except the parameter value is the URI to the XML.
Siebel****InXSL	The XSL data that will be used by webMethods JI to transform the XML method invoke input data. The XSL data will be inserted into the method invoke input data in the field EA_XSL_IN.

Parameter	Description
Siebel****InXSLURI	Same as above, except this is the URI of the XSL data. This value is inserted into the method invoke input data in the field EA_XSL_IN_URI. It must be accessible by the machine on which the method will be invoked (i.e. the machine on which webMethods JI is running).
Siebel****OutXSL	The XSL data that will be used by webMethods JI to transform the XML method invoke output data. The XSL data will be inserted into the method invoke input data in the field EA_XSL_OUT.
Siebel****OutXSLURI	The URI of the XSL data that will be used by webMethods JI to transform the XML method invoke output data. This value is inserted into the method invoke input data in the field EA_XSL_OUT_URI. It must be accessible by the machine on which the method will be invoked (i.e. the machine on which webMethods JI is running).

Siebel Property File Parameters Usage

1 The following rules apply to the property file:

- Each property occupies one line of the file. To continue a parameter value definition from one line to another, terminate the line with a backslash ('\') character.
- The format of each line is <parameter> = <value>. To put the equal sign in a parameter value, precede it with a backslash (“\=”).
- A line that contains only white space or whose first non-white space character is an ASCII '#' is ignored (thus, # indicates a comment line).
- The white space characters preceding the value definition are ignored. Use the backslash escaped characters as described here to get leading white space.

The following special characters are defined by using a backslash escape sequence:

Character	Key Sequence
space	\^ (a backslash and a space ^)

Character	Key Sequence
tab	\t
new line	\n
carriage return	\r
backslash	\ \
equals sign	\=
double quote	\”
single quote	\’
Unicode character	\uxxxx (xxxx is a 4-digit hexadecimal number)

2. If neither the Siebel****Method nor the Siebel****DefaultResponse parameters are defined, the servlet attempts to invoke the method defined by the EA_METHOD (client option) or methodName (properties) parameters, and it will send the XML request in the input data to the method invoke.
3. The Siebel****DefaultResponse and Siebel****Error parameters are sent exactly as defined (i.e. no XSL transformation is performed).
4. Error responses (Siebel****Error/Siebel****ErrorURI parameters) are sent with the HTTP status set to “200 OK”.
5. The Siebel****DefaultResponseURI and Siebel****ErrorURI parameter values are references to a value via a URI. The servlet reads the data referenced by the URI and uses it as the parameter value. No transformations of any kind are performed on the data (e.g., checking for backslash escape sequences); it is used exactly as it was read.
6. The Siebel****InXSLURI and Siebel****OutXSLURI parameter values are sent as-is to the method (i.e. the servlet does not read the data that is referenced by the URI).
7. If the Siebel*XSL* parameters are not defined, the existing XSL parameters will be used. See the “Specifying Client Options” on page 23 section in this chapter for more details.

8. In the error template (Siebel****Error or Siebel****ErrorURI), inserting the text "@ERRORTEXT" causes the servlet to insert a descriptive error message. See the XML Error Template Format section in this chapter for more details.
9. Parameters that are defined using a URI (e.g., Siebel****ErrorURI) can also be defined by using the basic (or non-URI) parameter (e.g., Siebel****Error). Should the user define both of the parameters in the properties file, the non-URI parameter will be used, e.g. Siebel[request type]<parameter name> and Siebel[request type]<parameter name>URI are both defined; the non-URI parameter, Siebel[request type]<parameter name> will be used.

Example Property File

```
environmentManager=localhost:30001
serviceName=m_bankingXML
methodName=Checking_Trans
xmlStart=<?Siebel

#init
SiebelInitDefaultResponse=<siebel-xmlext-fields-ret>\
      <support field="Account Number"/>\
      <support field="Date"/>\
      <support field="Description"/>\
      <support field="Detail"/>\
      <support field="Amount"/>\
</siebel-xmlext-fields-ret>

#query
SiebelQueryMethod=getCheckingTrans
SiebelQueryError=<siebel-xmlext-status>\
      <status-code>120</status-code>\
      <error-text>Query Failed</error-text>\
</siebel-xmlext-status>
SiebelQueryInXSLURI=file:///D:/Siebel/xsl/queryIn.xsl
```

Note that when the Siebel****DefaultResponse parameter is set, the Siebel****Method parameter for this request type is NOT set

Note here that when the Siebel****Method parameter is set, the Siebel****Error parameter is also set. The Siebel****DefaultResponse parameter should NOT be set

Sessioning Support

Service Sessioning

Service session support through the HTTP Gateway servlet is the same as that offered through the standard webMethods JI session functionality. That is, HTTP clients can elect to retain a session with an webMethods JI service by specifying a

session name in the request message. This will result in the webMethods JI service being bound to a session name. Thus, the client can re-access the webMethods JI service by specifying the session name on subsequent request messages. Note that, for each request message from the client, a separate HTTP Gateway servlet thread may process the message. This means that, for each request message, the thread must re-open the connection to the webMethods JI service.

HTTP clients make use of service sessioning by setting the EA_SESSION client option to the name of the session on the request message.

For more information on Service Sessioning, see the *webMethods JI User's Guide*.

HTTP Gateway Error Messages

Error Handling

Whenever an error occurs in processing of a client request message, an error response message is returned to the client (see *HTTP Response Messages*). The error is logged to the servlet log file.

Internal servlet errors will be logged to the servlet log file. Error message format is as follows:

```
ERROR: <error message>
```

Logging

The HTTP Gateway servlet supports four log levels (0-3) which output varying degrees of debug output to a logfile. The directory for the log file(s) is "<installDir>/WEB-INF/logs" where <installDir> is the automatically determined servlet installation dir or parameter passed to the servlet upon initialization, or the parameter in the properties file.

The log level is specified by the <logLevel> servlet initialization parameter, or in the HTTP Gateway servlet properties file. If the <logLevel> parameter is undefined, no logging occurs. If the log level is 0 or 1, the servlet will output one logfile with the name of "httpgateway.log". If the log level is >= 2, the servlet will output a logfile for each servlet thread, in addition to the "httpgateway.log". The name of the log files will be "httpgateway_<id>.log", where <id> is a unique identifier for the servlet thread. This same logfile is also written to by the JClient3 bean. The information below, for each log level, indicates what the JClient3 DebugController equivalent level is.

The following is a brief description of each log level:

0 Critical Errors and exceptions are logged. Corresponds to the JClient3 DebugController level of NONE.

1 Minimal logging. Messages about the servlet starting up or shutting down are logged. Corresponds to the JClient3 DebugController level of BASIC.

2 Increased logging. Configuration file properties are logged. Default servlet property values being used (value was not specified explicitly in configuration file) are logged. The servlet will log something like "Message received from <client>", including the client options and applicable message header contents. Include all name-value pairs that are sent to JClient3 This is needed because the parameters can come from multiple places and we need to know what the final values that are set and sent to webMethods JI. Corresponds to the JClient3 DebugController level of NORMAL.

3 Detailed logging, including inbound and outbound message contents. Include all name-value pairs that are sent to JClient3. Corresponds to the JClient3 DebugController level of DETAILED.

Note: At log level 2 or 3, numerous log files may be created, and each file could grow very large.

Getting Debug Information

If a request is sent to the servlet with the data field of the requested URL starting with "___debug=1", the servlet responds with the servlet version information as described in the Version Information section, along with other information as follows:

```
Init parameters:
     Parameter "<param>" = "<value>"


Config properties:
     Property "<property>" = "<value>"
```

The Init parameters section contains the parameters that were passed to the init() method of the servlet. The Config properties section shows a list of all the servlet's properties.

Scaling

Scaling of the servlet is dependent upon, and controlled by, the specific Java Application Server (JAS) in which the servlet is installed. Depending upon the JAS, different servlet and servlet container properties are available to manage the scaling behavior of the servlet.

Version Information

The version information for the servlet can be retrieved by running the servlet as an application, or by sending a special request.

Run the Servlet as an Application

The servlet has a main() method that prints the version information, therefore the servlet can be run from the command line. Navigate to the directory in which the servlet classes are installed and then run the following command:

```
java -classpath <servlet JAR classpath>;<JClient3 JAR classpath>;.
HTTPGatewayServlet
```

In this example, <servlet JAR classpath> is the path to the servlet.jar file and <JClient3 JAR classpath> is the path to the jclient3.jar file.

An example of the output follows:

```
EAi HTTP Gateway Servlet
  Servlet version:  0500
  JClient3 version: webMethods JI Version 3.5.1.4-0507
```

Send a Special Request to the Servlet

Send a request to the servlet with the data field of the requested URL starting with "__debug=1", and the servlet will respond with the information shown above, along with additional servlet information.

The version information is also returned when the getServletInfo() method in the servlet is called. This is the method, as defined by the Java Servlet standard, that should be called by the servlet engine to get information about the servlet.

Chapter 2. The MQ Gateway

This chapter describes webMethods JI's MQ Gateway, a feature that allows webMethods JI services to integrate with clients through IBM's MQSeries™. MQSeries is a multi-platform messaging and queuing system that provides a communication connection between clients and services that can be running on different operating systems and written with different development environments.

- With the webMethods JI MQ Gateway, you can integrate webMethods JI services with MQSeries clients. This provides the opportunity to access webMethods JI legacy screen data from clients written in any programming language supported by MQSeries, such as Lotus Notes™, Java™, Visual Basic™, C, and C++. These MQSeries clients can then access webMethods JI services from any platform supported by MQSeries.
- Further information about MQSeries can be found on the World Wide Web at http://www.software.ibm.com/ts/mqseries/.

Assumptions

- This document assumes that you are familiar with webMethods JI services. Services are discussed in this chapter only as they relate to or are used by the MQ Gateway. For more information on writing webMethods JI services, see the *webMethods JI User's Guide.*
- This document also assumes that you are familiar with how to write MQSeries client applications. For more information about writing MQSeries client applications, see IBM's *MQSeries Clients* manual at http://www-3.ibm.com/software/integration/mqfamily/library/manualsa/csqzaf/csqzaf26.htm

  and the *MQSeries Application Programming Reference* at http://www-3.ibm.com/software/integration/mqfamily/library/manualsa/csqzal/csqzal56.htm
- This document also assumes that you have purchased and successfully installed MQSeries.

About the MQ Gateway

The MQ Gateway is an webMethods JI application that allows MQSeries clients to retrieve data from legacy applications using IBM's MQSeries messaging software. Because the MQ Gateway receives messages from and sends messages to MQSeries clients, the MQ Gateway functions as an MQSeries service. At the

same time, by communicating with webMethods JI services, the MQ Gateway resembles a traditional webMethods JI JClient3 client. The MQ Gateway is a multi-threaded, stand-alone, server process.

The purpose of the MQ Gateway is simple: to receive a message from an MQSeries client, invoke the webMethods JI service, and return any message data to the MQSeries client that are returned by the webMethods JI service after it has interacted with the legacy application.

Clients that are used in conjunction with the MQ Gateway are not webMethods JI clients, but instead are user-written MQSeries client applications that issue MQSeries API calls to the MQSeries Queue Manager. The Queue Manager then passes a message from the client to the MQ Gateway. Rather than calling webMethods JI messaging API functions, MQSeries clients create messages that use a openly-defined message format that is sent to the MQ Gateway via MQSeries queues.

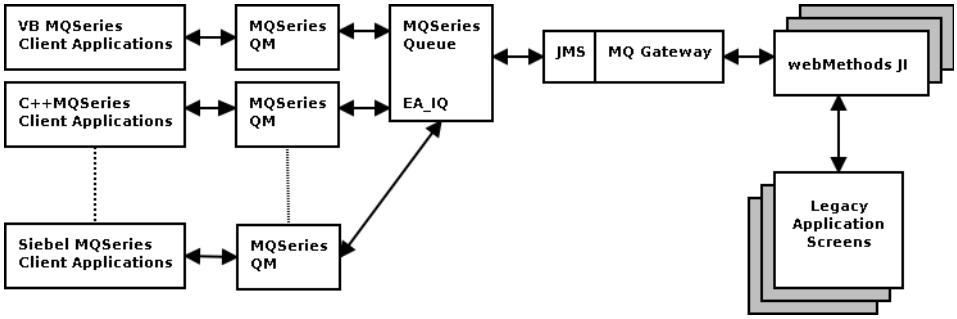


Figure 2. The MQ Gateway

webMethods JI's MQ Gateway then decodes these MQSeries messages and converts them to the message format expected by webMethods JI services. These services operate within the webMethods JI environment and communicate with legacy applications via the webMethods JI protocol agent and screen protocols, such as 3270, 5250, and telnet. After the service has completed communication with the legacy application, the service returns an webMethods JI service message to the MQ Gateway. The MQ Gateway then reformats the service message into the messaging format expected by the MQSeries client and passes the message to the MQSeries Queue Manager, which passes it to the MQSeries client.

The MQ Gateway can run on either the same system or a different system that an MQSeries queue manager is running on. There can be many queue managers in the MQSeries network, but only one will be designated as the MQ Gateway's servicing queue manager, which owns the MQ Gateway's main input queue. The MQ Gateway reads from this queue, listening for request messages from clients. When a request message is received, the MQ Gateway invokes the requested service method and sends the reply back to the MQ Client through the reply queue.

An MQSeries client can be running anywhere on the network. It does not have to be local to a queue manager.

> **Note:** The MQ Gateway does not need to be run from the same machine as the webMethods JI services with which the gateway communicates.

The MQ Gateway makes use of threads and thread pooling. Each thread is called a "worker thread". Each thread reads from the main MQSeries input queue, listening for request messages from clients. When a request message arrives on the input queue, a worker thread reads the message from the queue. This thread acts as an webMethods JI JClient3 client and invokes the requested webMethods JI service method. Once the method completes, the worker thread returns the reply message back to the client, through the specified output queue. (The output queue is specified by the client via the ReplyTo field of the request message header.) Once the worker thread places the reply message on the reply queue, it will again read from the main input queue, waiting for another request message.

MQ Gateway Issues and Restrictions

Installation Requirements The MQ Gateway requires the MQSeries 5.1 or 5.3 Client software to be installed on the same system, plus the MQSeries classes for Java Message Service (JMS) V1.1 (patch MA88 V5.200).

The main input queue, indicated by the `receiveQueueName` property in the configuration file, ***must*** be configured so that the Default Input Open Option is "shared" (not "exclusive").

General The only supported data type for request messages is XML. The XML data must conform to the MapMaker-generated DTD to insure the request message works correctly with the MapMaker generated service, unless custom service code is written to handle the non-conferment XML.

All webMethods JI services that are to be accessed by MQ clients, must be configured to send output in XML format.

The MQ Gateway will not support messages segmented by the sending application.

Unsupported MQSeries Functions The MQ Gateway does not support the following MQSeries functionality:

- MQSeries message groups - Each request message on the input queue is processed as a separate individual request.
- MQSeries Reference messages.
- The "Expiry" (i.e. time-to-live) option in MQSeries messages. These are ignored by the MQ Gateway.

The MQ Gateway does not generate MQSeries Report messages of the following types:

- Exception
- Positive Action Notification (PAN)
- Negative Action Notification (NAN)

MQSeries Configurations

There are numerous ways to configure an MQSeries network and its queues. For example:

- Clients can connect to the MQSeries Queue Manager servicing the MQ Gateway and open the MQ Gateway's main input queue directly.
- Clients can connect to a queue manager running on a remote system and open the main input queue remotely.
- Many MQSeries clients can use the same queue to retrieve their reply messages. The MQSeries administrator must configure the queues and channels appropriately. MQ clients must specify the reply queue name when sending the request message to the MQ Gateway. This requires using the Message ID and Correlation ID fields in the MQSeries message header.
- Each MQSeries client can use its own queue to retrieve the reply messages. The MQSeries administrator must again configure the queue manager appropriately to provide support for one reply queue per client. When the MQ client specifies the reply queue name (using the ReplyTo field in the request message header), it will receive the reply message on that queue.

The MQ Gateway will support these configurations as long as all client request messages get sent to the MQ Gateway's main input queue owned by the queue manager servicing the MQ Gateway, and that a reply queue exists for the reply message.

MQSeries Message Conventions

MQSeries Client Interfaces

MQSeries clients may use a set of MQSeries API calls to access the MQ Gateway. Detailed information about the MQSeries APIs can be found in IBM's *MQSeries Application Programming Guide* located at http://www-3.ibm.com/software/integration/mqfamily/library/manualsa/csqzal/csqzal56.htm

They may also use the JMS API. Information on using the JMS API is located at http://www-4.ibm.com/software/ts/mqseries/library/manualsa/csqzaw04/csqzaw.htm

Certain aspects of MQ Gateway operation will vary depending on whether the MQSeries API or the JMS API is used.

Setting Client Options

MQ clients will set client options to specify things such as environment manager, service name, method name, session name, etc that govern how the MQ Gateway will handle the request. These client options are also included in the configuration file `mqgw.cfg`. Client options specified within a request message (in the message header of a JMS request message, and in the body of a non-JMS request message) will override the corresponding property set in the configuration file. Client options that are *not* specified in the request message will use the value from the configuration file.

Note: MQSeries clients using the JMS API should use either setIntProperty() or setStringProperty() to set the client options, since all options are either of the type integer or string.

The following are the client options:

Property	Description
EA_ENVMGR	The IP address and port number of the environment manager to connect to. If this option is not specified, the default value from mqgw.cfg will be used.
EA_SERVICE	The name of the webMethods JI service to open. If this option is not specified, the default value from mqgw.cfg will be used.
EA_METHOD	The name of the method to invoke. If this option is not specified, the default value from mqgw.cfg will be used.

Property	Description
EA_SESSION	This will be used to indicate if the client wishes to open a connection to a session-based webMethods JI service. The value of this field will be the session name, or blank if no session is desired.
GW_SESSION	This will indicate if gateway sessioning is to be used.
SESSION_TIMEOUT	Amount of time that a gateway session thread will read from the temporary session queue (without receiving a message) before attempting to time out and return itself to the pool of available threads. If this option is not specified, the `sessionTimeout` property from the configuration file
EA_TIMEOUT	In seconds
EA_SLOPE	The rate (in seconds) at which to increase delay between connection retries.
EA_INITIAL_DELAY	The initial delay (in seconds) between connection retries.
EA_MAX_RETRIES	The maximum number of times to retry connections for this JClient3 service.
EA_INPUT_FIELD _NAME	The name that the MQ Gateway gives to the field in the input map passed to the E/A service method. This map contains the input data from the MQSeries client. If this option is not specified, the `inputFieldName` property from the configuration file will be used.

Property	Description
EA_OUTPUT_FIELD _NAME	The name of the field in the return map from the service method from which the MQ Gateway retrieves the output data. If this option is not specified, the `outputFieldName` property from the configuration file will be used.
EA_CLIENTID	Used by the client to identify itself; beneficial for logging purposes. This value is used when setting the client name property for the worker thread. (If the EA_CLIENTID option is not set, the client name property defaults to "`mqgw_<id>`", where `<id>` is the thread ID of the worker thread.) The client name will appear in the Client Name column of the System Monitor's Details and Clients views
LOG_LEVEL	The amount of debug output to be written to the MQ Gateway worker thread's logfile for this client request. Valid values are 0 - 3. If this value is different from the `loglevel` property specified in the configuration file, then the MQ Gateway's worker thread will temporarily set the log level to this value when it begins processing the request, and will set it back to the original log level value when done. This provides a way to debug a client's request without having to restart the gateway.
GW_EXPIRE_TIME	The amount of time the reply message will live on the reply queue before expiring.
EA_USE_ENCRYPTION	This is used to indicate (Y/N) if the data between the HTTP servlet thread and webMethods JI should be encrypted. If this option is not set, the `useEncryption` property from the configuration file is used.
EA_CIPHER_SUITE	Specifies which cipher suite is to be used. Only valid if encryption is enabled.

Using the ReplyTo Field

The ReplyTo field in the message header specifies the name of the queue to which the reply message should be sent. The client must set this field to the name of the queue from which it intends to read the reply message. This field is required, and if it is not set or is invalid, the MQ Gateway will log an error message and continue processing the next message (and thus, the client will not receive a reply from the MQ Gateway).

For MQSeries clients, two fields must be set, the `ReplyToQ` and `ReplyToQMgr`. For JMS clients it is only necessary to set the ReplyTo property.

Ensuring Receipt of Proper Reply Message

When a single queue is used by more than one MQ client for receiving its reply messages, it is very important that client developers take care to ensure each client receives the proper reply message. The MQ Gateway will always set the Correlation ID field on the reply message to the value of the Message ID field from the request message, before sending the reply message back to the client. These fields reside in the message header of the MQSeries message.

To ensure the proper reply message is received, the client must be coded to retrieve the reply message with the proper Correlation ID from the queue. For example, JMS clients must use a message selector on the `JMSCorrelationID` property to ensure they receive the proper reply message. C-based clients must set the MsgId and CorrelId fields properly before issuing their MQGET() call.

Clients must also assure that a unique Message ID is generated by the queue manager for each request message sent to the MQ Gateway. For a C-based client, this means setting the MsgId and CorrelId fields to NULL before sending each request message to the MQ Gateway. This informs the MQSeries queue manager to generate a unique message identifier for that message.

The mqgw.cfg File

The MQ Gateway properties file (mqgw.cfg) contains configurable properties that govern the gateway behavior. A sub-set of the properties listed here apply directly to the underlying JClient3 library.

The properties included in this file are:

Property	Description
queueMgrName	Name of the local MQSeries Queue Manager servicing the MQ Gateway. The default MQSeries queue manager name is used if this is not set.
queueMgrHostName	Host name of the queue manager servicing the MQ Gateway (if remote). If this is not set, the queue manager is assumed to be local. When set, this property, along with the following three properties • queueMgrPort • queueMgrTransportType • queueMgrChannel is used to establish the connection to the queue manager servicing the MQ Gateway. Default value for this field is blank (local queue manager).
queueMgrPort	Port number of queue manager (if remote). Default value for this field is blank (local queue manager).
queueMgrProtocolType	The protocol/transport type to use when connecting to the queue manager (if remote). Default value for this field is blank (local queue manager). **Note:** Within the mqgw.config file make sure the acceptable protocol types "#queueMgrProtocolType=1" is uncommented. The error "**Starting MQGateway with remote MQ manager reports couldn't find mqm.dll**" usually occurs when the mqseries client is installed with the MA88 java and jms patch for mqseries with remote MQ manager.
queueMgrServerChannel	Server channel name for the queue manager (if remote).

Property	Description
adminPort	Port number for MQ Gateway's administration listener socket. This port is used by the ea_mqgwstop and ea_mqgwstatus programs. If the MQ Gateway is unable to open this port, it will fail to start.
receiveQueueName	The queue name for incoming client request messages. Default is EA_IQ.
logfileName	The name and location of the MQ Gateway's logfile. Default is mqgw.log.
loglevel	The level of information to be logged to the MQ Gateway's logfile. Log levels are: • 0 - Only configuration file properties and errors are logged. No log files are generated for the worker threads. • 1 - Minimal logging. • 2 - Moderator logging. Worker threads display detail such as “Message received from <client>”, including the client options and applicable message header contents. • 3 - Detailed logging, including inbound and outbound message contents.
maxThreads	The maximum number of worker threads that are allowed in the MQ Gateway. Default is 1000.
minThreads	The minimum number of threads available in the worker thread pool. Default is 10.
jclient3Timeout	In seconds.

Property	Description
jclient3Slope	The rate (in seconds) at which to increase delay between connection retries.
jclient3InitialDelay	The initial delay (in seconds) between connection retries.
jclient3MaxRetries	The maximum number of times to retry connections for this JClient3 service.
sessionTimeout	In milliseconds (ms) - Amount of time that a gateway session thread will wait for a message from the client before timing out and returning to the pool of available threads. Default is 300000 ms.
idleTimeout	In milliseconds (ms) - Amount of time a worker thread will read from the input message queue (without receiving a message) before timing out. This timeout is used to implement session timeouts and dynamic thread pooling. Default is 10000 ms
threadIncrement	Number of threads to spawn when queue high water mark is reached. Default is 10.
queueBrowseInterval	In milliseconds (ms) - How often the input queue should be browsed to determine the number of input messages. Default is 2000 ms.
monitorInterval	In milliseconds (ms) - How often worker threads (busy vs. non-busy) should be monitored. Default is 2000 ms. Applicable for log level one or higher.

Property	Description
deadLetterQueue	Name of the queue on which the MQ Gateway will put undeliverable reply messages. Default is “ ”: no dead letter queue. If this property is set (i.e. not blank), a queue by this name must exist on the queue manager servicing the MQ Gateway. **Note:** If no dead letter queue is specified and an error occurs while sending a reply message, an error message will be written to the MQ Gateway's logfile, but the reply message itself will be lost.
expireTime	Amount of time the reply message will live on the reply queue before being expired by the MQSeries queue manager. Default 0: no expiration.
inputFieldName	The name that the MQ Gateway gives to the field in the input map passed to the service method containing the input data from the MQSeries client. Normally, “EA_XML” is used here to inform the service code to parse the XML input data, but a different name can be specified to prevent the XML parsing, thus providing the ability to pass the input from the MQSeries client straight through to the service method (default: “EA_XML”).
outputFieldName	The name of the field in the return map from the service method from which the MQ Gateway retrieves the output data. This is normally “EA_XML”, but a different name can be used in the case where the output from the service method is not parsed into XML format. This provides the ability to pass output “as is” from the service method back to the MQSeries client, so the service method can return non-XML formatted data to the MQSeries client.

Property	Description
environmentManager	Default value for the webMethods JI Environment Manager.
serviceName	Default value for the webMethods JI service name.
methodName	Default value for the webMethods JI method name

The JMS Format

The MQ Gateway uses a text-based messaging format that is passed between clients and the MQ Gateway through MQSeries queues. When the MQ Gateway receives a message from an MQSeries client, the message must be in the format described in this section. After the service has communicated with the appropriate legacy application(s) and the service message has been sent to the MQ Gateway, it returns the webMethods JI message in the format described in this section.

Support for JMS and Non-JMS Clients

The MQ Gateway supports JMS clients as well as non-JMS clients. JMS clients make use of the user-defined properties in the message header, to specify client option information such as the Environment Manager, service name, method name, etc. Other MQSeries client's messages do not have access to user-defined properties. The MQ Gateway looks at the request message to determine if it came from a JMS client or a non-JMS client. The JMS_IBM_MSGTYPE property in the request message specifies if the message came from a JMS client. JMS clients must set this property to indicate they are making use of the User Properties field to specify client options. When JMS_IBM_MSGTYPE is set to a value of 65536 the message is from a JMS client, and the MQ Gateway extracts the client options from the User Properties field. If the JMS_IBM_MSGTYPE is blank, or contains some other value, the message is from a non-JMS client.

The MQ Gateway must then extract the options from within the message itself. Because of this, the name-value options must be at the start of the message, with each option separated by a semicolon. The format of a message sent by a non-JMS client will look something like the following:

```
"EA_ENVMGR=localhost
```

```
:30001;EA_SERVICE=banking;EA_METHOD=GetCustInfo;
<XML data here>"
```

Note: Not all supported client options are shown in this example.

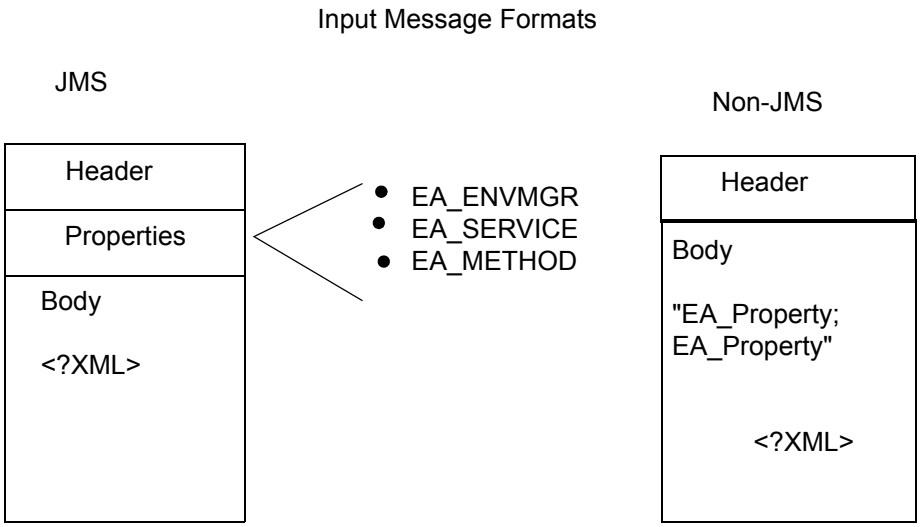


Figure 3. Support for JMS and Non-JMS Clients

Sessioning Support

Service Sessioning

Service session support through the MQ Gateway is the same as that offered through the standard webMethods JI session functionality. MQ clients can elect to retain a session with an webMethods JI service by setting the EA_SESSION client option to the name of the session on the request message. This results in the webMethods JI service being bound to a session name. Thus, the client can re-access that service by specifying the session name on subsequent request messages. Note that, for each request message from the client, a separate MQ Gateway worker thread may process the message. This means that for each request message, the worker thread must re-open the connection to the webMethods JI service.

For more information on Service Sessioning, see the *webMethods JI User's Guide*.

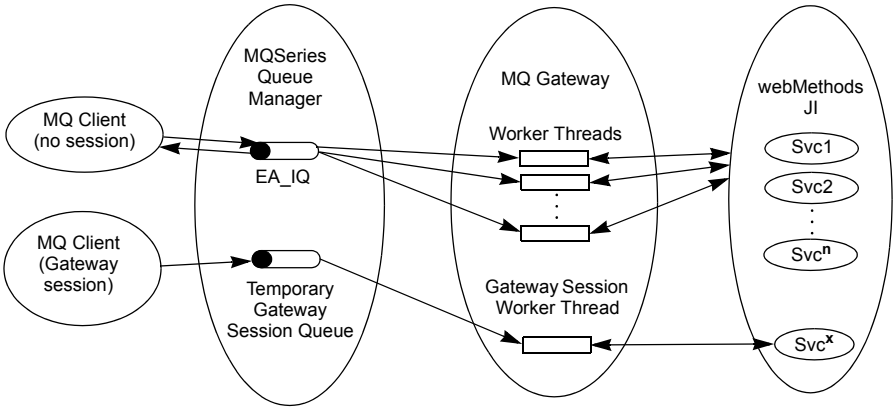


Figure 4. Gateway Sessioning

Gateway sessioning is a way for MQ clients to create a bound session with a worker thread inside the MQ Gateway. Gateway sessioning enables the worker thread to keep its connection with the webMethods JI service open, thus decreasing the amount of time to handle subsequent requests from the client. MQ clients should use this type of sessioning if they intend to issue multiple requests for the same webMethods JI service. Gateway sessioning is enabled by setting the GW_SESSION client option to Y on the request message.

When an MQ client uses gateway sessioning, there is one message queue dedicated for the MQ client to send request messages to the MQ Gateway worker thread. To insure this communication channel between the client and the worker thread remains intact, the MQ client must use the ReplyTo field in the message header properly. After sending its first request message and receiving the reply, the client must retrieve the ReplyTo field from the message header. This field contains the name of the dedicated message queue.

All subsequent request messages throughout the duration of the “session” must be sent to this queue. Note that all reply messages, however, are received from the queue specified by the client in its request messages.

MQ clients that don't make use of sessioning can be serviced by any available worker thread, and can end up accessing any webMethods JI service with the specified name. MQ clients that use gateway sessioning, however, are bound to a specific worker thread (Gateway Session Worker Thread), and a specific webMethods JI service (Svcx). When the client goes away, the thread that serviced the client (Gateway Session Worker Thread) is added back to the pool of available worker threads.

Starting and Stopping the MQ Gateway

Starting the MQ Gateway

To start the MQ Gateway, type the following at command line of the machine on which the gateway has been installed:

UNIX:

```
cd <JI_install_dir>/bin
./ea_mqgwstart.sh
```

> **Note:** To run the MQGateway in the background, use “nohup” and “&” in the command line when launching the MQGateway.

Windows:

```
cd <JI_install_dir>\bin
.\ea_mqgwstart
```

> **Note:** The Gateway does not run as a daemon process in the background; ea_mqgwstart will “tie up” the DOS window from which it was launched.
>
> The MQ Series Queue Manager must be running before starting the MQ Gateway.

Options:

```
ea_mqgwstart [-f config_file] [-i install_dir] [-v] [-h]
```

Parameter	Description
`-f config_file`	Identify name of configuration file. Path is relative to the user's current directory.
`-i directory`	Path to the *<JI_install_dir>* if not executing the command from *<JI_install_dir>/bin*. Path is relative to the user's current directory.
`-n`	Identify level of debug logging: 0 (min) - 3 (max)
`-v`	Displays the current version of the MQ Gateway.

Parameter	Description
`-h`	Displays a help message for this command.

Stopping the MQ Gateway

To shut down the MQ Gateway, type the following at the command line of the machine on which the gateway is running:

UNIX:

```
cd <JI_install_dir>/bin
./ea_mqgwstop.sh
```

Windows:

```
cd <JI_install_dir>\bin
.\ea_mqgwstop
```

Options:

```
ea_mqgwstop [-f config_file] [-i install_dir] [-v] [-h]
```

Parameter	Description
`-f config_file`	Identify name of configuration file. Path is relative to the user's current directory.
`-i directory`	Path to the *<JI_install_dir>* if not executing the command from *<JI_install_dir>/bin*. Path is relative to the user's current directory.
`-n`	Identify level of debug logging: 0 (min) - 3 (max)
`-v`	Displays the current version of the MQ Gateway.
`-h`	Displays a help message for this command.

MQ Gateway Status

To interrogate the MQ Gateway for status information, type the following at the command line of the machine on which the gateway is running:

UNIX:

```
cd <JI_install_dir>/bin
./ea_mqgwstatus.sh
```

Windows:

```
cd <JI_install_dir>\bin
.\ea_mqgwstatus
```

Options:

```
ea_mqgwstatus [-f config_file] [-i install_dir] [-v] [-h]
```

Parameter	Description
`-f config_file`	Identify name of configuration file. Path is relative to the user's current directory.
`-i directory`	Path to the *<JI_install_dir>* if not executing the command from *<JI_install_dir>/bin*. Path is relative to the user's current directory.
`-n`	Identify level of debug logging: 0 (min) - 3 (max)
`-v`	Displays the current version of the MQ Gateway.
`-h`	Displays a help message for this command.

MQ Gateway Error Messages

Error Handling

Errors are returned to MQSeries clients by the MQ Gateway. If an error occurs while the MQ Gateway is processing messages from MQSeries clients or webMethods JI services, a message will be returned to the MQSeries client. 65537 in JMS_IBM_MSGTYPE will be put into the queue from which the client is reading.

Error message format is as follows:

```
Message text: Java exception
```

Error Logging

When the gateway encounters an error while processing the request message it will send a reply message with information regarding the error.

The MQ Gateway also uses a dead letter queue into which reply messages will be put if they can't be sent back to the client. The queue name is specified as the property deadLetterQueue in the properties file. For example, if the MQ Gateway receives an error while trying to send the reply back to the client, resulting in the reply message to not be sent, it will log the error and put the reply message onto the dead letter queue. The system administrator can then retrieve the reply message and forward as necessary. In all cases, the gateway attempts to send a reply message back to the client.

Note: If no dead letter queue is specified, and an error occurs while sending a reply message, an error message will be written to the MQ Gateway's logfile, but the reply message itself will be lost.

Chapter 3. XML and XSLT Support in webMethods JI

XML and XSLT Support Overview

With the release of webMethods JI version 3.5, all webMethods JI client libraries can send and receive XML-formatted data to/from webMethods JI services. XSLT functionality is available to transform input and output XML data.

> **Note:** XML support is **not** provided for C services.

String Size Support

JI 4.6 Java-based client and server software uses Java 17 and supports very long strings up to 2^{64}-1 bytes. Native ActiveX, Standard C and thread-safe C client libraries are unsupported.

XSLT Processor

The Oracle Java API for XML Processing (JAXP) version 1.1 is used as the XSLT processor. This package contains the following files:

- jaxp.jar - Java XML API classes
- crimson.jar - XML parser
- xalan.jar - XSLT transformer

The `crimson.jar` file replaces the `parser.jar` file, formerly used by webMethods JI. To reflect this change, change the `ea_envmgr.lax` file and add `xalan.jar` to the class path.

Invoking a Method and sending Input Data to the Service

To invoke a method sending XML-formatted data, invoke a service message and define the input message using `EA_XML` or `EA_XML_URI`.

Client Input Message - Specifying XML Parsing

XML parsing is implemented on the server side. To specify that the XML data should be parsed, the client **must** use the predefined field names "`EA_XML`" or "`EA_XML_URI`" in the input message. When "`EA_XML`" is in the input message, the value for this field is the XML data in String format. When "`EA_XML_URI`" is in the input message, the value for this field is a reference to the XML document (URI: Uniform Resource Identifier).

When multiple `EA_XML/EA_XML_URI` fields are passed in the input message, the input map should be built as follows (the following example shows three EA_XML fields in input message):

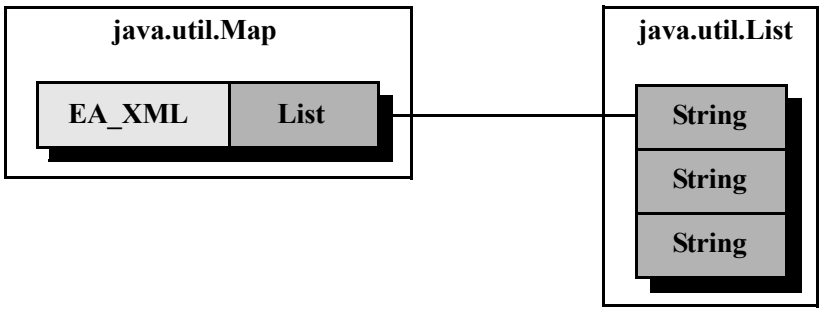


Figure 5. Input map

You may also pass the XML data "as is" to the host application. To force the XML data to be passed on to the host application un-parsed, do **NOT** use "`EA_XML`" or "`EA_XML_URI`" as the field name for the String field containing the XML data. This allows service developers to write custom code to parse the XML data.

Errors Encountered During Parsing

If the XML parser encounters any error while parsing the input XML data contained in the `EA_XML` field, the error will be logged, the parsing of the `EA_XML` field will stop, and the service front-end will continue with the next field in the input data. Thus, if an error occurs while parsing XML data, no input parameters will be set in the map for that `EA_XML` field.

MapMaker Generated DTD

During the client generation process, MapMaker creates a DTD describing method inputs. A stand-alone DTD file is always generated.

An XML file (one for each method of each service) is only generated if "Include Sample Input Data" is checked on the Generate Code dialog (Client tab). The DTD contained within the XML file is an exact copy of the stand alone DTD file, and resides in the same directory as the client code. The XML file name is "`ServiceName_MethodName.xml`".

The client may then use this skeleton XML file and DTD, to create properly formatted XML data to send to the service in an input message. This ensures that the XML will work correctly with the MapMaker-generated map.

To insure that the XML can be parsed into a valid Map/List structure and work correctly with a MapMaker-generated map, the user's XML must conform to the MapMaker generated DTD. To achieve this, the input XML can either:

- Reference the MapMaker-generated DTD (as described above) in the `DOCTYPE` statement.
- Reference a DTD that is a superset/subset of the MapMaker-generated DTD in the `DOCTYPE` statement.
- Conform to the MapMaker-generated DTD (if no `DOCTYPE` specified).

If the user's XML data does not conform to the webMethods JI supplied DTD, or if it contains no DTD at all, then the XML will be parsed according to guidelines outlined in *Parsing Rules*.

Parsing Rules

Overview

These rules govern how an XML is parsed into a map structure containing name/value pairs:

- Every XML element (i.e. "name=value") will result in an entry in a List that is contained within a Map.
- Every repeating XML element at the same level will be contained in a list.
- All elements at the 1st level below the root XML level will be used as inputs to the map.

If formatting elements are used inside a #TEXT entry and the format needs to be preserved, then the #TEXT should be enclosed within a CDATA structure, as follows:

```
<Foo> TEXT1 [!CDATA <EM> B </EM> ] TEXT2 </Foo>
```

Otherwise, #TEXT elements will be parsed into #TEXT and child elements. White space will not be preserved unless a CDATA format is used.

> **Note:** XML processing instructions contained within the input XML (e.g.“<?InstructionText?>”) will be ignored by the parser.

Configurable Parsing Options

Four configurable properties are available to govern how the XML parser operates while parsing XML input messages. These properties appear in the properties table whenever a Service Master node is selected in the tree. They are editable for Java services only.

See the *webMethods JI User's Guide - Chapter 5 Configuration Manager* for information on *Setting Properties for Service Masters*.

The properties are:

- **XMLAttrParsingOptions** - Attribute parsing option determines how XML attributes are handled by the parser.
- **XMLUseValidatingParser** - The validating vs. non-validating parser option specifies whether the XML parser should act as a validating or non-validating parser.
- **XMLNamespaceAware** - This option specifies whether the XML parser should be name space-aware while parsing. If this option is disabled, then duplicating element type names or attribute names in an XML document may result in name conflicts during parsing.
- **XMLOnOutput** - If there is no EA_XML field in the output map from the service, then the XMLOnOutput property is used to determine if the output should be in XML format. When enabled, the entire output map is converted into XML and sent back in one field, called `EA_XML`. When disabled, the entire map is sent back to the client unchanged.

Attribute Parsing Options

The XML parser supports three options as to how to handle XML attributes, by making use of the JService property XMLAttrParsingOptions. The three options for parsing attributes are:

- **Ignore all attributes in the XML input message** - Any attributes encountered while parsing the XML input message will be ignored and not included in the resulting Map/List.
- **Parse attributes into fields in the resulting Map** - Attributes encountered while parsing will be included as fields in the resulting Map/List. These attributes will be treated as sub-elements of the XML tag to which they belong. For example, the following XML:

```
<Person Gender="male">
   <Name> Joe Smith </Name>
```

```
</Person>
```

Treating the "`Gender`" attribute as a sub-element of the `Person` tag, this XML would then be parsed as if it were written as follows:

```
<Person>
   <Gender> male </Gender>
   <Name> Joe Smith </Name>
</Person>
```

- **Parse attributes into fields in a separate Map** - Attributes encountered while parsing will be included as fields in a separate Map. The name of this map will be "EA_XML_ATTR". The service developer may choose to write custom code to access this attribute data. As the XML parser traverses through the input XML data and generates the Map/List for the service method, it will use the "attribute parsing options" property to determine how to handle XML attributes, and then generate the resulting Map/List structure accordingly.

Examples

The following examples show what the resulting Map will look like for each of the three attribute parsing options, when the same XML file is parsed. Here is the XML file to be parsed:

```
<A>
   <Person Gender="male">
       <Name Last="Smith"> Joe </Name>
       <Addr> 123 4th Street </Addr>
   </Person>
</A>
```

This is what the resulting Map looks like when attributes are ignored. Notice that the attributes "`Gender`" and "`Last`" are not included in the Map.

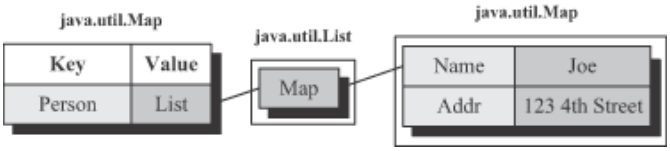


Figure 6. The map when attributes are ignored

The following diagram shows how the resulting Map looks when attributes are parsed into fields to be included in the Map. Here, the attributes are treated as sub-elements of the tag to which they belong. The "`Gender`" attribute becomes a sub-element of the `Person` tag, and is a field in the Map along with the `Name` and `Addr` fields (also sub-elements of the `Person` tag).

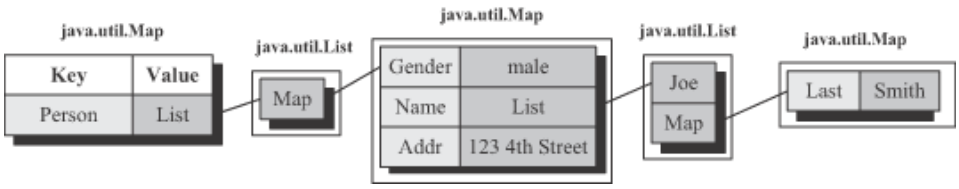


Figure 7. When attributes are parsed into fields to be included in the map

The following diagram shows the map resulting when attributes are parsed into fields in a separate Map. Here, the attributes are included in a separate Map called `EA_XML_ATTR`.

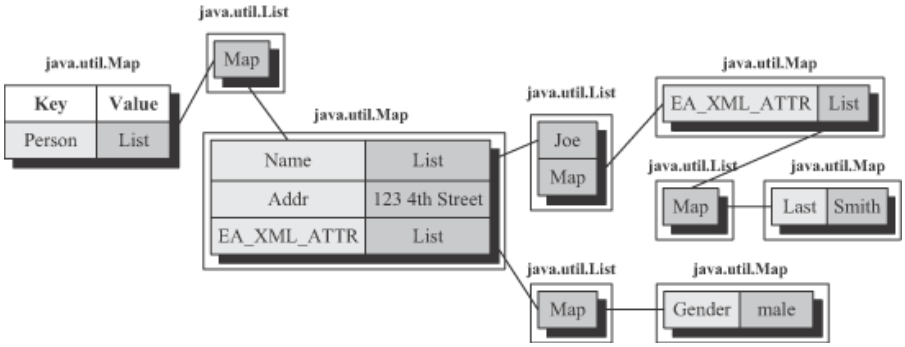


Figure 8. The map resulting when attributes are parsed into fields in a separate map

XSLT Support

Support for XSLT is provided by allowing the following fields in the input message for a method invoke:

Field	Description
`EA_XSL_IN`	Used to transform input EA_XML input fields. Contains the XSL data in string format.
`EA_XSL_IN_URI`	Used to transform input EA_XML input fields. Contains a reference to the XSL data via a URI (URI: Uniform Resource Identifier).
`EA_XSL_OUT`	Used to transform output data (only if “Service output in XML format” is enabled). Contains XSL data in string format.

Field	Description
EA_XSL_OUT_URI	Used to transform output data (only if "Service output in XML format" is enabled). Contains a reference to the XSL data via a URI.

The "EA_XSL_IN" fields describe the transformation to take place against the XML data provided in the "EA_XML" field(s) in the same input message. If more than one "EA_XSL_IN" field is present, only the first one is used; the rest are ignored.

When an "EA_XSL_IN" field is included in the input message, and the low-level service code encounters an "EA_XML" field while iterating through the message fields, it will first transform the XML data by running it through an XSLT processor. The XML data (from the "EA_XML" field) and the XSL data (from the "EA_XSL_IN" field) are passed into this processor, which then outputs the transformed data in XML format. This data is then parsed into a map (by using the existing XML parser), and then added to the input map.

If an "EA_XSL_IN" field is included in the input message, but there is no corresponding "EA_XML" field, the "EA_XSL" field is ignored, since it has no relevance without the associated XML data.

XSLT Method Input Data as a String

This JClient3 example shows the building of method input data that contains a XML string and a XSLT string.

```
// Create a Map object
Map map = new OrderedMap();
String xml = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>" +
    "<developers>" +
    "     <developer>" +
    "          <type>early adopter</type>" +
    "          <attitude>computers can do anything</attitude>" +
    "     </developer>" +
    "     <developer>" +
    "          <type>reluctant adopter</type>" +
    "          <attitude>well if they did it, I guess I could do it too</
attitude>" +
    "     </developer>" +
    "     <developer>" +
    "          <type>future manager</type>" +
```

```
    "          <attitude>I'm leaving at 5 and I don't care what needs to be
done</attitude>" +
    "     </developer>" +
    "</developers>";

ArrayList al = new ArrayList();
al.add(xml);
map.put("EA_XML", al);

al = new ArrayList();
String xsl = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>" +
    "<xsl:stylesheet version=\"1.0\" xmlns:xsl=\"http://www.w3.org/1999/XSL/
Transform\">" +
    "     <xsl:strip-space elements=\"*\"/>" +
    "     <xsl:template match=\"@*|node()\">" +
    "          <xsl:copy>" +
    "               <xsl:apply-templates select=\"@*\"/>" +
    "               <xsl:apply-templates/>" +
    "          </xsl:copy>" +
    "     </xsl:template>" +
    "     <xsl:template match=\"type\">" +
    "          <developerType>" +
    "               <xsl:value-of select=\".\"/>" +
    "          </developerType>" +
    "     </xsl:template>" +
    "     <xsl:template match=\"attitude\">" +
    "          <developerAttitude>" +
    "               <xsl:value-of select=\".\"/>" +
    "          </developerAttitude>" +
    "     </xsl:template>" +
    "</xsl:stylesheet>");
al.add(xsl);
map.put("EA_XSL_IN", al);
```

XSLT Method Input Data as a URI

This JClient3 example shows the building of method input data that contains a XML string and XSLT as a URI (the URI points to a file containing the XSLT data).

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

String xml = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>" +
```

```
    "<developers>" +
    "     <developer>" +
    "          <type>early adopter</type>" +
    "          <attitude>computers can do anything</attitude>" +
    "     </developer>" +
    "     <developer>" +
    "          <type>reluctant adopter</type>" +
    "          <attitude>well if they did it, I guess I could do it too</
attitude>" +
    "     </developer>" +
    "     <developer>" +
    "          <type>future manager</type>" +
    "          <attitude>I'm leaving at 5 and I don't care what needs to be
done</attitude>" +
    "     </developer>" +
    "</developers>";

ArrayList al = new ArrayList();
al.add(xml);
map.put("EA_XML", al);

al = new ArrayList();
String xsl_uri = "file:///xslt/xslt.txt";

al.add(xsl_uri);
map.put("EA_XSL_IN_URI", al);
```

Returning XML Output to the Client

The configurable JService property, `XMLOnOutput` (Service output in XML format), specifies whether the output data from a service should be in XML-format. When this property is set, the reply message contains a String field named "`EA_XML`" which contains XML-formatted data.

The "`EA_XSL_OUT`" fields describe the transformation to take place against the output data if "Service output in XML format" is enabled for the service.

If an "EA_XSL_OUT" field was included in the input message, but there is no output data, or the "Service output in XML format" configuration option is disabled, the "EA_XSL_OUT" field is ignored, since it has no relevance without associated XML data.

If more than one "`EA_XSL_OUT`" field is present, only the first one is used; the rest are ignored.

The output XML data is created on the server side by traversing the service's reply Map/List structure, and creating XML elements for each element in the Map/List. This XML does not conform to any specific DTD. It is simply well-formed XML defining the name/value pairs contained within the reply Map/List.

You may choose to return the XML data to the client in conformance with a specific DTD, using custom code in the service. The EA_XML String field can be created in custom code to contain the output XML data. When the service sends the reply message back to the client, the low-level XML-parsing service code will recognize the `EA_XML` field in the output map and assume it already contains XML-formatted data. In this case, it will simply send it, along with any other fields in the output map, straight through to the client as is.

> **Note:** Anytime the `EA_XML` field exists in the output map from the service, the low-level parsing code will send the entire output map to the client unchanged, regardless of how the `XMLOnOutput` property is set.

If there is no `EA_XML` field in the output map from the service, the `XMLOnOutput` property is used to determine if the output should be in XML format. When this property is enabled, then the entire output map is converted into XML and sent back in one field, called `EA_XML`. If it is disabled, then the entire map is sent back to the client unchanged.

If the customer prefers to perform their own XSLT processing, they may do so by writing custom service code. The XSL and XML data would then be included in the input message, in fields NOT named "EA_XSL" and "EA_XML". Like the XML parsing classes, the XSLT processing classes would be provided in the E/A installation "classes" directory, for the customer to make use of.

Logging

Messages regarding the transformation of XML Input/Output will appear in the service log (if the service log level is >= 6) when an XML fields are being transformed.

The message regarding XML output transformation will only appear in the service log (if the service log level is >= 6) if the method output is XML ("Service output in XML format" is enabled) and is being transformed.

JAXP Warning Message

With version 1.1 of the JAXP, a warning message will appear in the JCluster log when the log level is set high enough.

This message will appear the first time an XSLT transform or an XML parse is performed and it has no effect on the transforming or parsing. Oracle is not going to fix the problem.

Chapter 4. Web Services

This document describes webMethods JI's compliance with Web Services technology. There is currently an effort underway to expose webMethods JI services as Web Services.

This provides the ability to:

- Generate a WSDL document describing the JI service.
- Register the JI service in a UDDI registry.
- Handle SOAP requests to invoke methods in a JI service.

Assumptions

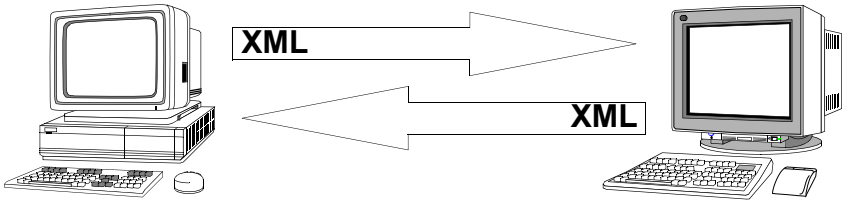
- This document assumes that you are familiar with webMethods JI services. Services are discussed in this document as they relate to or are used by the SOAP Gateway servlet. For more information on writing webMethods JI services, see the *webMethods JI User's Guide.*
- The implementation of the SOAP Gateway servlet requires the installation and configuration of a third-party servlet engine application.
- The testing of a generated web service requires the installation of a third-party XML testing tool, which is capable of:
  - Generating a SOAP request from a WSDL document.
  - Editing the generated SOAP request.
  - Sending the SOAP request to your server.
  - Receiving and displaying the SOAP response from your server.

  This functionality was tested using XML Spy 4.4. Instructions for this are provided in this document.

About Web Services

A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language.

Figure 9 shows the relationship between a web service provider and a web service requester:



Computer	**A**	**Computer**	**B**
Language:	Perl	Language:	Java
O/S:	Windows	O/S:	Linux

Figure 9. A Basic Web Service

There are several XML messaging options, such as SOAP, XML Remote Procedure Calls (XML-RPC), or simply passing arbitrary XML documents using standard HTTP GET/POST. Any of these options can work. This document, however, focuses on SOAP.

A web service may also have two additional properties:

- A web service should be self-describing. If you publish a new web service, you should also publish a public interface to this service. At the least, your service should include readable documentation, so that other developers can integrate your service without much difficulty.
- A web service should be discoverable. There should be a simple mechanism by which to publish the fact that you have created a new web service. Furthermore, there should be a simple mechanism whereby interested parties can find the service and locate its public interface.

A web service is, therefore, any service that:

- Is available over the internet.
- Uses a standardized XML messaging system.
- Is not tied to any one operating system or programming language.
- Is self-describing using a common XML grammar.
- Is discoverable through the use of a simple find mechanism.

Web Service Architecture

Web service architecture is described by means of a four-layer protocol stack. Following is a brief description of each layer:

Layer	Description
Service transport	This layer is responsible for transporting messages between applications. This layer includes HTTP, SMTP, FTP, and newer protocols, such as BEEP.
XML messaging	This layer is responsible for encoding messages in a common XML format so that messages can be understood at each end. This layer includes XML-RPC and SOAP.
Service description	This layer is responsible for describing a web service's public interface. Service description is handled using WSDL.
Service discovery	This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Service discovery is handled via UDDI.

The following figure illustrates the web service protocol stack:

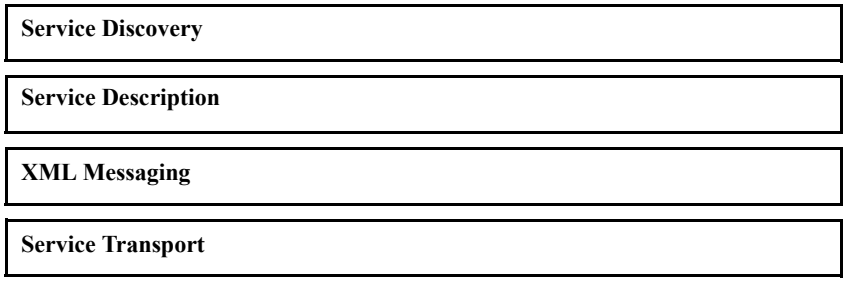


Figure 10. Web service protocol stack

Service Transport: HTTP

The bottom of the web service protocol stack is service transport. This layer is responsible for actually transporting XML messages between two computers. HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. As a result, SOAP messages may sneak through appearing as HTTP messages.

This makes it possible to integrate remote applications.

XML Messaging: SOAP

Simple Object Access Protocol (SOAP) is an XML-based protocol for exchanging information between computers. The main focus of SOAP is Remote Procedure Calls (RPCs) transported via HTTP. SOAP is platform-independent and therefore enables diverse applications to communicate with each other. The SOAP specification defines three major parts:

- SOAP envelope specification (section 4 of the specification). The SOAP XML Envelope defines specific rules for encapsulating the data that is transferred between computers. This includes application-specific data, such as the method name to invoke, method parameters, or return values.
- Data encoding rules (section 5 of the specification). SOAP includes its own set of conventions for encoding data types. This enables different computers to agree on rules for encoding specific data types. These conventions are mostly based on the W3C XML Schema specification.
- RPC conventions (section 7 of the specification). SOAP defines a simple convention for representing remote procedure calls and responses. This enables a client application to specify a remote method name, include any number of parameters, and receive a response from the server.

The SOAP Message

A one-way message, which is either a request from a client or a response from a server, is officially referred to as a SOAP message. Every SOAP message has a mandatory Envelope element, an optional Header element, and a mandatory Body element. Each of these elements has an associated set of rules. The full SOAP specification is available at http://www.w3.org/tr/soap.

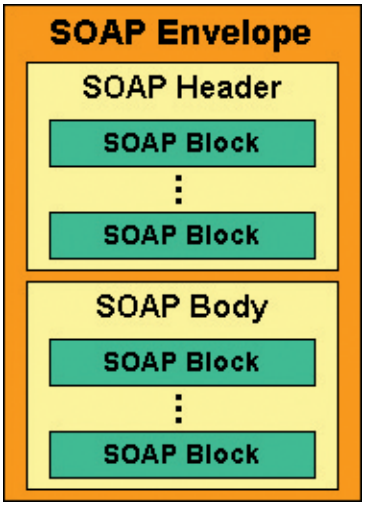


Figure 11. SOAP XML Message Main Elements

<SOAP-ENV:Envelope> Every SOAP message has a root Envelope element. Unlike other specifications, such as HTTP and XML, SOAP does not define a traditional versioning model based on major and minor release numbers. Rather, SOAP uses XML name spaces to differentiate versions. The version is referenced within the Envelope element.

For example:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

The SOAP 1.1 namespace URI is http://schemas.xmlsoap.org/soap/envelope/, whereas the SOAP 1.2 namespace URI is http://www.w3.org/2001/12/soap-envelope.

> **Note:** The exact value of the SOAP1.2 envelope namespace will likely change to reflect the final date of the SOAP 1.2 release. The value, http://www.w3.org/2001/12/soap-envelope, reflects the specification from December 2001.

<SOAP-ENV:Header> The optional Header element is used for specifying additional application-level requirements. For example, The Header element can be used to specify a digital signature for password-protected services. Currently, many SOAP services do not utilize the Header element.
However, it is believed that as SOAP services mature, the Header element shall provide an open mechanism for authentication, transaction management and payment authorization.

<SOAP-ENV:Body> The Body element is mandatory for all SOAP messages. Typical uses of the Body element include RPC requests and responses.

<SOAP-ENV:Fault> In the event of an error, the Body element of a SOAP response includes a Fault element. The Fault sub-elements include the faultCode, faultString, faultActor, and detail elements. These sub-elements categorize and describe the error.

A SOAP Request

Following is an example of a SOAP request. This SOAP request invokes the MainMethod method and supplies this method with a parameter. This method is a bank account query and the method parameter is the bank account number:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:MainMethod xmlns:m="MainMethod">
      <AccountNum xsi:type="xsd:string">123456778</AccountNum>
    </m:MainMethod>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A SOAP Response

Following is an example of a SOAP response. This SOAP response consists of two bank transactions performed on 05/25/97. The first bank transaction is a $225.68 deposit and the second bank transaction is a $20.52 withdrawal:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <SOAP-ENV:Body>

    <MainMethodResponse>

      <TableTemplate_0>
        <Amount_field xsi:type="xsd:string"> 225.68 </Amount_field>
        <Date_field xsi:type="xsd:string">05/25/97 </Date_field>
        <Type_field xsi:type="xsd:string">PYMT   </Type_field>
      </TableTemplate_0>

      <TableTemplate_0>
```

```
        <Amount_field xsi:type="xsd:string">  20.52-</Amount_field>
        <Date_field xsi:type="xsd:string">05/25/97 </Date_field>
        <Type_field xsi:type="xsd:string">LCHG   </Type_field>
      </TableTemplate_0>

    </MainMethodResponse>

  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Resources

Following are some useful SOAP resources:

- The official W3C SOAP specification: http://www.w3.org/tr/soap/
- The default namespace for the SOAP envelope: http://www.w3.org/2001/12/soap-envelope
- The default namespace for SOAP encoding and data types: http://www.w3.org/2001/12/soap-encoding
- Online SOAP tutorial: http://www.w3schools.com/soap/

Service Description: WSDL

The Web Services Description Language (WSDL) is a specification for describing web services in a common XML grammar. WSDL replaces SOAP as the means of expressing web services in the UDDI registry. SOAP requests and responses are created according to the specifications laid forth in WSDL. The information described in WSDL is divided into four main categories:

- Interface information. This includes descriptions of all publicly available functions.
- Data type information. This includes definitions of message requests and message responses.
- Binding information. This includes information about the transport protocol to be used.
- Address information. This includes the actual location of the service provider, specifically the URL for invoking the specified web services.

The WSDL Elements

A WSDL document defines a web service using six major elements. The following table lists and briefly describes these:

WSDL Element	Description
`definitions`	The root element, which defines the name of the web service, declares multiple name spaces, and contains the rest of the elements described here.
`types`	This element defines the data types used in messages.
`message`	This element defines the message elements of a communication.
`portType`	This element defines the exposed (legal) operations.
`binding`	This element defines the communication protocols.
`service`	This element aggregates a set of related ports.

Note: The following sections provide more detailed explanations of each element, including examples. These examples all relate to a single WSDL document. This WSDL document was produced by MapMaker from A JI service named “EATutorialSVC”. This service consists of one method named “MainMethod”, which requires an “AccountNum” parameter and returns a list of bank account transactions. Each banking transaction returned by the method consists of three parameters.

Note:

<wsdl:definitions> The <definitions> element is the root element of a WSDL document. The <definitions> tag contains name space attributes, which define the XML name spaces used throughout the document.

In the following example, you can see a <definitions> element, its name space attributes and the main elements found in every WSDL document:

```
<wsdl:definitions
  targetNamespace="http://localhost:8080/JISOAP/EATutorialSvc"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:impl="http://localhost:8080/JISOAP/EATutorialSvc-impl"
  xmlns:intf="http://localhost:8080/JISOAP/EATutorialSvc"
```

```
  xmlns:tns1="http://wsdl.tutorial.ea"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>.........definition of data types........</wsdl:types>

  <wsdl:message>.......definition of a message.......</wsdl:message>

  <wsdl:portType>......definition of a port.........</wsdl:portType>

  <wsdl:binding>.......definition of a binding.......</wsdl:binding>

  <wsdl:service>.......definition of a service.......</wsdl:service>

  <-- extensibility element -->

</wsdl:definitions>
<wsdl:types>
```

The <types> element defines the data type definitions that are used for messages. For maximum platform neutrality, WSDL uses data types as defined in the XML Schema specification.

In the following example, a <type> element defines the data types used in the service's method:

```
<wsdl:types>

  <schema targetNamespace="http://wsdl.tutorial.ea"
          xmlns="http://www.w3.org/2001/XMLSchema">

    <complexType name="JI_MainMethod">
      <sequence>
        <element name="TableTemplate_0" nillable="true"
                 type="tns1:MainMethod_TableTemplate_0_TableTemplate"/>
      </sequence>
    </complexType>

    <complexType name="MainMethod_TableTemplate_0_TableTemplate">
      <sequence>
        <element name="Type_field" nillable="true" type="xsd:string"/>
        <element name="Date_field" nillable="true" type="xsd:string"/>
        <element name="Amount_field" nillable="true" type="xsd:string"/>
      </sequence>
```

```
    </complexType>
    <element name="JI_MainMethod" nillable="true"
             type="tns1:JI_MainMethod"/>

  </schema>

</wsdl:types>
```

<wsdl:message> A <message> element defines the data elements of a communication. Each message consists of one or more logical parts. The parts can be compared to the parameters of a function call in a traditional programming language.

In the following example, two <message> elements define the MainMethodRequest and MainMethodResponse messages:

```
<wsdl:message name="MainMethodRequest">
  <wsdl:part name="AccountNum" type="SOAP-ENC:string"/>
</wsdl:message>

<wsdl:message name="MainMethodResponse">
  <wsdl:part name="return" type="tns1:JI_MainMethod"/>
</wsdl:message>
<wsdl:portType>
```

The <portType> element defines the operations that can be performed by a web service, and the messages that are involved. A port type can be compared to a function library (or a module, or a class) in a traditional programming language. Each port type can consist of one or more operations. An operation represents one method and can be compared to a function in a traditional programming language. A WSDL may define four port types:

- One-way. The port can receive a message.
- Request-response. The port can receive a request and return a response.
- Solicit-response. The port can send a request wait for a response.
- Notification. The port can send a message.

In the following example, a <portType> element, defines the MainMethod operation. This definition includes the method name, the method parameter to expect from the client, and a reference to the SOAP request and response messages previously defined in the two <message> elements:

```
<wsdl:portType name="EATutorialSvc">
  <wsdl:operation name="MainMethod" parameterOrder="AccountNum">
    <wsdl:input message="intf:MainMethodRequest"/>
    <wsdl:output message="intf:MainMethodResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

```
<wsdl:binding>
```

Each binding element describes a supported protocol. Similar to the <portType> element, the binding element includes supported operations, as well as the input and output for each operation. The bindings provide concrete information on what protocol is being used, how the data is being transported, and where the service is located.

SOAP requests and responses are created according to the specifications laid forth in this element.

In the following example, a <binding> element defines the binding of the EATutorialSvc service. The <wsdlsoap:binding> sub-element represents a SOAP binding transported in the form of SOAP messages via HTTP:

```
<wsdl:binding
  name="EATutorialSvcPortSoapBinding"
  type="intf:EATutorialSvc">


  <wsdlsoap:binding
    style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>


  <wsdl:operation name="MainMethod">


    <wsdlsoap:operation soapAction=""/>


    <wsdl:input>
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="MainMethod"
        use="encoded"/>
    </wsdl:input>


    <wsdl:output>
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost:8080/JISOAP/EATutorialSvc"
        use="encoded"/>
    </wsdl:output>


  </wsdl:operation>
</wsdl:binding>
```

<wsdl:service> The <service> element defines the ports supported by the web service. Each port is defined in a <port> element and represents one supported protocol. The service element is a collection of ports. Web service clients can learn from the service element where to access the service, through which port to access the Web service, and how the communication messages are defined.

In the following example, the <service> element consists of one port which specifies the SOAP binding and refers to the JISOAP Gateway servlet:

```
<wsdl:service name="EATutorialSvc">
  <wsdl:port
    binding="intf:EATutorialSvcPortSoapBinding"
    name="EATutorialSvcPort">
    <wsdlsoap:address
      location="http://localhost:8080/JISOAP/EATutorialSvc"/>
  </wsdl:port>
</wsdl:service>
```

WSDL Namespace Tags

The following table provides a summary of the WSDL namespace elements, their attributes, and their child-nodes:

Element	Attributes	Children
`<definitions>`	name, targetNamespace, xmlns	`<types>`, `<message>`, `<portType>`, `<binding>`, `<service>`
`<types>`	(none)	`<xsd:schema>`
`<message>`	name	`<part>`
`<portType>`	name	`<operation>`
`<binding>`	name, type	`<operation>`
`<service>`	name	`<port>`
`<part>`	name, type	`(empty)`
`<operation>`	name, parameterOrder	`<input>`, `<output>`, `<fault>`

Element	Attributes	Children
`<input>`	name, message	(empty)
`<output>`	name, message	(empty)
`<fault>`	name, message	(empty)
`<port>`	name, binding	`<wsdlsoap:address>`

WSDL Resources

Following are useful WSDL resources:

- The official W3C WSDL specification: http://www.w3.org/tr/wsdl
- Online WSDL tutorial: http://www.w3schools.com/wsdl/

Service Discovery: UDDI

Universal, Description, Discovery, and Integration (UDDI) represents a technical specification for registering and finding web services. UDDI enables companies to find one another on the Web and make their systems interoperable for e-commerce, virtually streamlining online transactions. UDDI is often compared to a telephone book in that it allows businesses to list themselves by name, product, location, or the web services they offer. Accordingly, the data captured within UDDI is divided into three main categories:

- White pages. This category includes general information about the company providing a web service. This information may include the business name, business description, and address.
- Yellow pages. This category includes general classification data for either the company or the service offered. This data may include industry, product, or geographic codes based on standard taxonomies.
- Green pages. This category includes technical information about a web service. This includes a pointer to an external specification and an address for invoking the web service.

At its core, UDDI consists of two parts. First, UDDI is a technical specification for building a distributed directory of businesses and web services. Data is stored within a specific XML format, and the UDDI specification includes API details for searching existing data and publishing new data. Second, the UDDI Business Registry, also known as the UDDI “cloud services”, is a fully operational

implementation of the UDDI specification. Launched in May 2001 by Microsoft and IBM, the UDDI registry now enables anyone to search existing UDDI data. It also enables any company to register itself and its services.

UDDI Technical Architecture

The UDDI technical architecture consists of three parts:

- UDDI data model. An XML schema for describing businesses and web services.
- UDDI API. A SOAP-based API for searching and publishing UDDI data.
- UDDI cloud services. Operator sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.

The current UDDI cloud services are provided by Microsoft and IBM. Theses cloud services provide a logically centralized, but physically distributed, directory. This means that data submitted to one root node is automatically replicated and distributed across all other root nodes. currently, data replication occurs every 24 hours.

It is possible, however, for a large company to set up a private UDDI registry. for registering all internal web services. As such a registry is not automatically synchronized with the root UDDI nodes, it is not considered part of the UDDI cloud.

UDDI Data Model

UDDI includes an XML schema that describes four main types of information:

- businessEntity
- businessService
- bindingTemplate
- tModel

The following figure illustrates the containment hierarchy:

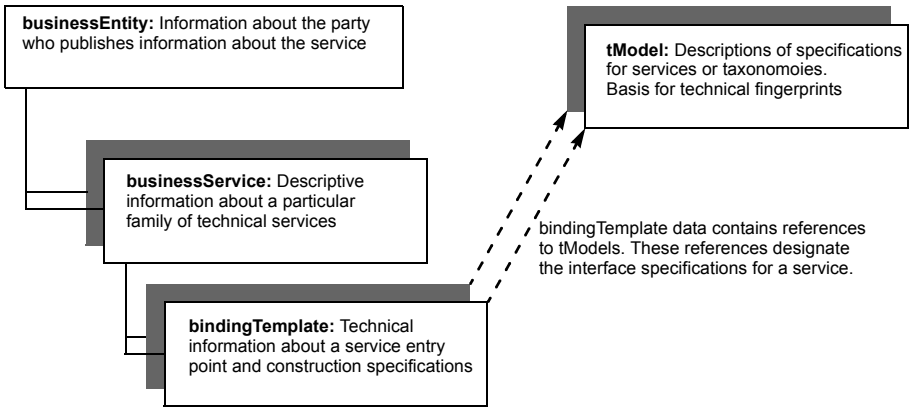


Figure 12. The UDDI data model

<businessEntity> The businessEntity element includes information about the actual business. This includes business name, description, address, and contact information. The following example illustrates a businessEntity element's basic structure:

```
<businessEntity
  businessKey="0076b468-eb27-42e5-ac09-9955cff462a3"
  operator="Microsoft Corporation"
  outhorizedName="Sefi Musael">


  <name>................business name.....................</name>
  <description>.........business description........</description>
  <contacts>............contact information............</contacts>
  <identifierBag>.......business identifiers.......</identifierBag>
  <categoryBag>.........business categories.........</categoryBag>


</businessEntity>
```

Upon registering, each business receives a unique businessKey value. For example, the businessKey for Microsoft is 0076b468-eb27-42e5-ac09-9955cff462a3. This key is used to tie a business to its published services.

In addition to basic contact information, the businessEntity record can include optional business identifiers and business categories For example, UDDI is currently set up to accept both Dun & Bradstreet D-U-N-S numbers and Thomas Registry supplier IDs.

Businesses can also register multiple business categories. This can include industry, product, service or geographic codes based on standard taxonomies. UDDI is currently prepopulated with the NAICS, UNSPSC, and ISO 3166 business categories.

<businessService> The businessService element includes information about a single web service or a group of related web services. This includes name, description, and possibly an optional list of bindingTemplates. The following example illustrated the businessService element's basic structure:

```
<businessService
  serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
  businessKey="ba744ed0-3aaf-11d5-80dc-002035229c64">


  <name>................service name.....................</name>


  <description>..........service description........</description>


  <bindingTemplates>
    </bindingTemplate>
    </bindingTemplate>
  <bindingTemplates>
```

```
</businessService>
```

Note: Like the businessEntity, each businessService has a unique serviceKey.

<bindingTemplate> The bindingTemplate element includes information about how and where to access a specific web service. The following example illustrates a bindingTemplate which is referenced by the previous businessService element:

```
<bindingTemplate
  serviceKey="d5921160-3e16-11d5-98bf-002035229c64"
  bindingKey="9594a970-3e16-11d5-98bf-002035229c64">

  <description>binding description.....................</description>

  <accessPoint>url for connection to service provider...</accessPoint>

  <tModelInstanceDetails>
    <tmodelInstanceInfo
      tmodelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64" />
  </tModelInstanceDetails>

</bindingTemplate>
```

<tModel> tModel stands for technical model. tModels are primarily used to provide pointers to external technical specifications. The bindingTemplate provides information on where to access the SOAP binding but it does not provide information on how to interface with it. The tModel element fills this gap by providing a pointer to an external specification. This makes it possible for various web services to share a single tModel, effectively standardizing web service interface.

The following example illustrates a tModel element which is referenced by the previous bindingTemplate:

```
<tModel
  tModelKey="uuid:0e727db0-3e14-11d5-98bf-002035229c64"
  operator="www.ibm.com/services/uddi"
  authorizedName="0100001QS1">

  <name>................tModel name.........................</name>
  <description>.........tModel description............</description>

  <overviewDoc>
    <description>.......WSDL link....................</description>
    <overviewURL>.......URL for web service WSDL......</overviewURL>
```

```
  </overViewDoc>

  <categoryBag>
    <keyedReference
      tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
      keyName="uddi-org:types"
      keyValue"wsdlSpec" />
  </categoryBag>

</tModel>
```

UDDI Resources

Following are some useful UDDI resources:

- The official UDDI specification:
  http://uddi.org/specification.html
- Online UDDI tutorial:
  http://www.systinet.com/download/tutorialfive.pdf
- UDDI FAQ:
  http://www.uddi.org/faqs.html

Developing a webMethods JI Web Service

Following is the recommended development plan for creating a web service:

1. Creating core functionality — this refers to creating the webMethods JI service in MapMaker. For instructions, see Chapter 10: "Services" of the *webMethods JI User's Guide*.
2. "Generating a WSDL in MapMaker" on page 92.
3. "Setting Up the SOAP Gateway" on page 95.
4. Deploying your webMethods JI web service — for instructions, see Chapter 10: "Services" of the *webMethods JI User's Guide*.
5. "Testing your Generated Web Service" on page 103.
6. "Publishing a Web Service to UDDI" on page 105.

Generating a WSDL in MapMaker

WSDL Generation — Overview

MapMaker provides the ability to generate a WSDL file for your service. This file, named <ServiceName>.wsdl, is created and placed in the directory specified in the Definitions tab of the Generate Code dialog box (described in the following section, "Generating the WSDL Code" on page 92). One WSDL file is generated per-MapMaker service, so maps that contain multiple services result in multiple WSDL output files. Web service code is also generated, and can then be deployed into the SOAP Gateway.

Generating the WSDL Code

To generate a WSDL for the service you have created in MapMaker, proceed as follows:

1. Choose one of the following commands:

   Generate all code: Select Tools > Generate > Generate All Code... or click the Generate All Code button,

   Or

   Generate Definitions: Select Tools—Generate—Generate Definitions, or click the Generate Definitions button.
2. In the Generate Code dialog box, select the Definitions tab (Figure 13).

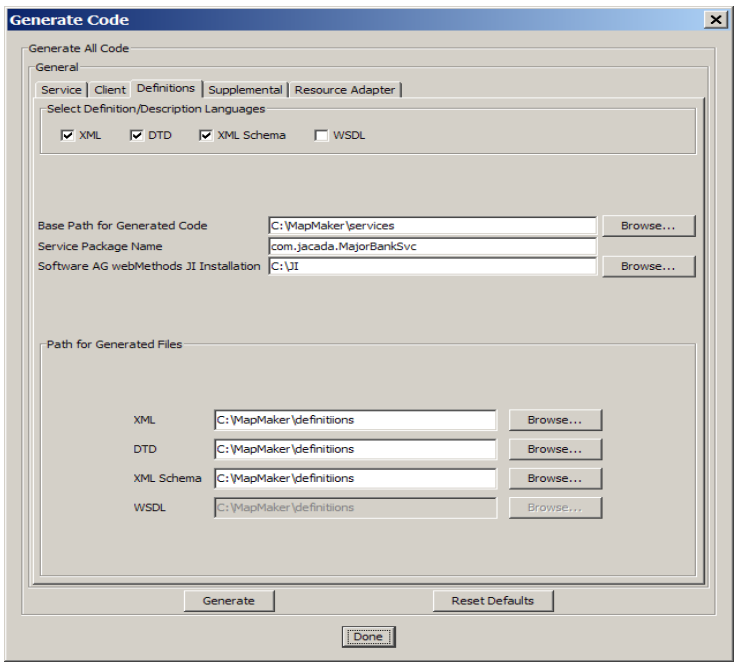


Figure 13. The Generate Code dialog box - Definitions tab

Note: If you have chosen to generate only the Definition files, this will open the Generate Code dialog box directly in the Definitions tab.

3 In the Definitions tab, proceed as follows:
   - Make sure the WSDL checkbox is checked.
   - Under Path for Generated Files, in the WSDL field, specify the path that will contain the generated WSDL.
4 Click on the **Generate** button.
5 Click **OK**.

Note: MapMaker allows you to create a WSDL definition file only if the method's parameters match the Web Service Support setting in the service's Properties View.

Additional WSDL Settings in MapMaker

MapMaker provides the ability to set the default service base path for SOAP address binding. This property, defined in the service's Properties View (Figure 14), specifies the location (URL) of the SOAP Gateway.

Figure 14. The Service Properties view

For more information on this setting, see "Configuring Service Properties" in Chapter 10: "Services" of the *webMethods JI User's Guide*.

Setting the Default Path for Generated WSDLs

To set the default path for generated WSDLs, proceed as follows:

1 Select **File** > **Properties**. The Properties dialog box opens.
2 In the Properties dialog box, select the Default Directories tab (Figure 15).

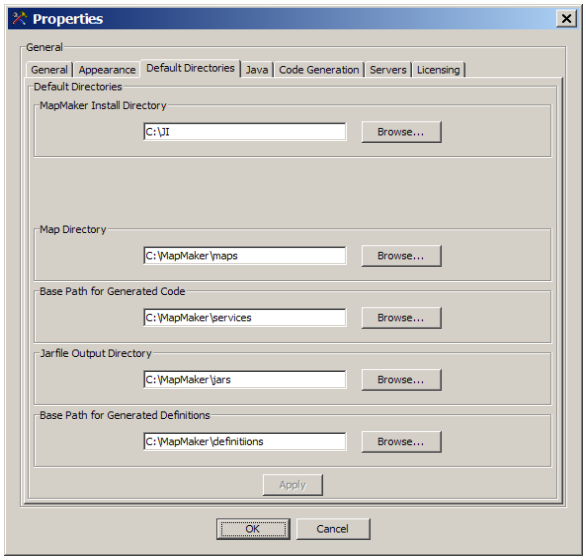


Figure 15. The Default Directories tab

3 Under Base Path for Generated Definitions, enter the desired default path.

4 Click **Apply**.

Note: While generating the WSDL, you may choose not to accept these settings.

Setting the Default Base Path for SOAP Address Binding

To set the default base path for SOAP address binding, proceed as follows:

1 In the Services tab, select the service in the Tree View to display its properties in the Properties View (Figure 14).

2 Under Service Base Path for SOAP Address Binding, enter the desired path.

For example: http://localhost:8080/JIWSVC/services/.

3 Click the **Apply** button .

Note: While generating the WSDL, you may choose not to accept these settings.

Setting Up the SOAP Gateway

webMethods JI SOAP Gateway — Overview

The SOAP Gateway is a web application servlet, provided as part of the webMethods JI installation package. It is based on the Apache Axis 1.1 SOAP implementation. The SOAP Gateway is named JIWSVC.war, and it is located in the *<JI_Install_Dir>\lib* directory.

The SOAP Gateway handles SOAP-over-HTTP requests. The communication architecture is similar to that of the HTTP Gateway, in that the SOAP Gateway provides a configuration file mechanism to support connection properties, and uses the `JClient3` library to access JI services.

SOAP Gateway Requirements

The SOAP Gateway requires the installation and configuration of a web container, such as Apache Tomcat (included in the webMethods JI installation).

The SOAP Gateway Messaging Styles

The SOAP Gateway supports two messaging styles: Document and RPC. This provides flexibility in the communication between the client and the server, as the information can be passed in the most suitable format (either an XSD/XML or an MLM data type), and processed by the webMethods JI component that is most equipped to handle it.

Document-Style Messaging and the XSD/XML External Data Type

Document style allows an XML document to be embedded in the SOAP message body, which contains the data to be processed. The XML document can be of any format that has been agreed upon between the server and the client. The webMethods JI server controls how the document is handled, processing it according to its contents.

Document style messaging fits the XSD/XML external data type, which expects a complete XML document (declaration and document contents). The SOAP Gateway can easily handle the XML message, by passing it on to the webMethods JI server for XML data interpretation. The WSDL file generated by MapMaker contains the XML schema defining the request and response messages for each method in the webMethods JI service.

RPC Style Messaging and the MLM External Data Type

The RPC messaging style allows a client to select a specific service and method on a remote server; invoke the method with defined parameters and formats; and receive a response from that method. The server is used by the client as a platform for processing data in the order defined by the client. The message definition is a standard method/function call definition, including a name, inputs and outputs.

RPC style is used for webMethods JI methods with MLM external data types.

Selecting the Messaging Style in MapMaker

MapMaker can generate WSDL and expose webMethods JI web services as either RPC-style or Document-style services. The style in which the service is deployed to the SOAP Gateway is determined by the **Web Service Support** setting in the Service's **Properties** View (Figure 14). The generated WSDL file reflects the service's style.

SOAP Gateway Packaging

The SOAP Gateway is provided as a `WAR` file named `JIWSVC.war`, which resides in the *<JI_Install_Dir>\lib* directory. This file consists of all the class files and JARs that make up the SOAP Gateway. JIWSVC.war is installed in the same way as the HTTP Gateway (see "Installing the HTTP Gateway Servlet" on page 11). The following is the directory structure and contents of the WAR file:

```
WEB-INF/
WEB-INF/classes
WEB-INF/classes/*.class
WEB-INF/config
WEB-INF/config/JISOAP.cfg
WEB-INF/lib
WEB-INF/lib/*.jar
WEB-INF/logs
WEB-INF/web.xml
WEB-INF/server-config.wsdd
```

The following table describes the files in the SOAP Gateway directory structure.

File/Directory	Description
WEB-INF/classes/	Contains the Java class files.

File/Directory	Description
WEB-INF/config	Contains the configuration properties files. The default configuration file, `JIWSVC.cfg`, is provided.
WEB-INF/lib/	Contains `JAR` files required by the SOAP Gateway. These include: • `JClient3.jar` • `SSL jars (jcert.jar, jnet.jar, jsse.jar)` • `The Axis jars`
WEB-INF/logs/	An empty directory, which usually contains the SOAP Gateway log files.
WEB-INF/web.xml	The standard Axis `web.xml` file, with additional initialization parameters for the SOAP Gateway.
WEB-INF/server-config.wsdd	The standard Axis `server.config` file, with additional information necessary for the SOAP Gateway. **Note:** This file should not be modified.

Initialization Parameters

The `web.xml` file provided in the `JIWSVC.war` file contains the following parameters used by the SOAP Gateway:

Parameter	Description
propertiesFile	The name of the SOAP Gateway's properties file. The file is assumed to reside in the *WEB-INF/config* directory.

Parameter	Description
logLevel	The initial value of the log level to be used by the SOAP Gateway. This parameter may also be set in the properties file. By default, `logLevel` is set to 0. If its value is larger than 0, a file named `soapgateway.log` is created in the *WEB-INF/logs* directory.

SOAP Gateway Request URL

The request URL required to access the SOAP Gateway has the following format:

```
http://host:port/<webappname>/<servicename>[?data]
```

The following table describes the URL's components:

URL Component	Description
host:port	The location of the web server.
webappname	The name of the SOAP Gateway web application.
servicename	The name of the webMethods JI service.
data	One of the query strings supported by the SOAP Gateway.

The following is an example SOAP Gateway request URL:
http://myhost:8080/JIWSVC/services/banking

Determining the Properties File to be Used

Like the HTTP Gateway, the SOAP Gateway reads configurable properties from a file. The name of the properties file to be used is determined by one of the following (listed by precedence, from highest to lowest):

1. The `configFile` SOAP Header block.
2. The service name specified in the request URL (with `.cfg` appended).
3. The default properties file, specified in the `propertiesFile` initialization parameter of the `web.xml` file. If this parameter is not specified in the `web.xml` file, the default `JIWSVC.cfg` is used.

The properties included are the same as those listed for the HTTP Gateway in "The Properties File" on page 15.

If `configFile` is specified in the SOAP Header, but the file does not exist, a SOAP Fault containing a `JIWSVCSoapGatewayException` is returned. The following is an example of the response SOAP Body in this case:

```
<soapenv:Body>
  <soapenv:Fault>
   <faultcode>soapenv:Server.userException</faultcode>
<faultstring>com.jacada.ea.gateways.soap.JIWSVCSoapGatewayException:
Configuration file specified in SOAP Header not found (C:\jakarta-tomcat-
4.1.18\webapps\JIWSVC\WEB-INF\config\Test.cfg).</faultstring>
   <detail/>
  </soapenv:Fault>
 </soapenv:Body>
```

If `configFile` is not specified in the SOAP Header and the `serviceName.cfg` configuration file does not exist, the SOAP Gateway attempts to look for the default configuration file.

If neither the `serviceName.cfg` file nor the default configuration file exist, a SOAP Fault containing a `JIWSVCSoapGatewayException` is returned. The following is an example of the response SOAP body in this case:

```
<soapenv:Body>
  <soapenv:Fault>
   <faultcode>soapenv:Server.userException</faultcode>
<faultstring>com.jacada.ea.gateways.soap.JIWSVCSoapGatewayException: Could
not find default configuration file (C:\jakarta-tomcat-
4.1.18\webapps\JIWSVC\WEB-INF\config\JIWSVC.cfg). No config file to use!</
faultstring>
   <detail/>
  </soapenv:Fault>
 </soapenv:Body>
```

No matter how the SOAP Gateway determines the name of the configuration file to use, it always looks for the file in the *WEB-INF/config* directory.

Determining the webMethods JI Service Name

The SOAP Gateway determines the service name based on the request URL. For example, if the request URL is http://myhost:8080/JIWSVC/services/myservice, then the SOAP Gateway accesses the service named `myservice`. However, if the `serviceName` property is set in the configuration file, it overrides the service name specified in the request URL.

Query Parameter Support

The SOAP Gateway provides the following support for query strings in the request URL:

- `?WSDL` — this query string causes the SOAP Gateway to return the WSDL for a deployed service.

> **Note:** This capability is supported for HTTP GET requests only.

- ?___debug (where ___ is three underscores in succession) — this query string is no longer supported. However, each service that is deployed into the SOAP Gateway contains a `JIDebug()` method, which can be used to obtain version information from the SOAP Gateway. This information includes the version numbers of the SOAP Gateway and the `JClient3` library, along with the date and time on which the service class files were created.

SOAP Gateway Logging

Each service deployed as a web service into the SOAP Gateway provides a logging capability. To enable logging, the `logLevel` property must be set in the `web.xml` file. Valid `logLevel` values are 0, 1, 2 or 3. The following table describes each log level value.

Log Level Value	Description
0	A log file named `soapgateway.log` is created, showing the servlet initialization.
1	The client IP address and the request URI are shown for each received request. In addition, a separate log file named `<serviceName>_<threadID>.log` is created, showing basic service details.
2 or 3	A separate log file named `<serviceName>_<threadID>.log` is created. The higher the log level, the more details appear in the file. **Note:** To get the full debugging information on requests, set the logLevel to 3.

Log Level Value	Description
Missing or commented out	No log file is created for the service in question (the only exception to this is described in the following section, "Changing the Log Level Dynamically" on page 101).

Changing the Log Level Dynamically

The SOAP Gateway supports the ability to change the log level on a per-request basis. This is done by setting the `logLevel` property in the configuration file (which is, by default, `JIWSVC.cfg`) to the desired value before issuing a request.

The configuration file `logLevel` setting always takes precedence over the `web.xml` file `logLevel` setting. If the two are different, the logLevel property is changed for the current request only, and is reset to its original value when the current request is completed.

Version Information

The SOAP Gateway version and JClient3 version can be retrieved by invoking a special method within the service, called `JIDebug()`. The following is an example of the `JIDebug()` method output:

```
JI SOAP Gateway
  Gateway version: 1001
  JClient3 version: webMethods JI Version 4.0.1.4-1001
  Service created: Jun 26, 2003 1:24:06 PM
```

Configuring the Web Container

To configure your web container, you must install the JIWSVC servlet into the web container and copy your WSDL file into the JIWSVC servlet. Following are instructions for setting up the SOAP Gateway using the Apache Tomcat , which is supplied with the webMethods JI installation:

To set up the SOAP Gateway using Apache Tomcat:

1. Start Apache Tomcat.
2. From the *$JI_HOME\lib* directory, copy the *JIWSVC.war* file into the following directory:

   $Apache_Tomcat_HOME\webapps
3. Stop and start Apache Tomcat.

4 Using your Windows Explorer, browse to the *$Apache_Tomcat_HOME\webapps\JIWSVC* directory, and see if the *.WAR file exploded into its subdirectories, as seen in the following figure:

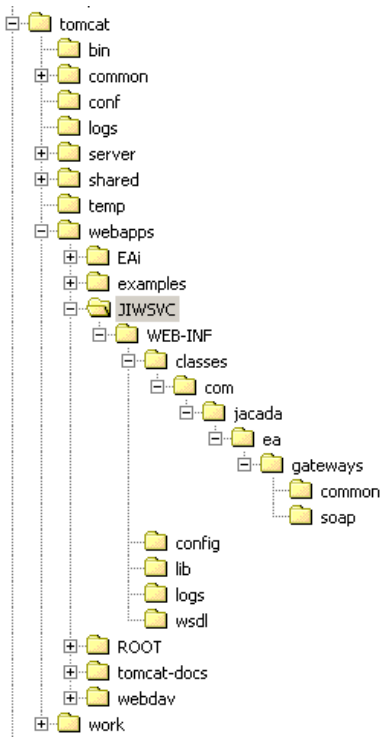


Figure 16. JIWSVC expanded sub-directories

5 Deploy the web service into the SOAP Gateway (by choosing **Tools > Deploy Map...** from the menu and deploying the service using the **Deploy to Resource Server** dialog box's **Web Services** tab).

6 Stop and start Apache Tomcat.

After completing this procedure, you are now ready to test your web service by sending a SOAP request to the server.

Note: Web Service Deployment requires a JVM (java.exe or javaw.exe) on the user's PATH. The specified JVM can be configured by editing all instances of the "jvm=" property in the <JI_install_dir>/templates/JIDeployWebService.xml file.

Testing your Generated Web Service

You can test your web service by simulating a SOAP client. To do this, you must create a SOAP request from your WSDL. Then you must send the SOAP request to the server and examine the SOAP response. There are several visual tools used for this, such as xmlspy and CapeClear. Following are instructions for testing your web service using xmlspy 5:

> **Note:** You can obtain xmlspy 5 at http://www.xmlspy.com/download.html

To test the web service using xmlspy 5:

1 Select **SOAP > Create new SOAP request**. A dialog box opens:

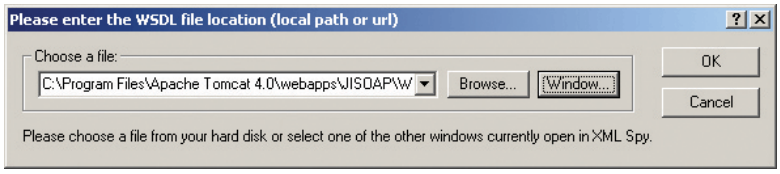


Figure 17. xmlspy: Creating a New SOAP Request

2 Enter the path to your web service's WSDL file and click **OK**.
Another dialog box opens:

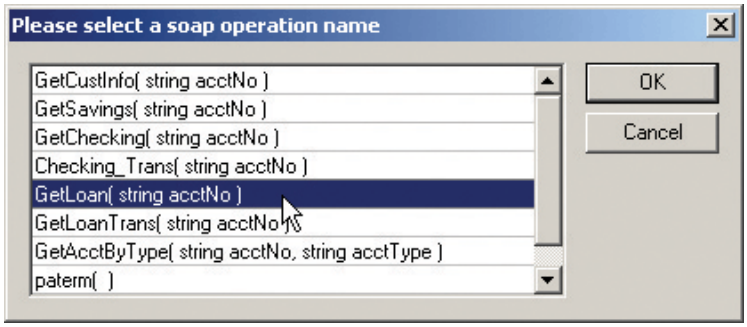


Figure 18. XML Spy: Select a method

3 Select a method for the SOAP request to invoke and click **OK**.

4 You should receive the SOAP request in Enhanced Grid view. To switch to Text view, click the **Text View** icon ( ) on the toolbar, or select **View > Text view**.

The following figure provides an example of a SOAP request in XML Spy. The highlighted text represents the value for a method parameter.

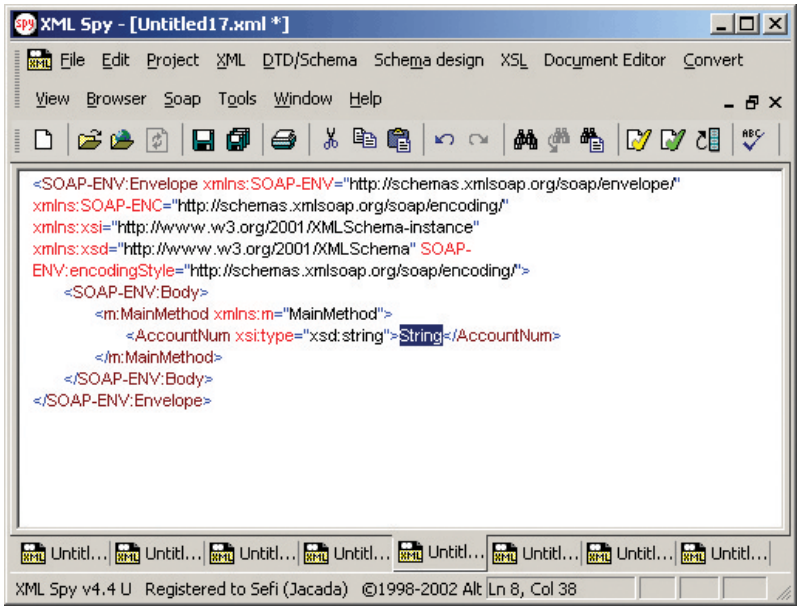


Figure 19. XML Spy: A SOAP request

5 In the SOAP request's body provide the necessary method parameter(s).

6 Select **SOAP > Send request to server**.

7 If there are no errors, you should receive the SOAP response in Enhanced Grid view.

To switch to Text view, click the **Text View** icon ( ) on the toolbar, or select **View > Text view**.

The following figure provides an example of a successful SOAP response.

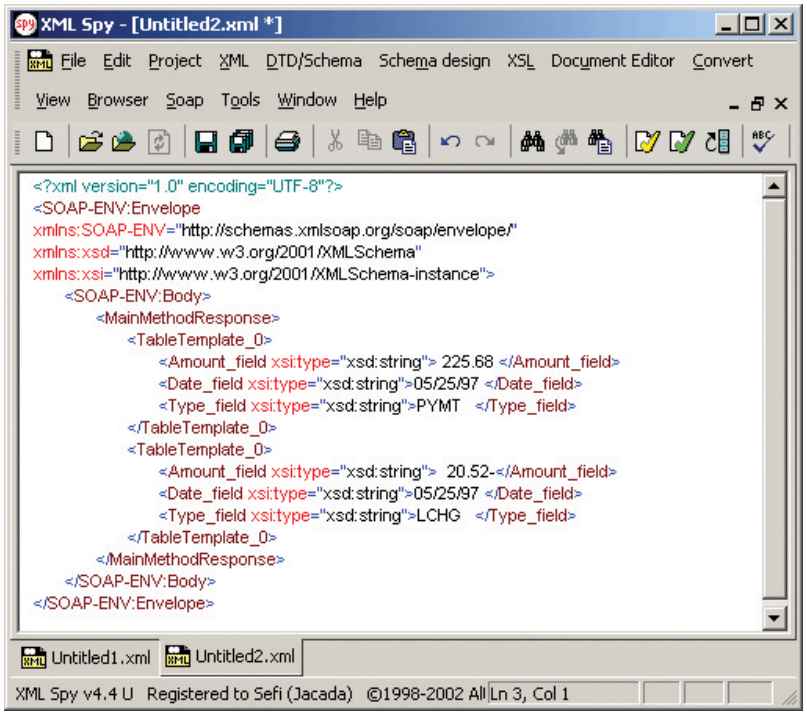


Figure 20. XML Spy: A successful SOAP response

8 If there is an error, you will get an error message box.

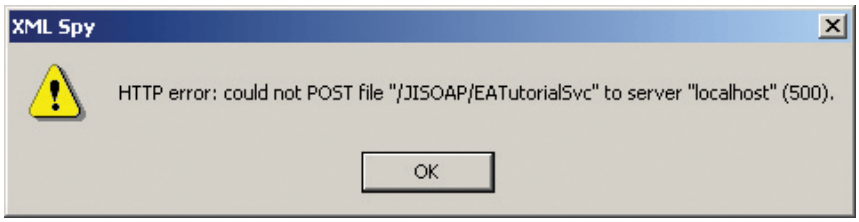


Figure 21. XML Spy: Error message

You can examine the `<SOAP-ENV:Fault>` element to learn about the cause of the error.

Publishing a Web Service to UDDI

The webMethods JI MapMaker tool does not support UDDI publishing. However, after creating your web service, you can publish it to UDDI using one of the following options:

- "Publishing your Web Service via the UDDI API" (on page 106).
- "Publishing your Web Service to UDDI via Web-based Publishing" (on page 107).

Publishing your Web Service via the UDDI API

You can publish your web service directly to a UDDI registry. This is done through direct communication with the UDDI registry's publishing API. The publishing API is divided into three main subsections: authenticating users, saving data, and deleting data.

Following is an overview of the main UDDI publishing functions:

get_authToken	Requests an authentication token from the operator site.
discard_authToken	Requests that the specified authentication token be discarded.
save_binding	Inserts or updates a `bindingTemplate` record.
save_business	Inserts or updates a `businessEntity` record.
save_service	Inserts or updates a `businessService` record.
save_tModel	Inserts or updates a `tModel` record.
delete_binding	Deletes a `bindingTemplate` record specified by the `bindingKey`.
delete_business	Deletes a `businessEntity` record specified by the `businessKey`.
delete_service	Deletes a `businessService` record specified by the `serviceKey`.
delete_tModel	Hides a `tModel` record specified by the `tModelKey`.

To publish to a public production registry, use one of the following URLs:

- Microsoft: https:// uddi.microsoft.com/publish
- IBM: http://www-3.ibm.com/software/solutions/webservices/uddi/

Publishing your Web Service to UDDI via Web-based Publishing

Both Microsoft and IBM provide test registries where you can experiment with publishing new data. It is highly recommended that you experiment with these test registries first, prior to publishing data to live servers.

To access web-based test registries, use one of the following URLs:

- Microsoft: https://test.uddi.microsoft.com
- IBM: https://uddi.ibm.com/testregistry/registry.html

For the web-based production registries, use the following URLs:

- Microsoft: http:// uddi.microsoft.com (Use Internet Explorer for this link)
- IBM: http:// www-3.ibm.com/services/uddi/

Note: This section focuses on publishing a web service on the Microsoft UDDI registry.

To publish a web service on the Microsoft UDDI registry:

1. Login to the Microsoft UDDI registry at http:// uddi.microsoft.com and sign in by performing user authentication. (Use Internet Explorer for this link).
2. Publish a new business entity.
3. Publish a business service. This includes providing binding details.
4. Publish a new tModel record.

Logging onto the Microsoft UDDI Registry

Microsoft provides user authentication via Microsoft Passport. To publish UDDI data, you can either register a new Passport account or enter the login and password for an existing Passport account, such as a *hotmail* mail account.

Figure 22. Microsoft Passport Sign-In

Once logged in via Microsoft Passport, click the **Administer** link on the left part of your screen.

You will see the general UDDI administration screen:

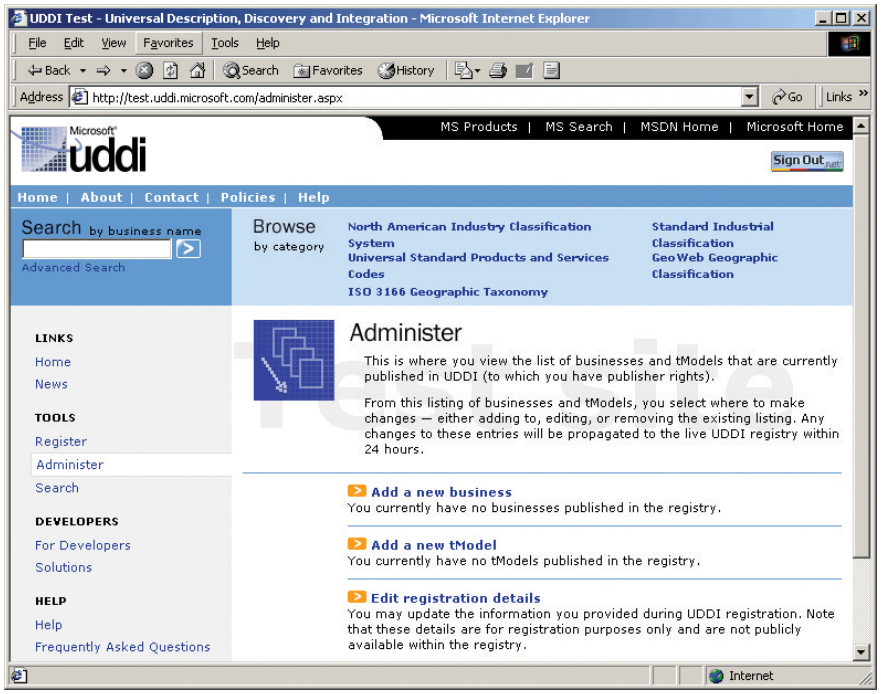


Figure 23. The general UDDI administration screen

Publishing a Business Entity

To add a new business:

1 From the UDDI administration screen, click the **Add a New Business** link. You are now requested to enter business detail.

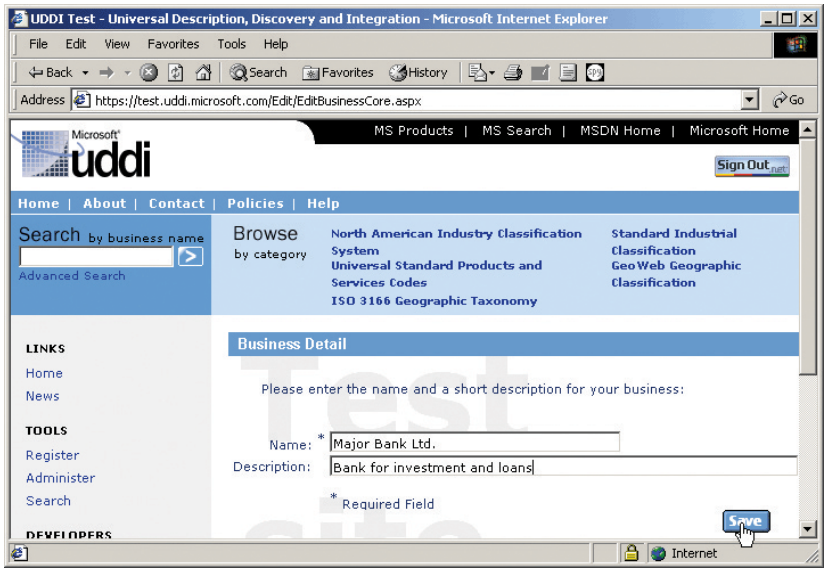


Figure 24. Add a new business entity

2 Fill in the Name and Description and click **Save**.

3 From the Edit Business page, you may set the following:

- Contacts
- Identifiers
- Business classifications

Choosing each of these settings, sends you to another form, which you should fill in and then click the **Continue** button on the bottom right of this page.

4 After providing the desired information, scroll down to the bottom of the Edit Business page until you reach the **Publish** button.

5 Click the **Publish** button to publish your business entity.

Your data is now saved to the registry.

Publishing a Business Service

To add a new business service:

1 Return to the Administration home page. You should now see your published business entity.

2 Select **Edit Business**.

3 Click **Add a Service**.

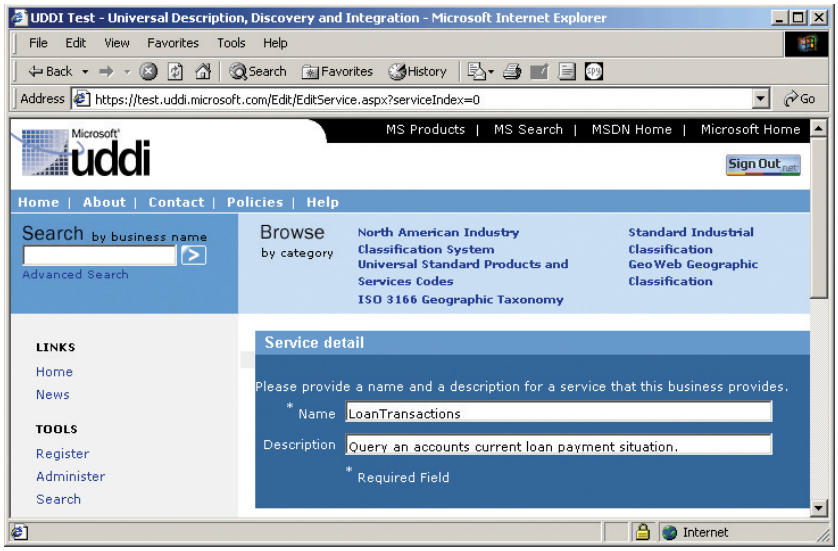


Figure 25. Add a new business service

4 Enter the service name and description, and click **Continue**.

5 From the Edit Service page, click **Define New Binding**.

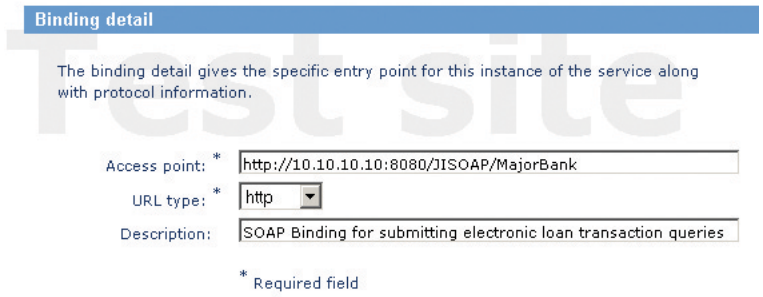


Figure 26. Binding detail

6 Enter the Access point, URL type, and Description, and click **Continue**. The Access point is the URL for connecting to the SOAP gateway on your server. The screen is refreshed with a new section for adding specification signatures.

7 Click **Add Specification Signature**. The Edit Specification Signature page is displayed.

> **Note:** The Edit Specification Signature page defines the relation of your binding template with tModels. In this page you can add information to your tModel data. All fields on this page are optional.

8 Click the **Continue** button several times, until you reach the **Publish** button and click it.

Your business record is now saved.

Publishing a tModel Record

To add a new tModel:

1 In the UDDI Administration page, click the **Add a new tModel** link. A new page is displayed.

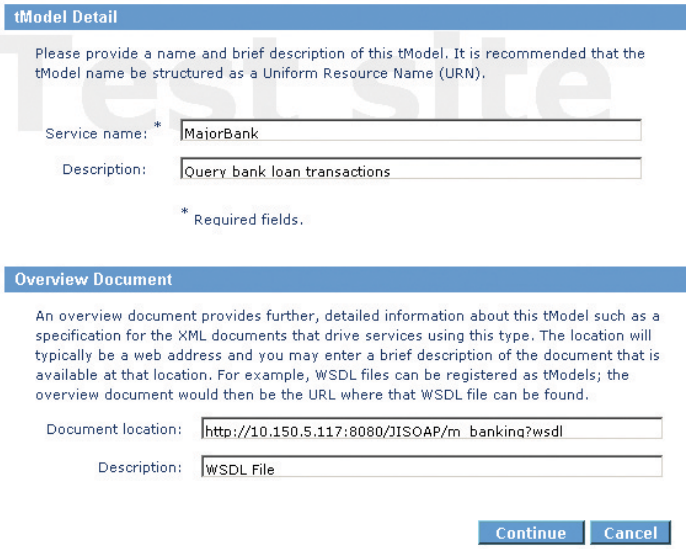


Figure 27. Add a new tModel

2 In the tModel Detail section provide a Service Name and Service Description.

> **Note:**

Note: The name you provide is used by the bindingTemplate as a specification signature.

3 In the Overview Document section provide the location and description of your web services WSDL file.

4 Click **Continue**.

5 Click **Add a Classification** to add a business classification, or click **Add an Identifier** to add identifier information. These settings are optional.

6 Click **Publish**.

Your tModel record is now saved.

Note:

Note: After publishing your web service, you can check it by performing a business inquiry in the UDDI registry.

Chapter 5. MQSeries Integration

MQSeries

IBM's MQSeries™ is a multi-platform messaging and queuing system that provides a communication connection between clients and services that can be running on different operating systems and written with different development environments. MQSeries applications are written using a common interface known as the Messaging Queue Interface (MQI), allowing applications developed on one platform, to be easily transferred to another platform.

Further information about MQSeries can be found on the World Wide Web at http://www-3.ibm.com/software/integration/mqfamily/.

JMS Overview

webMethods JI's implementation of MQSeries integration makes use of the Java Message Service API. JMS provides a standard interface to messaging-oriented middleware (MOM) platforms. JMS also uses Java Naming and Directory Interface (JNDI) which provides a common interface to many directory services (LDAP, NDS, DNS, etc.). This allows the JMS application to look up and locate servers/services, message queues (in the case of MQSeries), and other network resources.

Note: The example provided does not make use of the JNDI features available. It is hard coded for MQSeries connections only.

Note: Automated MQ Host Access via MapMaker-generated services is not supported at this time.

Prerequisites

In order for webMethods JI to communicate with an MQ server using JMS, the MQ server must have JMS libraries installed, configured, and available. The vendor provided JMS libraries must be obtained and are used by the Java

application (in our case the webMethods JI Server, specifically the custom Java service) to communicate with the MQ server using the server's underlying transport layer (e.g. MQ).

See http://www-3.ibm.com/software/integration/mqfamily/library/manuals99/csqzaw/csqzaw.htm for more information about using JMS with MQ Series.

Note that each vendor's MQ implementation for JMS will vary.

Using the JMS MQ Series Test Service Bean

Step 1: MQSeries Setup

If you have MQSeries installed, make sure the MQSeries JMS classes are also installed and available in the system class path for use by webMethods JI. See the documentation included with the JMS classes for more information on which jar files are required.

Note: The example application will not work as provided, unless a queue manager and send/receive queues are created.

Use the MQSeries server administration tool to create a queue manager called "qm" and two queues named "sendqueue" and "recvqueue". These are the send and receive queues used by the sample application.

Note: If you wish to use existing queues, modify the provided service bean to use the names of your queue manager and queues.

Refer to the MQSeries documentation or contact network administration on how to set up these objects within MQSeries.

Step 2: webMethods JI Service Configuration

Create a new default service. See the *webMethods JI User's Guide* for information on adding services. Configure the new service as follows:

- Class name for service bean: `com.jacada.ea.jmstest.JMSMQTestBean`
- Service bean code location: Class path (if the service bean classes are included in the webMethods JI server jar files, otherwise use local file here and point it to the jar file which contains the JMS MQ test classes).

Everything else should be configured as needed (pooled, min instances, timeouts, etc.)

Edit the Environment Manager's *ea_envmgr.lax* file. Use a text editor to add the JMS.jar files in the Java class path, defined under the heading `LAX.CLASS.PATH`. When the file has been edited and saved, stop and restart the Environment Manager to put the class path changes into effect.

Step 3: Start the Echo Application

> **Note:** Modify the following command invocation as required to reflect your own directory structure and installation.

Start the JMS MQ echo application:

```
java -classpath .;"C:\Program Files\MQSeries\java\lib\jms.jar";"C:\Program
Files\MQSeries\java\lib\jndi.jar";"C:\Program
Files\MQSeries\java\lib\ldap.jar";"C:\Program
Files\MQSeries\java\lib\fscontext.jar";"C:\Program
Files\MQSeries\java\lib\com.ibm.mqjms.jar";"C:\Program Files\MQS
eries\java\lib\com.ibm.mqbind.jar";"C:\Program
Files\MQSeries\java\lib\com.ibm.mq.jar";"C:\Program
Files\MQSeries\java\lib\com.ibm.mq.iiop.jar";C:\javastuff\ea3
512\classes\javax_ejb.zip;C:\javastuff\ea3512\classes\mapmaker.jar;C:\javas
tuff\ea3512\cla
sses\ea.jar;C:\javastuff\ea3512\classes\common.jar;C:\javastuff\ea3512\lib\
jclient3.jar;C:\ja
vastuff\ea3512\examples\mqseries com.jacada.ea.jmstest.JMSMQResponder
```

> **Note:** If you wish to use existing queues, modify the provided service bean to use the names of your queue manager and queues.

This acts as a back end data source for the JMS service bean. The MQ JMS service bean is now ready to use by clients.

Step 4: Run the Test Client

> **Note:** Modify the following command invocation as required to reflect your own directory structure and installation.

```
java -classpath .;"C:\Program Files\MQSeries\java\lib\jms.jar";"C:\Program
Files\MQSeries\java\lib\jndi.jar";"C:\Program
Files\MQSeries\java\lib\ldap.jar";"C:\Program
```

```
Files\MQSeries\java\lib\fscontext.jar";"C:\Program
Files\MQSeries\java\lib\com.ibm.mqjms.jar";"C:\Program Files\MQS
eries\java\lib\com.ibm.mqbind.jar";"C:\Program
Files\MQSeries\java\lib\com.ibm.mq.jar";"C:\Program
Files\MQSeries\java\lib\com.ibm.mq.iiop.jar";C:\javastuff\ea3
512\classes\javax_ejb.zip;C:\javastuff\ea3512\classes\mapmaker.jar;C:\javas
tuff\ea3512\cla
sses\ea.jar;C:\javastuff\ea3512\classes\common.jar;C:\javastuff\ea3512\lib\
jclient3.jar;C:\ja
vastuff\ea3512\examples\mqseries
com.jacada.ea.jmstest.EAJMSMQTestClient <env. host> <env. port> <service
name>.
```

<service name> is the configured name of your JMS test bean service. This connects to the service and does a single method invoke of the method named “test” defined in the service bean.

Writing Non-host Service Beans

Service beans not making use of MapMaker's maps, should implement com.jacada.ea.ejb.SessionBean instead of implementing EAServiceBeanIF, or extending EAServiceBean. EAServiceBeanIF defines many methods which are only used for MapMaker TN-based maps, and do not need to be implemented by other beans which do not use MapMaker maps.

webMethods JI accepts SessionBean as a valid bean component type for java services. The service uses introspection to discover what capabilities and interfaces the deployed bean component supports.

Example Custom Java Service

A custom webMethods JI service class file is provided as an example. This file shows the Java code required to provide access to an MQSeries host application, using the MQSeries JMS API.

The following code is the test service bean provided with the webMethods JI release. It can be found in the webMethods JI server installation directory *<JI_install_dir>/examples/jms/mqseries/com/jacada/ea/jmstest* in the file `JMSMQTestBean.java`.

```
package com.jacada.ea.jmstest;

/*
 * Copyright (c) 2000,2010 by Software AG. All Rights Reserved.
 *
 *
```

```
 */

import java.io.*;
import java.util.*;
import javax.jms.*;
import javax.ejb.*;
import com.ibm.mq.jms.*;
import com.jacada.ea.ejb.server.*;

/** JMSMQTestBean is an example service bean which makes use of the MQ
series
 *  JMS implementation to send and receive data from a JMS/MQ application.
 *
 *  @author Software AG
 *  @see EASessionBean
 */
public class JMSMQTestBean implements EASessionBean {
       java.util.Map vars = null;              //Our method input parameters
        PrintWriter writer = null;               //Our log output stream
        QueueSession session = null;             //The JMS session object
        QueueSender sender = null;               //Our JMS queue sender
        QueueReceiver receiver = null;           //Our JMS queue receiver
        QueueConnection queueConnection = null;  //Our JMS queue connection
object


        public JMSMQTestBean() {
                vars = new HashMap();     //Initialize the input parameters
table
        }

        /** Close the connection to the backend datasource.
         *  @return       True if sucessful, false if disconnect failed
         *  @throws       IOException if an error occurs during disconnect
process.
         */
        public boolean closeConnection() throws IOException {
                // Make sure we close and cleanup all JMS object references
otherwise
                // they will continue to consume system memory and other MQ
resources
                try {
                        session.close();
```

```
                        sender.close();
                        receiver.close();
                        queueConnection.stop();
                        queueConnection.close();
                }
                catch(JMSException jmsx) {
                        throw new IOException(jmsx.toString());
                }
                session = null;
                sender = null;
                receiver = null;
                queueConnection = null;
                return true;
        }

        /** Invoke a method.  Send a message to the send queue and
         *  read a message back from the receive queue.
         *  @return A Map object containing our method invoke results
         *  @throws JMSException if something goes wrong sending or receiving
         *          JMS messages from the queues
         */
        protected java.util.Map doMethodInvoke() throws JMSException {
                if(sender != null) {
                        // Create a new map message object
                        MapMessage msg = session.createMapMessage();
                        if(vars != null) {
                                Iterator i = vars.keySet().iterator();
                                while(i.hasNext()) {
                                        // Set input parameters as message
parameters
                                        String key = (String)i.next();
                                        Object val = vars.get(key);
                                        if(val != null) {
                                               // Set the message field and
flatten the object
                                                // value to a string type
                                                msg.setObject(key,
val.toString());
                                        }
                                }
                        }
                        else {
```

```
                                msg.setObject("message","This is a test");
                        }

                        // Send the message to the send queue
                        sender.send(msg);
                        log("Sent message " + msg);

                        // Clear our input variables
                        vars.clear();

                        // Get a message off the receive queue
                        Message m = receiver.receive();
                        log("Got message " + m);

                      // Parse the return message parameters into the method
return map
                        if(m instanceof MapMessage) {
                                java.util.Map resp = new HashMap();
                                MapMessage reply = (MapMessage)m;
                                Enumeration e = reply.getMapNames();
                                while(e.hasMoreElements()) {
                                       String key = (String)e.nextElement();
                                        resp.put(key, reply.getObject(key));
                                }
                                return resp;
                        }
                }
                return null;
        }

        public void ejbActivate() {
                // EJB activate code goes here, excluded for this example
        }

        public void ejbPassivate() {
                // EJB passivate code goes here, excluded for this example
        }

        public void ejbRemove() {
                // EJB remove code goes here, excluded for this example
        }
```

```
/** Get the host name/address property
 *  @return  A String containing the backend datasource address
 */
public String getHost() {
        return "eatest";
}

/**
 * Get the value of an input parameter
 * @return java.lang.Object
 * @param methodName java.lang.String
 * @param varName java.lang.String
 */
public Object getInputParameter(String methodName, String varName) {
        // left blank for this example
        return null;
}

/** Get input parameter metadata
 *  @params     The method name to get input parameters for
 *  @return     An array of EAMethodParameterIF objects describing
 *          the input parameters
 */
public EAMethodParameterIF[] getInputParameters(String s) {
        // left blank for this example
        return null;
}

/** Get output parameter metadata
 *  @params     The method name
 *  @return     An array of EAMethodParameterIF objects describing
 *          the method output metadata
 */
public EAMethodParameterIF[] getOutputParameters(String s) {
        // Left blank for this example
        return null;
}

public int getPort() {
        return -1;
}
```

```
        public String getSessionName() {
                return null;
        }

        /** Get a list of method names this bean supports
         *  @return     A String array of method names
         */
        public String[] getSupportedMethodNames() {
                String s[] = {"test"};
                return s;
        }

        public long getTimeout() {
                return -1;
        }

        public int getTraceLevel() {
                return 0;
        }

        /** Check if the bean supports the given method
         *  @param name         A method name
         *  @return                     True if the bean supports the given
method, false if not
         */
        public boolean hasSupportedMethod(String name) {
                if(name.equals("test")) return true;
                return false;
        }

        /** Invoke a specific method of the service bean
         * @param name  The name of the method to invoke
         * @return      A Map containing the method invoke return parameters
         * @throws Exception    If the method invoke fails
         */
        public java.util.Map invoke(String name) throws Exception {
                if(name.equals("test")) {
                        return doMethodInvoke();
                }
                return null;
        }
```

```
        /** Set the input parameters for a method and invoke a specific
         *  method of the service bean
         * @param name  The name of the method to invoke
         * @param params        The method input parameters
         * @return      A Map containing the method invoke return parameters
         * @throws Exception    If the method invoke fails
         */
        public java.util.Map invoke(String name, java.util.Map params)
throws Exception {
                if(params != null) {
                        Iterator i = params.keySet().iterator();
                        while(i.hasNext()) {
                                String key = (String)i.next();
                                Object val = params.get(key);
                                setInputParameter(name, key, val);
                        }
                }
                return invoke(name);
        }

        /** Is the service bean connected to a back end datasource or not?
         *  @return     true if connected, false if not
         */
        public boolean isConnected() {
                if(session != null) return true;
                return false;
        }

        /** Get the state of the service bean's verbose mode
         *  @return     true if on, false if off
         */
        public boolean isVerbose() {
                return false;
        }

        /** Output a log to the logging output stream.  If the bean
         *  is running in the E/A environment, logging to the writer
         *  object will go to the service log file (if any).
         *  Writing to System.out will send the log message
         *  to the jcluster log file (if any).
         *  @param msg  The message to log
         *
```

```
         */
        public void log(String msg) {
                if(writer != null) writer.println(msg);
                else System.out.println(msg);
        }

        /** Open and initialize our queue connections
         *  @return true if successful, false if not
         *  @throws IOException If connection initialization failed
         */
        public boolean openConnection() throws IOException {
           try {
                  // Normally you would look up the queue connection factory
and queues
                   // via JNDI for portable code across multiple JMS server
types.  Here
                 // we are simply instantiating the queue factory directly.
Please
                   // refer to the JNDI reference or to the
                  // documentation for your specific JMS library if you wish
to use
                   // JNDI to lookup queue and queue connection factory
references.

                   QueueConnectionFactory queueConnFactory = new
MQQueueConnectionFactory();

((MQQueueConnectionFactory)queueConnFactory).setQueueManager("qm");

                   // Get a queue connection
                   log("Creating Queue connections");
                   queueConnection =
queueConnFactory.createQueueConnection();
                   queueConnection.start ();

                   // Get a queue session
                   log("Creating queue session: not transacted, auto ack");
                   session = queueConnection.createQueueSession(false,
Session.AUTO_ACKNOWLEDGE);

                   // Get queue references
                   log("Creating Queue, senders");
                   // Queue we will write to ...
                   Queue sendQueue = session.createQueue("sendqueue");
```

```
                // Queue we will read from ...
                Queue rcvQueue = session.createQueue("recvqueue");
                sender = session.createSender(sendQueue);
                receiver = session.createReceiver(rcvQueue);
        }
        catch (Exception e) {
                e.printStackTrace ();
                throw new IOException(e.toString());
        }
        return true;
    }

    public boolean openConnection(java.util.Map connectParams) throws
IOException {
            return openConnection();
    }

    /** Return the backend datasource connection to a sane state to
prepare
     *  for a new client.
     *  @throws        Exception if an error occurred during the reset
process.
     */
    public void resetConnection() throws Exception {
            // not implemented in this example
    }

    public void setHost(String hostName) {
            // not implemented in this example
    }

    /** Set our method input parameters
     *  @param name The method input parameter name
     *  @param type The value of the method input
     */
    public void setInputParameter(String name, Object type) {
            vars.put(name, type);
    }

   public void setInputParameter(String mName, String name, Object type)
{
            setInputParameter(name, type);
    }
```

```
        public void setPort(int port) {

        }

        public void setSessionContext(SessionContext ctx) {
                // Set the EJB session context.  Not implemented for this
example.
        }

        public void setSessionName(String sessionName) {
                // not implemented in this example
        }

        public void setTimeout(long timeout) {
                // not implemented in this example
        }

        public void setTraceLevel(int traceLevel) {
                // not implemented in this example
        }

        public boolean setVerbose(boolean val) {return true;}

        public void setWriter(Writer pw) {
                if(pw instanceof PrintWriter) {
                        writer = (PrintWriter)pw;
                }
                else {
                        writer = new PrintWriter(pw);
                }
        }
}
```

Chapter 6. Siebel 7 Integration

Siebel 7

Siebel System's Siebel 7 is a fully integrated suite of applications for customer relationship management (CRM)—including applications for partner relationship management (PRM) and employee relationship management (ERM)—all based on a common Web architecture. By providing organizations with real-time analytical ability to track and understand both customer behavior and key indicators of corporate performance, Siebel 7 enables organizations to be "digitally wired" to their customers, partners, and employees.

Further information about Siebel can be found on the World Wide Web at http://www.siebel.com/products/.

Siebel eBusiness Applications

Siebel provides a comprehensive family of eBusiness application solutions that provide the connections to the Siebel database for every category of sales, marketing, and customer service.

The following eBusiness applications are part of the Siebel suite of applications:

- Siebel Sales
- Siebel Service
- Siebel Marketing
- Siebel Call Center
- Siebel Field Service
- Siebel eMail Response
- Siebel Pricer
- Siebel Professional Services
- Siebel Finance

The following Siebel-related terms are explained below:

- Siebel Applets
- Siebel Business Components
- Siebel Call Center

Siebel Applets

Siebel applets are “visual application units” that appear on the screen of the end user. They are used by the Siebel eBusiness applications to allow the user to enter, view, and navigate through data.

Siebel applets should not be confused with Java applets. Java applets, which operate in Web browsers, are developed using the Java programming language, while Siebel applets are developed in Siebel Tools, with or without the need to write code. However, there are some similarities between Siebel applets and Java applets. For example, Siebel applets are generally self-contained and perform specific roles, similar to a Java applet. But rather than being displayed only in a Web browser, they can be displayed in one or more Siebel applications.

Siebel Business Components

A business component is a single, logical business entity that accesses data that relates to the specific business operation being addressed by the business component. This allows applets to reference data from business components rather than having to access the database tables directly, making it easier to develop applets that are based on similar business operations. Relationships between different business components can be established, and many different business objects can access a single business component.

A business component not only encapsulates data but also enables the developer to define the behavior of the entity.

The examples provided in this document use business components as the primary vehicle to retrieve the webMethods JI data. This is useful when using webMethods JI to access legacy data from a Siebel Thin Client for Windows or when multiple applets need to share the same business logic.

Siebel Call Center

A Siebel eBusiness Application is comprised of various components. These are viewable using the Siebel Call Center. Various interface elements in a Siebel eBusiness application are shown in the following image:

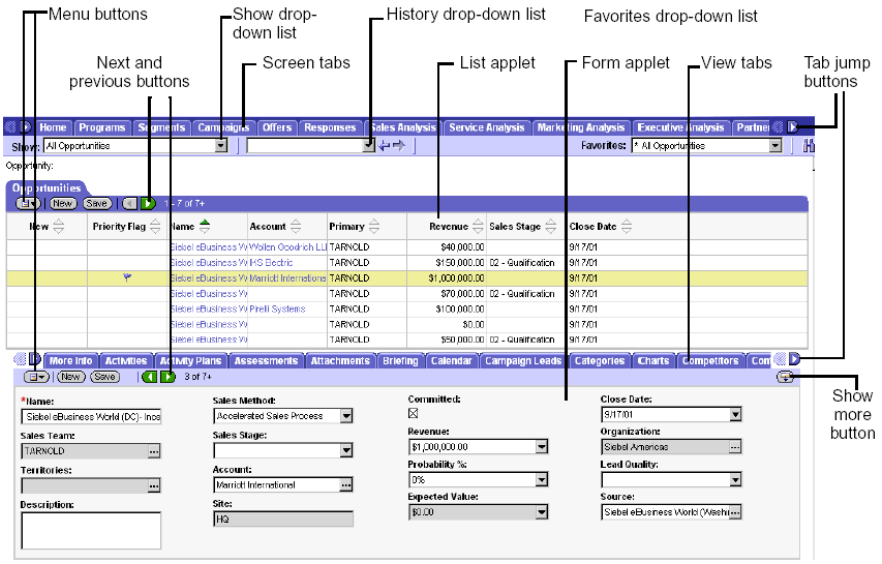


Figure 28. A Siebel eBusiness Application viewed through the Siebel Call Center

Following is an explanation of the Siebel eBusiness Application components:

- **Application:** An application is a collection of screens. The application is opened in a Web browser on the user's desktop by attaching to a specified URL. The screens are accessed from the tab bar and the Site Map, as defined in the application. Siebel eService is an example of an application. Each combination of screens that is appropriate to a specific class of users can be provided as an application.
- **Page Tab:** A page tab object definition associates a screen to the page tab's parent application object definition and includes it as a tab in the tab bar.
- **Screen Menu Item:** A screen menu item object definition associates a screen to the application and includes the screen as a menu item in the Site Map.
- **Screen:** A screen is a logical collection of views. It is not a visual construct in itself; rather, it is a collection of views that the menu bar and view bar can display. The active screen is selected from the Site Map or the tab bar.
- **Screen View:** A screen view object definition associates a view with the screen view's parent screen object definition. This is how views are included in screens.
- **View:** A view is a collection of applets which appear on screen at the same time. A view can be thought of as a single window's worth of related data forms (applets). The Siebel application window displays one view at any one time. The user can select the current (active) view from the second-level navigation tab or from the Site Map. A view is associated with the data and relationships in a single business object.
- **Applet:** An applet is a form, composed of controls, that occupies a portion of the Siebel application window. An applet can be configured to allow data entry, provide a scrolling table of business component records, or display business graphics, a navigation tree, or a similar user interface unit. It provides

viewing, entry, modification, and navigation capabilities for data in one business component. Popup windows for multi-value groups and record selection are also implemented as applets.

- **Control:** One control object definition corresponds to one data control in a form applet, such as a text box, check box or command button. A control is something in the applet with which the user can interact. A control usually either exposes data from one field in the business component, or invokes programming logic (in the case of a Push Button control).
- **List:** List is a child object type of Applet. A list object definition specifies property values that pertain to the entire scrolling list table and provides a parent object definition for a set of list columns.
- **List Column:** A list column object definition corresponds to one column in the scrolling list table in a list applet, and to one field in the business component.
- **Web Template, Applet Web Template, View Web Template:** Identify external HTML (or other markup language) files that define the layout and Siebel Web Engine interactions for an applet or view.
- **Applet Web Template Item:** Defines a control, list item, or special Web control in the Web implementation of an applet.
- **View Web Template Item:** Defines the inclusion of an applet in the Web implementation of a view.

Siebel Tools

Siebel Tools is an object-oriented client/server application development environment. Siebel Tools includes a Siebel VB development environment that allows Siebel developers to rapidly create and deploy business objects for users of Siebel eBusiness applications.

The following image points out the components of the Siebel Tools user interface:

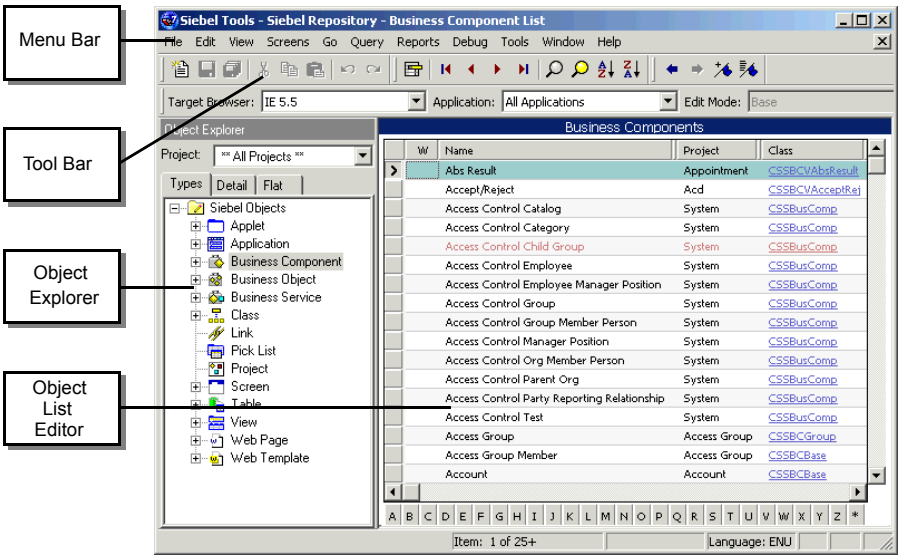


Figure 29. The Siebel Tools Interface

Installing Siebel Tools

To install Siebel Tools on a workstation:

1. Navigate to the *\sea7xx\tools* directory on the Siebel eBusiness Applications Client CD-ROM in Windows Explorer.
2. Double-click *setup.exe* to start the Siebel Tools installation program.
3. For setup language, choose **English**.
4. In the **Welcome** dialog box, click **Next**.
5. In the **Setup Type** dialog box, select the type of Siebel Tools installation to install on this PC:
   - **Typical Setup:** Install all Siebel Tools components. This option is recommended for most users.
   - **Compact Setup:** Install all modules except the help files, report source code, and demonstration database.
   - **Custom Setup:** Lets you customize your installation by choosing among different components. Each component is listed with the amount of disk space it requires. Click the **Disk Space** button to see how much disk space is available on the hard drive and network drives that are accessible from this PC. This option is recommended for experienced administrators only.

Note: A warning appears if there is insufficient disk space to install Siebel Tools on the destination host machine. In this case, you must free some disk space before continuing with the installation process.

Select a destination directory. The default directory for Siebel Tools is *C:\sea7xx\tools*.

Note: The Installer does not permit installing Siebel Tools into a directory path containing more than 19 characters (for example, M:\SIEBEL\10910\TOOLS).

Integrating webMethods JI with Siebel

There are two different ways of integrating webMethods JI with Siebel:

- **Integration using a Virtual Business Component:** In an existing eBusiness Application, a Virtual Business Component is created. This Virtual Business Component is added to a Business Service in the application, and is set to communicate with webMethods JI through the webMethods JI HTTP Gateway Servlet. This is explained and detailed in "Integration using a Virtual Business Component" on page 135.
- **Integration using Integration Objects:** In an existing eBusiness Application two Integration Objects are created. One Integration Object is internal and the other is external. The two Integration Objects are mapped to each other, and a workflow is created to manage the flow of information between them. Communication with the webMethods JI Server is done with the use of XML documents. This is explained and detailed in "Integration Using Integration Objects" on page 156.

To Integrate webMethods JI with Siebel you must do the following:

1. Set up the webMethods JI environment.
2. Set up the HTTP Gateway servlet environment.
3. Set up the Siebel environment.
4. Set up Siebel Tools.
5. Customize the Siebel eBusiness Application using Siebel Tools.
6. Bring up Siebel Call Center to test the Siebel eBusiness Application.

Setting up the HTTP Gateway Servlet Environment

The webMethods JI XML HTTP Gateway is a servlet than can run in any servlet engine (JRun, Apache/Tomcat, etc.). The Servlet properties file contains configurable properties that govern the servlet behavior and all parameters to execute a webMethods JI Service and Method.

The properties file is specified in the HTTP request header from the client: http://localhost:8880/EAi/httpgateway/config/Siebel_eaihttp.cfg

The webMethods JI Demo Kit contains property files to handle XML messages in the following Siebel integration implementations:

- Messages from a custom business service that formats XML in a format that webMethods JI can handle. A custom business service solution also knows how to handle messages returned from webMethods JI.
- Generic XML messages when using the Siebel XML Gateway business service in conjunction with Siebel Virtual Business Components.

This file must be located in the HTTP Gateway Servlet configuration directory:

<HTTP_gateway_inst_dir>\WEB-INF\config

For more information about installing and configuring the HTTP Gateway Servlet, refer to the *HTTP Gateway User Guide.*

Note: The integration examples provided in this chapter assume that the HTTP Gateway is properly configured and running.

Setting up the Siebel Environment

The following instructions assume that the Siebel 7 eBusiness Application and Siebel 7 Tools environments have been installed and configured properly. All changes to the Siebel environment are made using Siebel Tools. The examples provided in this chapter are for Siebel 7 Call Center. When using other Siebel 7 eBusiness Applications the basic steps presented here will still hold true but the developer will need to work with the views, applets and business components specific for that particular Siebel eBusiness Application.

The following steps are required to set up the Siebel Environment:

1. Setup the Siebel web client(s).
2. Setup the Siebel Data Base.
3. Setup Siebel Tools.

Setting up the Siebel Web Client

To install the Siebel Web Client:

1. Open *uagent.cfg* configuration file located under *C:\sea7xx\client\BIN\ENU*.
2. Change the **RepositoryFile** setting from **siebel.srf** to **softwareag.srf**. \
3. Scroll down to the **[Sample]** section and copy the **ConnectString** entry.
4. Save and close the *uagent.cfg* configuration file.
5. Open the *tools.cfg* configuration file located under *C:\sea7xx\tools\BIN\ENU*.
6. Scroll down to the **[Sample]** section and replace the **ConnectString** entry copied in step 3.

7 Save and close the *tools.cfg* configuration file.
8 In the *C:\sea703\client\OBJECTS\ENU* directory, create a copy of the *siebel.srf* file and rename it to *"softwareag.srf"*.

Setting up the Siebel Data Base

To set up the Siebel Data Base:

1 Modify the Siebel Tools shortcut from the Windows Start menu. Find the short cut under **Start—Programs—Siebel Tools 7.0.x—Siebel Tools**. Right-click and select **Properties**. At the end of the **Target** field add the following entry to the existing one:

```
/editseeddata /u sadmin /p sadmin /d sample
```

Note: Make sure there is a space between the existing entry and the new entry.

2 Click **Apply** and **Close**.
3 Modify the Siebel Call Center shortcut from the Windows Start menu. Find the shortcut under **Start > Programs > Siebel Client 7.0.x > Siebel Call Center - ENU**. Right-click and select **Properties**. At the end of the Target field add the following entry to the existing one:

```
/u sadmin /p sadmin /d sample
```

Note: Make sure there is a space between the existing entry and the new entry.

4 Click **Apply** and **Close**.
5 Bring up the Siebel Call Center by selecting **Start > Programs > Siebel Client 7.0.x > Siebel Call Center - ENU**. You will get two or three dialogs.
6 Check **Always trust content from Siebel Systems**.
7 Click **Yes**.
8 Close the Siebel Call Center.

Setting-up Siebel Tools

Assuming it is already installed on your system, Siebel Tools must be configured to work with the deployment browser.

Follow these steps to setup Siebel Tools to use Internet Explorer 5.5 and later versions:

1 In Siebel Tools, select **View > Toolbars > Configuration Context…**

The configuration toolbar is displayed.

2 From the configuration toolbar's **Target Browser** field, select **Target Browser config…**

The **Target Browser Configuration** dialog box opens:

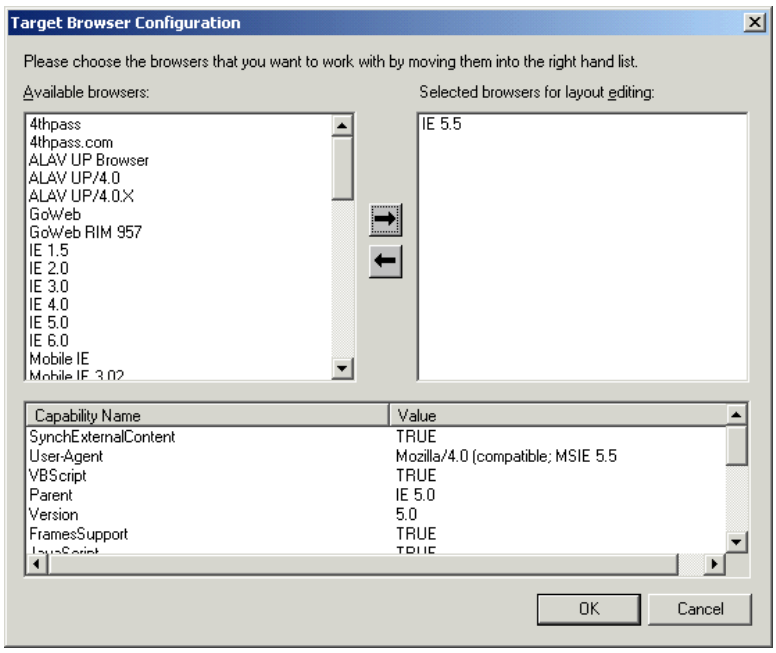


Figure 30. Target Browser Configuration dialog box

3 Select **IE 5.5** from **Available browsers** and move it to **Selected browsers** for layout editing area by clicking on the right-arrow (➡) button.

4 Click **OK** to close the dialog box.

5 Select **View > Toolbars > Configuration Context…** to close the configuration toolbar.

Integration using a Virtual Business Component

The XML Gateway Business Service is a standard Business Service supplied by Siebel in the EAi Module. A Virtual Business Component (VBC) uses a Business Service that responds to `init`, `query`, `delete`, `preinsert`, `insert` and `update` requests, which the Virtual Business Component generates.

The XML Gateway Business Service contains methods that transform standard VBC requests into XML representations. These XML messages are transformed, using XSL transformation, into XML messages that webMethods JI can understand. Likewise, the reply from webMethods JI must be transformed into an XML message that the XML Gateway business service can understand.

This is all set up using Siebel Tools. Instructions for this are provided in the in subsequent sections.

The following diagram illustrates integration using virtual business components:

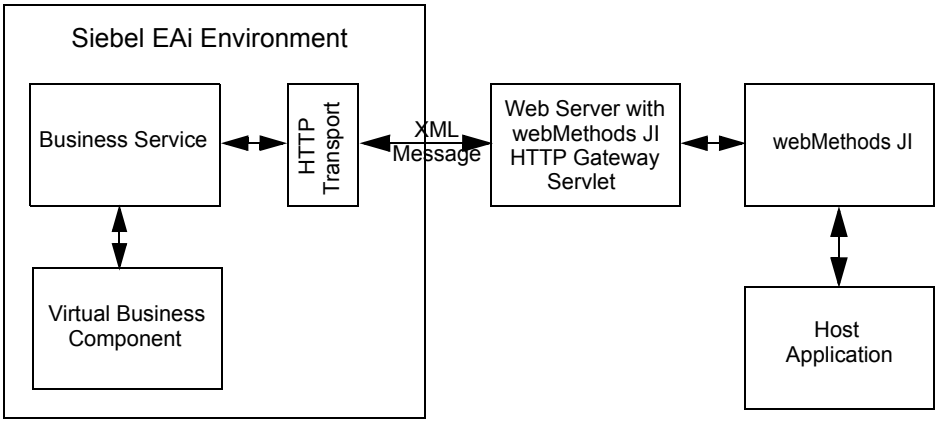


Figure 31. Runtime architecture - integration using Virtual Business Components

The Siebel EAi Environment

The Siebel EAi Environment consists of:

- A Business Service.
- A Virtual Business Component.

The Virtual Business Component

As stated above, a Virtual Business Component will issue `init`, `query`, `delete`, `preinsert`, `insert` or `update` requests depending on the operation performed. When a Virtual Business Component is first displayed, an `init` request followed by a `query` request are issued to obtain the data to display. A `query` request can also be issued by the end-user. While inserting a new record, a `preinsert` request is issued. When the new record is saved, an `insert` request is issued. As their names imply, the `delete` and `update` requests are issued when a record is deleted or updated.

The Business Service

To communicate with webMethods JI using Virtual Business Components, we create a custom-written business service. A custom-written business service formats XML messages in a format that webMethods JI can understand without performing an XSL transformation. Likewise, this custom-written business service can be written to understand the returned XML message with performing XSL transformation.

Development in Siebel Tools

To integrate webMethods JI with Siebel, you must perform the following steps in Siebel Tools:

1. **"Creating a New Project":** In this step, you create a new project. This project will include all of the new settings and necessary configurations for integration with webMethods JI. In the example provided in this document, a new project named: "`Software AG Integration`" is created.
2. **"Modifying an Existing Business Component":** In this step, you modify an existing business component by adding a new field to it. This new field is used to specify information to be obtained from the legacy application. In the provided example, the `Contact` business component is modified, by adding a new field named "`Account Number`". This field is used to obtain checking transactions information from the `MajorBank` legacy application.
3. **"Modifying a Business Component's Associated Applets":** In this step, you modify one of the applets associated with the business component to display the new field. In the provided example, the `Contact Profile Applet` applet is modified to include a new control representing the `Account Number` field previously created.
4. **"Creating a Virtual Business Component":** In this step, you create a new virtual business component. In the provided example, a virtual business component is created named "`JCDA Account VBC`".
5. **"Creating Fields in the Virtual Business Component":** In this step, you create fields for the new virtual business component. These fields will display information derived from the legacy application. In the example provided in this document, five fields are created named: `Account Number`; `Amount`; `Date`; `Description`; `Detail`.
6. **"Setting the Virtual Business Component's User Properties":** In this step, you set the virtual business component's user properties.
7. **"Linking the Modified Business Component to the New Virtual Business Component":** In this step, you create a link between the previously modified Siebel business component and the newly created virtual business component. In the provided example, a link is created between the `Contact` business component and the `JCDA Account VBC` virtual business component. This link will be named: "`Contact/JCDA Account VBC`"

8 **"Modifying an Existing Business Object":** In this step, you modify a standard Siebel business object to include the new virtual business component and the new link that ties it to the standard Siebel business object. In the provided example, the `Contact` business object is modified to include the `JCDA Account VBC` virtual business component and the `Contact/JCDA Account VBC` link.

9 **"Creating a New List Applet":** In this step, you create a new list applet so that the Siebel application be able to display the new virtual business component in its web layout. In the provided example, a new List applet is created named: "`JCDA Transaction Detail`".

10 **"Adding Fields to the New List Applet":** In this step, you add fields to the new list applet to identify the fields from the new virtual business component. In the provided example, the following fields are added to the `JCDA Transaction Detail` list applet: `Detail`; `Description`; `Date`; `Amount`.

11 **"Creating a Web Layout for the Applet":** In this stage, you create a web layout for the new list applet. This defines how the new list applet is physically displayed on a web browser.

12 **"Adding the New Applet to the View":** In this step, you add the new applet to the view. This defines where the applet is to be placed in the existing web layout.

13 **"Compiling the Project":** In this step, you compile the whole project in Siebel Tools. In the provided example, the `JCDA Integration` project is compiled.

14 **"Testing the Software AG Integration with Siebel":** In this step, you bring up the Siebel Call Center, and check to see that the Siebel client is properly integrated with Software AG. In the provided example, once connected to the Siebel Call Center, the **Profile** tab under the **Contacts** tab will contain the webMethods JI host information.

Creating a New Project

In Siebel Tools, you create a new project. This project will contain all of the subsequent settings and configurations necessary for integration with webMethods JI.

To create a new project:

1 In the Object Explorer, select **Project**. The **Projects** window is displayed:

2 In the **Projects** window, right-click and select **New Record** from the popup menu.

3 Change the values of the following columns for this new project:

Field	Value
Name	JCDA Integration

Field	Value
Locked	<checked>

Note: Setting the check box in the Locked column allows you to make changes to the project.

The new project is created and locked.

Modifying an Existing Business Component

You modify an existing Siebel business component, to enable the display the information from the legacy application through one of the Siebel client's existing components. In the following example, the `Contact` business component is modified to store and display an account number representing the checking account number for a contact. This is done by creating an `Account Number` field, which will be used to obtain checking transactions information from the `MajorBank` legacy application.

To modify the business component:

1. In the Object Explorer, select **Business Component**. The **Business Components** window is displayed.
2. In the **Business Components** window locate and select the `Contact` business component.
3. Lock the project containing the `Contact` business component by selecting **Tools > Lock Project**. This makes it possible to modify the `Contact` business component.
4. In the Object Explorer expand the `Business Component` branch by clicking on the **[+]**.
5. Select the `Field` branch. The **Fields** window is displayed under the **Business Components** window.
6. In the **Fields** window, right-click and select **New Record** from the popup menu.
7. Change the values of the following columns for this new Field record:

Field	Value
Name	Account Number
Join	S_CONTACT_X

Field	Value
Column	ATTRIB_34

The new `Account Number` field is created.

Modifying a Business Component's Associated Applets

You modify an applet associated with the previously modified business component. The applet is modified so that the new field(s), added to the business component, will be displayed by the Siebel client. In the following procedure, the `Contact Profile Applet` applet is modified to display the `Account Number` field previously added to the `Contact` business component.

To modify the applet:

1 In the Object Explorer, select **Applet**. The **Applets** window is displayed.

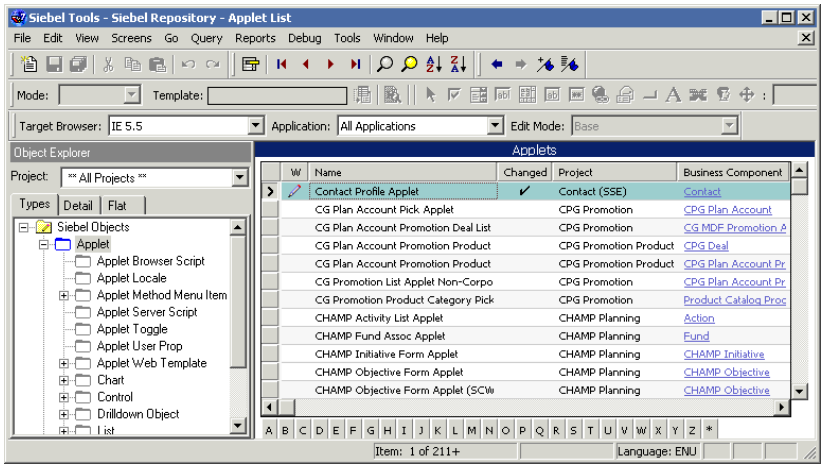


Figure 32. The Applets window

2 In the **Applets** window, locate and select the `Contact Profile Applet` applet.

3 Lock the project containing the `Contact Profile Applet` applet by selecting **Tools > Lock Project**. This makes it possible to modify the `Contact Profile Applet` applet.

4 In the Object Explorer, expand the `Applet` branch by clicking on the **[+]**.

5 Select the `Control` branch. The **Controls** window is displayed under the **Applets** window.

6 In the **Controls** window, right-click and select **New Record** from the popup menu.

7 Change the values of the following columns for this new control record:

Field	Value
Name	Account Number
Caption	Account Number
Field	Account Number

Creating a Virtual Business Component

You create a new virtual business component. Through this virtual business component, Siebel communicates with webMethods JI. The following procedure creates a new business component named "`JCDA Account VBC`".

To create the new business component:

1 In the Object Explorer, from the Siebel Object tree, select **Business Component**. The **Business Components** window is displayed:

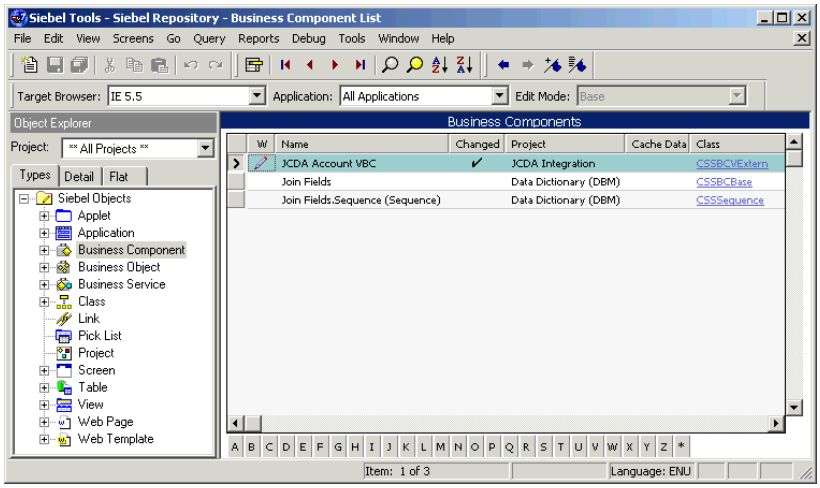


Figure 33. Business Components window

2 Right-click on the **Business Component** window and select **New Record** from the popup menu provided.

3 Change the values of the following columns for this new virtual business component:

Field	Value
Name	JCDA Account VBC

Field	Value
Project	JCDA Integration
Class	CSSBCVExtern

Creating Fields in the Virtual Business Component

To provide functionality, you must create fields in the virtual business component. These fields contain different data types acquired from the legacy application and will be displayed as part of the Siebel client's web layout. The following procedure creates five fields in the `JCDA Account VBC` virtual business component.

To create the fields:

1 While the `JCDA Account VBC` virtual business component is still selected, expand the **Business Component** object and select **Field**. The **Fields** window is displayed below the **Business Components** window:

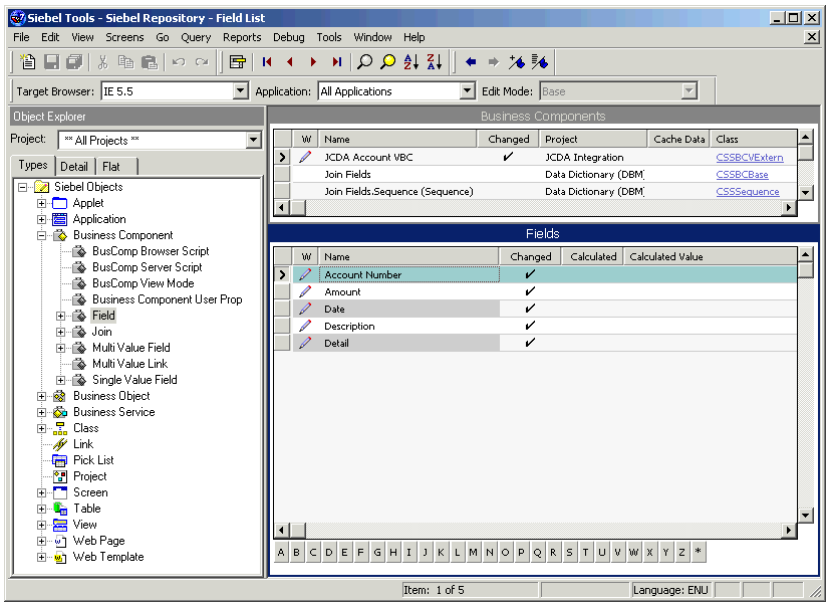


Figure 34. The Fields window

2 Right-click on the **Fields** window and select **New Record** from the popup menu.

3 Enter the name of the field and repeat steps 2—3 for each of the following field names: `Account Number`; `Amount`; `Date`; `Description`; `Detail`.

The fields are now added to the virtual business component.

Setting the Virtual Business Component's User Properties

Setting a virtual business component's user properties communicates a property value to C++ code that implements specialized business component behavior. The values of these properties can be changed during configuration. These values reside in the Siebel repository file. The following procedure sets user properties in the `JCDA Account VBC` virtual business component.

To set user properties in the virtual business component:

1 In the Object Explorer under the **Business Component** branch, select **Business Component User Prop**. The **Business Component User Properties** window is displayed below the **Business Components** window:

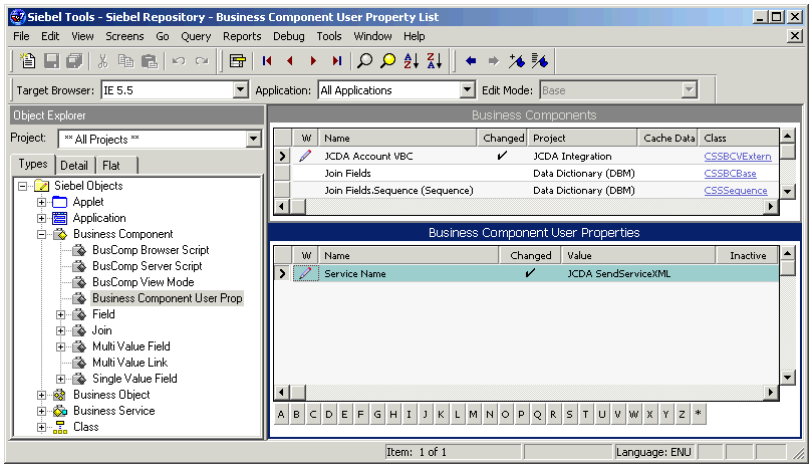


Figure 35. The Business Component User Properties window

2 Right-click on the **Business Component User Properties** view and select **New Record**.

3 Enter the following information (You must repeat step 2 for each new entry):

Name	Value
Service Name	XML Gateway
Remote Source	webMethods JI

Name	Value
Service Parameters	Transport=EAI HTTP TRANSPORT;HTTPRequestURLTemplate= http:/localhost:8880/EAi/httpgateway/config/MajorBankVBC.cfg; HTTPRequestMethod=POST;HTTPContentType=text/plain (**Note:** The value of `HTTPRequestURLTemplate` should contain the correct IP for the Siebel Gateway.)

Linking the Modified Business Component to the New Virtual Business Component

A link specifies the relationship between two business components. You must create a link that ties the recently modified Siebel business component to the new virtual business component. The procedure described below creates a link that ties the `Contact` business object to the `JCDA Account VBC` virtual business component created previously. Later, when bringing up the Siebel Call Center, selecting the `Contact` tab will activate the new `JCDA Account VBC` virtual business component.

To create the link:

1 From the Siebel Objects tree, select **Link**. The **Links** window opens:

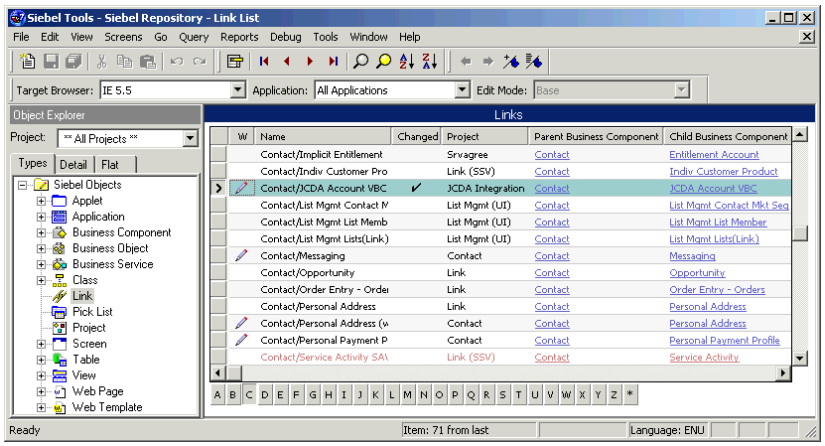


Figure 36. The Link window

2 Right-click on the **Links** window and select **New Record**.

3 Change the following values for the record that is created:

Column	Value
Project	JCDA Integration
Parent Business Component	Contact
Child Business Component	JCDA Account VBC
Source Field	Account Number
Destination Field	Account Number

Note: The link is automatically named as "Contact/JCDA Account VBC".

Modifying an Existing Business Object

You modify an existing business object to include the new virtual business component and the new link. In the procedure described below, the `Contact` business object is modified to include a new component, that is the new `JCDA AccountVBC` virtual business component with the new link.

To modify the business object:

1 In the Object Explorer, select **Business Object**. The **Business Objects** window is displayed.

2 In the **Business Objects** view, select the `Contact` business object (click on the C at the bottom of the list to show only Business Objects starting with C).

3 Back in the Object Explorer, expand the **Business Object** branch by clicking on the [+].

4 Select **Business Object Component**. The **Business Object Components** window is displayed under the **Business Objects** window:

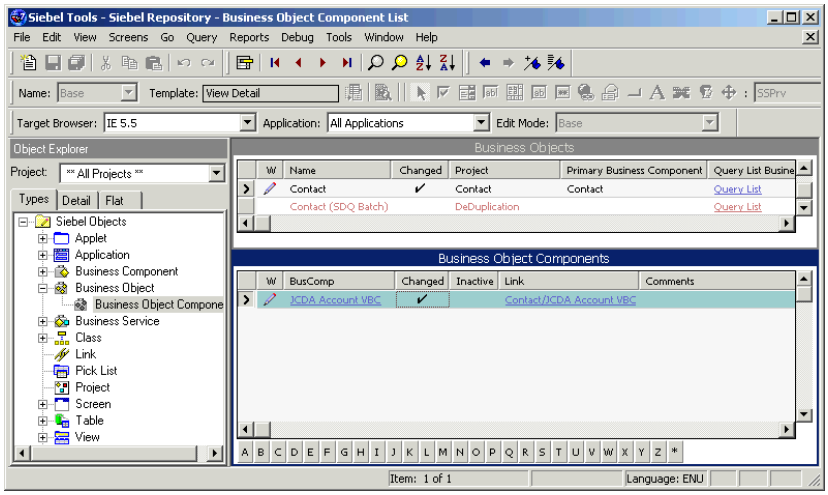


Figure 37. The Business Objects view

5 Right-click on the **Business Object Components** window and select **New Record** from the popup menu.

6 Modify the new business object component record as follows:

Column	Value
BusComp	JCDA Account VBC
Link	Contact/JCDA Account VBC

Creating a New List Applet

An applet allows access to the data of a single business component for viewing, editing, and modifying fields in the business component. You create a new list applet so that the Siebel client will be able to display the information retrieved from the legacy application. The new applet must display the data contained in the newly created virtual business component. The following procedure creates a new applet named "`JCDA Transaction Detail`", which is associated with the `JCDA Account VBC` virtual business component.

To create the new list applet:

1 In the Object Explorer, select **Applet**. The **Applets** window is displayed:

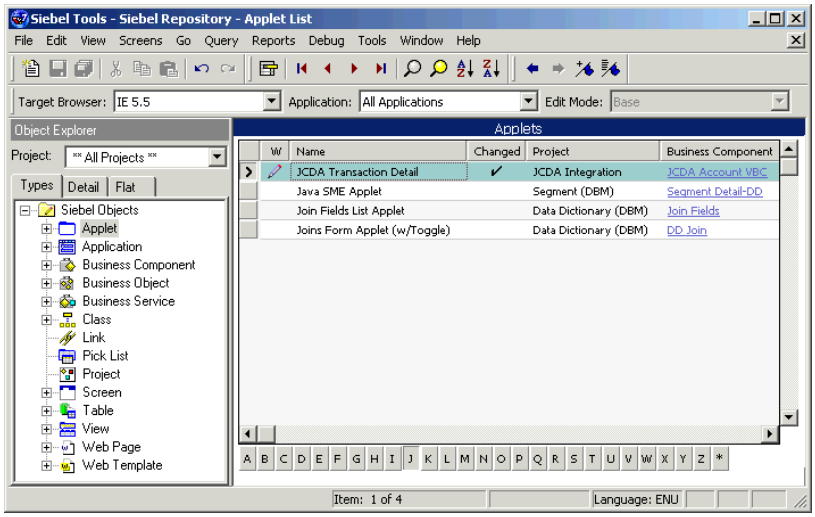


Figure 38. The Applets window

2 Right-click in the **Applets** window and select **New Record**.

3 Change the values for the following columns of the new applet record:

Column	Value
Name	JCDA Transaction Detail
Project	JCDA Integration
Business Component	JCDA Account VBC
No Delete	True
No Insert	True
No Merge	True
No Update	True
Title	Transactions

Adding Fields to the New List Applet

You must add fields to the new applet so that these will contain the different items of information retrieved from the legacy application. Each of these fields must correspond with a field in the virtual business component. The following procedure `JCDA Transaction Detail` list applet to identify the fields from the `JCDA Account VBC` virtual business component that will be displayed in the list applet.

To add fields to the new list applet:

1. In the Object Explorer, expand the **Applet** branch. Make sure that the `JCDA Transaction Detail` applet is still selected.
2. Select the **List** branch under the **Applet** Branch. The **Lists** window is displayed under the **Applets** window.
3. In the **Lists** window, add a new record by right clicking and selecting **New Record**. This creates a List record for this Applet.
4. In the Object Explorer, expand the **List** branch and select the **List Column** branch.

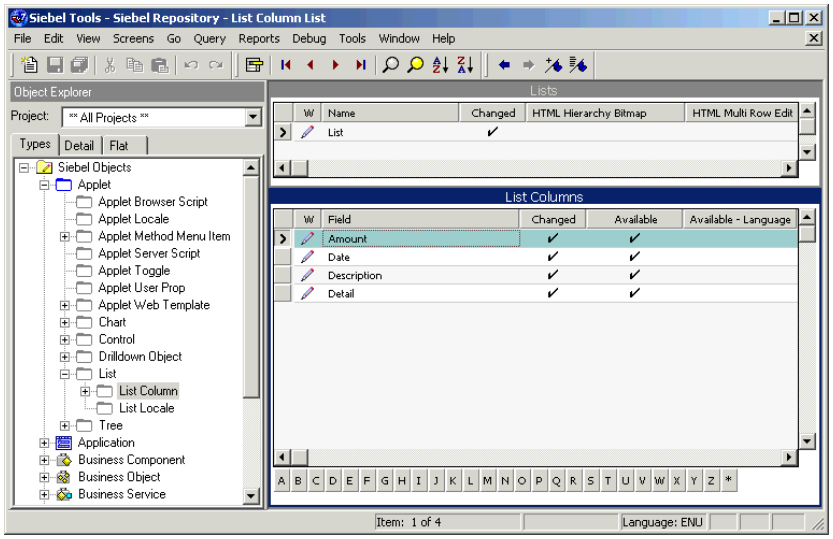


Figure 39. The List Columns view

5. Add the following records to the List Columns by right-clicking and selecting **New Record** within the List Columns window for each record:

Name

Detail

Description

Name

Date

Amount

Creating a Web Layout for the Applet

You create a web layout for the new applet. This defines how the new applet is physically displayed on a web browser connected to the Siebel client. The following procedure creates a web layout for the `JCDA Transaction Detail` applet using the Applet Wizard.

To create a web layout for the applet:

1. In the Object Explorer, select the **Applet** branch. Make sure that the `JCDA Transaction Detail` applet is still selected.
2. In the **Applets** window, right-click and select **Edit Web Layout** from the popup menu. A dialog box appears asking if you are willing to associate a web template using the Applet Wizard. Click **Yes** to associate a web template using the Applet Wizard.

   The Applet Wizard opens:

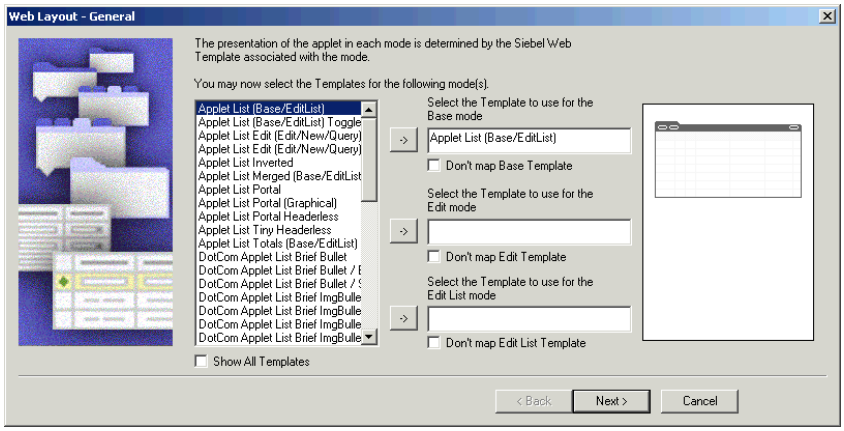


Figure 40. The Applet Wizard

3. In the dialog select **Applet List (Base/EditList)** and click the top right arrow button to set the value of the **Select the Template to use for Base mode** field.
4. Check both the **Don't map Edit Template** and **Don't map Edit List Template** check boxes.
5. Click **Next**.
6. Move the four available fields to the Selected Fields by selecting them and clicking on the right arrow button. Move the fields in the following order:

`Date`; `Description`; `Detail`; `Amount`. Then, click **Next**. This is seen in the following image:

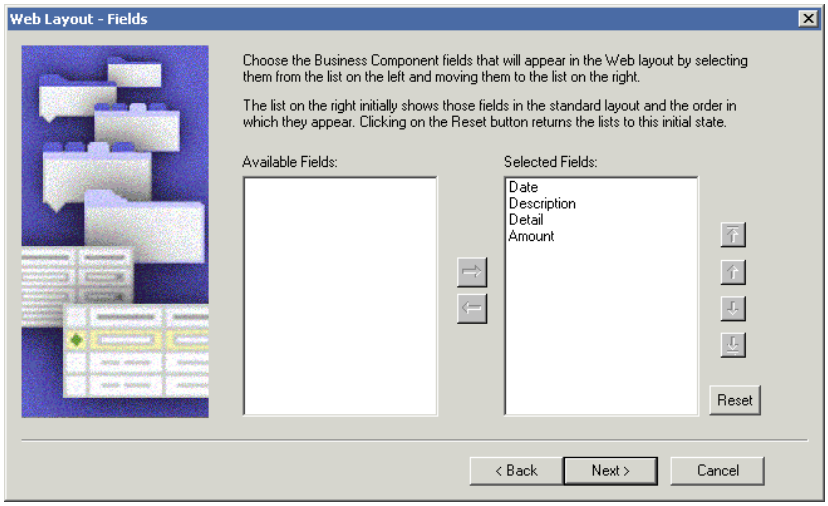


Figure 41. Choosing Business Component fields that will appear in web layout

7 Move all of the controls under **Selected Controls**, except for `GotoNextSet` and `GotoPreviousSet` to the **Available Controls** side by selecting each one to move and clicking on the left arrow button. This is seen in the following image:

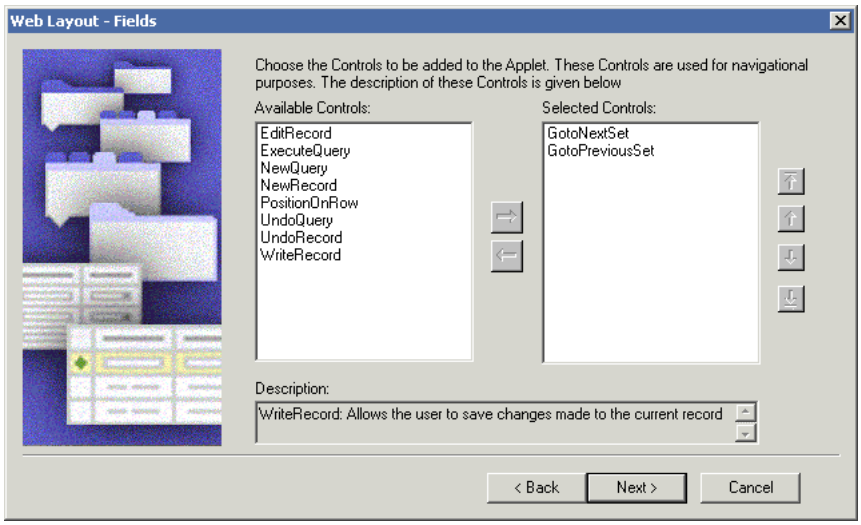


Figure 42. Choose controls to be added to the applet

8 Click **Next**.

9 Click **Finish**. HTML mappings are generated. The newly created applet window is displayed.

10 Close the newly created Applet view by clicking on the button under the button that would close the main Siebel Tools window.

The new applet is created.

Adding the New Applet to the View

A view presents one or more applets together at one time in a predefined visual arrangement and logical data relationship. The next step is to add the new applet to a view, so that it will be displayed on a web browser. The following procedure adds the `JCDA Transaction Detail` applet to the `Contact Profile View` view.

To add the new applet to the view:

1 In the Siebel Objects tree, select **View**. The **Views** window is displayed:

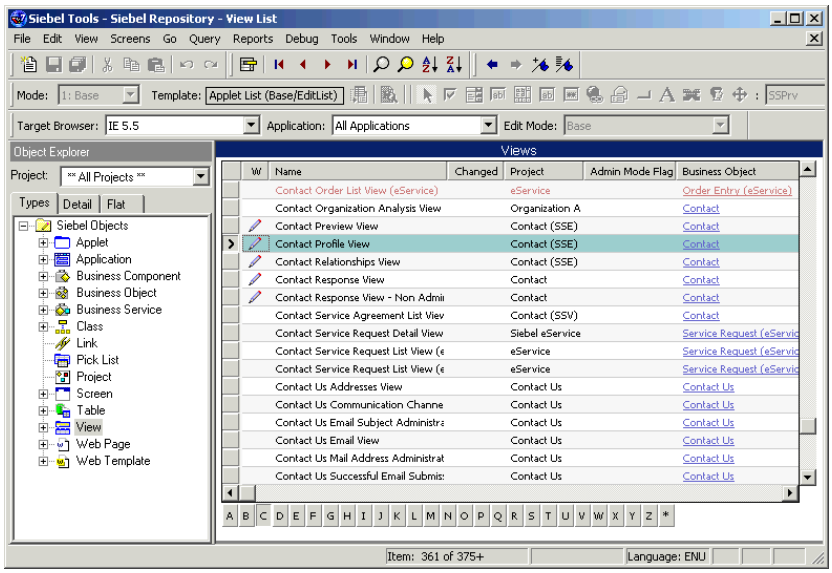


Figure 43. The Views window

2 In the **Views** window, select **Contact Profile View**. Right-click and select **Edit Web Layout** from the popup menu. The web layout is displayed on the right pane and the Applets list is displayed on the left under the Object Explorer.

3 In the Web Layout view, scroll down until you see the **Child or Grandchild Applet** sections. These sections should be highlighted in red.

4 From the **Applets** view box, drag the `JCDA Transactions Detail` icon applet into the first **Child or Grandchild Applet** section, as seen in the following image:

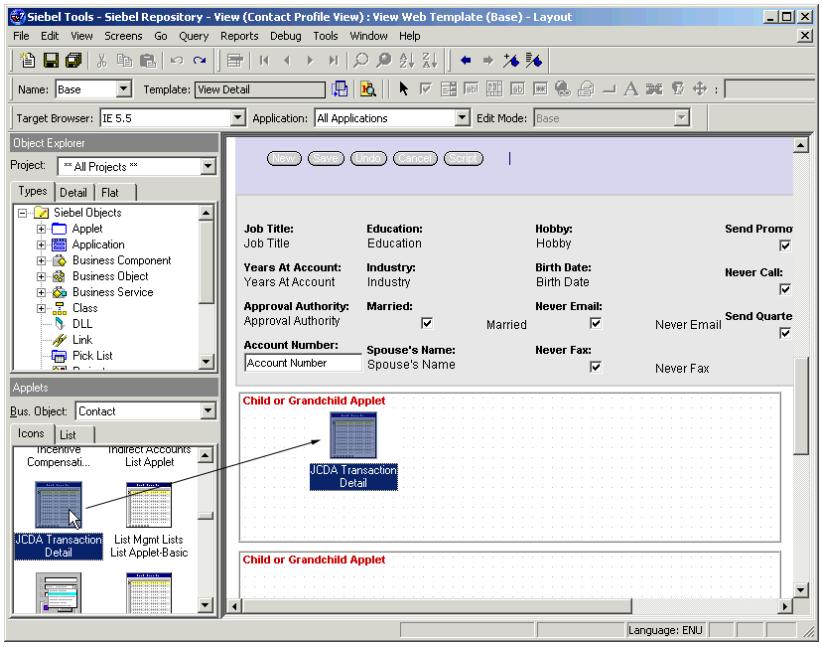


Figure 44. Drag the new applet into the Applet view

The **Child or Grandchild Applet** section should now look like this:

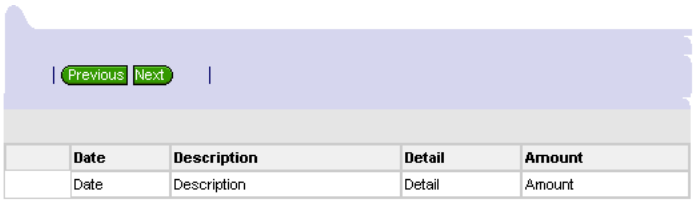


Figure 45. Resulting applet in web layout

5 Select **File > Save All**, and click **Yes** in the dialog box as seen below:

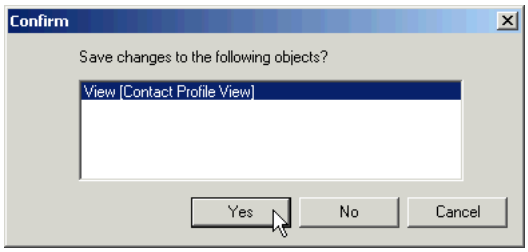


Figure 46. Save changes to Applet View

The new applet has been added to the view and the changes to the view are now saved.

Compiling the Project

After all work in Siebel Tools is done, you compile the project using the Siebel Object Compiler. The following procedure describes how to compile the Software AG Integration project.

To compile the project:

1 In Siebel Tools, select **Tools > Compile Projects**.

The **Object Compiler** dialog box opens:

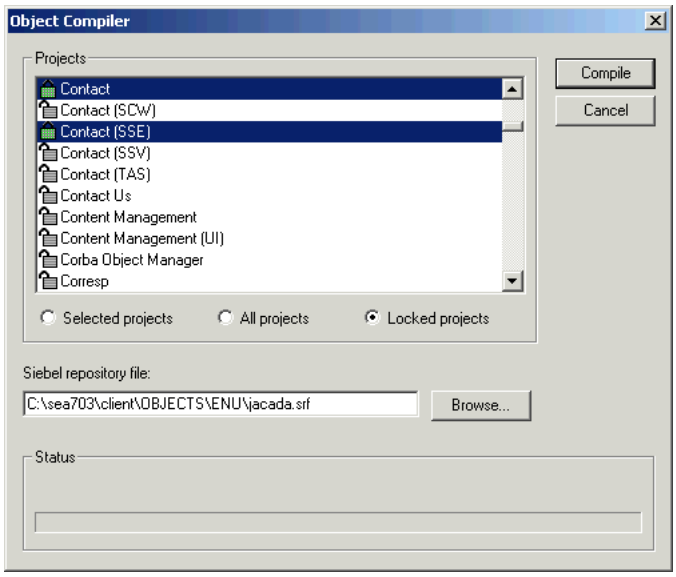


Figure 47. The Object Compiler dialog box

2 In the Object Compiler dialog box:

- In the **Siebel repository file** field, enter *C:\sea703\client\OBJECTS\ENU\softwareag.srf*.
- Select **Locked projects**.

Note: This procedure assumes that Siebel eBusiness Applications are installed into directory C:\sea703. If Siebel eBusiness Applications are installed into a different location, change these steps accordingly.

3 Click **Compile**.

The Software AG Integration project is compiled.

Testing the Software AG Integration with Siebel

After having concluded all of the necessary settings in Siebel Tools, you may test the Software AG Integration using the Siebel Call Center. Your Siebel Client should include a link to the webMethods JI environment. This means, that from within the Siebel client you will actually to access information from a legacy application, which was funneled through webMethods JI.

To activate the runtime:

1. Start the webMethods JI environment.
2. Start the Siebel Call Center.
3. If this is the first time you bring up the Siebel Call Center, log in as `sadmin` and use the sample data base.

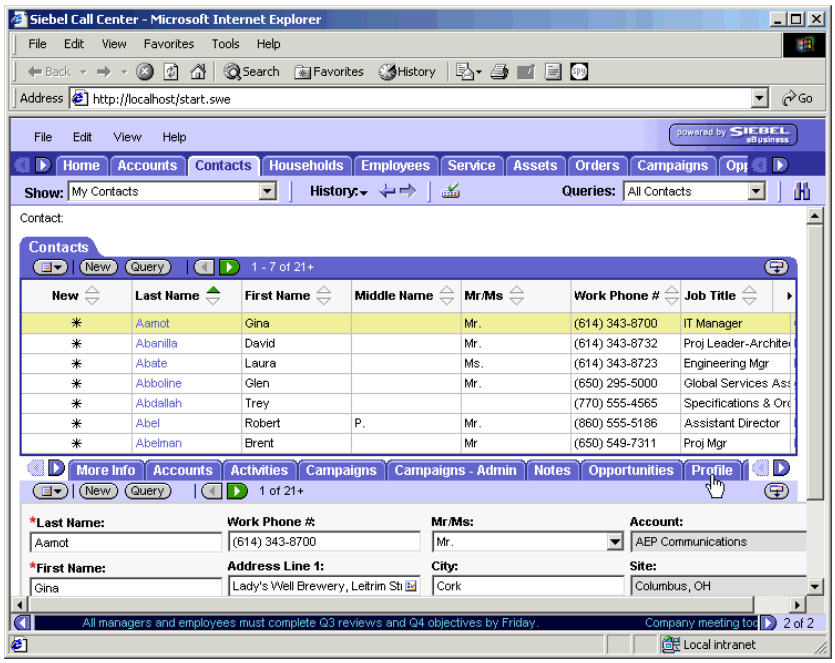


Figure 48. The Siebel Call Center

4. Select the **Contacts** tab. A second tab bar will appear under the **Contacts** tab.
5. Select the **Profile** tab from the second tab bar.

   This will display the Contact Profile View as created earlier in Siebel Tools.
6. Verify that the **Date**, **Description**, **Detail** and **Amount** fields are displayed with relevant information.

Files Used for Integration

QueryDemoRet.xsl

```
<?xml version="1.0" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
 <xsl:template match="/">
  <xsl:for-each select="mthGetCheckingDetails">
   <siebel-xmlext-query-ret>
    <xsl:for-each select="dtCheckingDetails">
     <row>
      <value field="Account Number">
       <xsl:value-of select="dfAccount"/>
      </value>
      <value field="Amount">
       <xsl:value-of select="dfCurrentBalance"/>
      </value>
      <value field="Detail">
       <xsl:value-of select="dfAvailableBalance"/>
      </value>
      <value field="Date">
       <xsl:value-of select="dfDate"/>
      </value>
      <value field="Description">
       <xsl:value-of select="dfType"/>
      </value>
     </row>
    </xsl:for-each>
   </siebel-xmlext-query-ret>
  </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

QueryDemoReq.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
```

```
 <xsl:template match="/">
  <xsl:for-each select="siebel-xmlext-query-req">
   <mthGetCheckingDetails>
    <xsl:for-each select="match">
     <AcctNum>
      <xsl:value-of select="."/>
     </AcctNum>
    </xsl:for-each>
   </mthGetCheckingDetails>
  </xsl:for-each>
 </xsl:template>

</xsl:stylesheet>
```

MajorBankVBC.cfg

```
xmlStart=<?
EA_ENVMGR=localhost:30001
EA_SERVICE=m_bankingXML

SiebelInitDefaultResponse=<siebel-xmlext-fields-ret>\
<support field="Account Number"/>\
<support field="Date"/>\
<support field="Description"/>\
<support field="Detail"/>\
<support field="Amount"/>\
<support field="Index"/>\
</siebel-xmlext-fields-ret>

SiebelQueryMethod=Checking_Trans
SiebelQueryInXSLURI=file:/C:/EAi/Siebel/QueryDemoReq.xsl
SiebelQueryOutXSLURI=file:/C:/EAi/Siebel/QueryDemoRet.xsl
```

Integration Using Integration Objects

Integration Objects are created using the Integration Object Builder in Siebel Tools. The Integration Object Builder provides several options for creating Integration Objects.

In the example provided in this document, an internal Integration Object will be created using the EAI Siebel Wizard option to represent the FINCORP Account Business Object. Also, an external Integration Object will be created using the EAI DTD Wizard to represent webMethods JI.

Creating a workflow that uses Integration Objects and the Data Mapper, data movement between Siebel and external systems can be easily created. By using the standard Siebel EAi Business Services, Integration Objects and the Data Mapper there is no eScript to be written.

Runtime Architecture

The figure below illustrates the dataflow for communicating between a Siebel eBusiness Application and webMethods JI using Integration Objects, the Data Mapper and XML services:

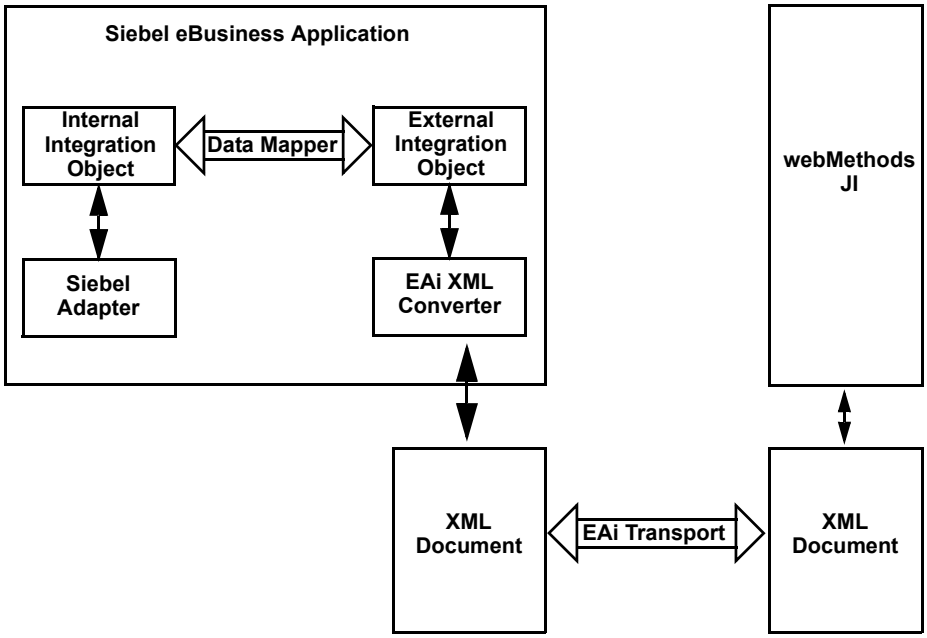


Figure 49. Runtime dataflow - integration using Integration Objects

Development in Siebel Tools

To integrate between a Siebel Client and a JI Service using Integration Objects:

1. Create an internal Integration Object to represent a business object.
2. Create an external Integration Object to represent the webMethods JI Service.
3. Compile the Integration Objects.
4. Map the data between the two Integration Objects in the Siebel web client.

5 Create a workflow.
6 Test the workflow.

Creating an Internal Integration Object

An internal Integration Object is needed in order to represent the Siebel web client.

To create an internal Integration Object, you must:

1 Start the Integration Object Builder in Siebel Tools.
2 Create the internal Integration Object in the Object Builder.
3 Activate fields.
4 Activate Component Keys.

Starting the Integration Object Builder in Siebel Tools

To start the Integration Object Builder:

Note: The first 3 steps of the following procedure are not necessary if Integration Objects are already enabled in the Object Explorer.

1 In Siebel Tools, select **View > Options…** The **Development Tools Options** dialog is displayed.
2 Select the **Object Explorer** tab, find **Integration Object** in the Available top level objects list and move it to the Visible top level objects list by clicking on the right arrow. This is seen in Figure 50.

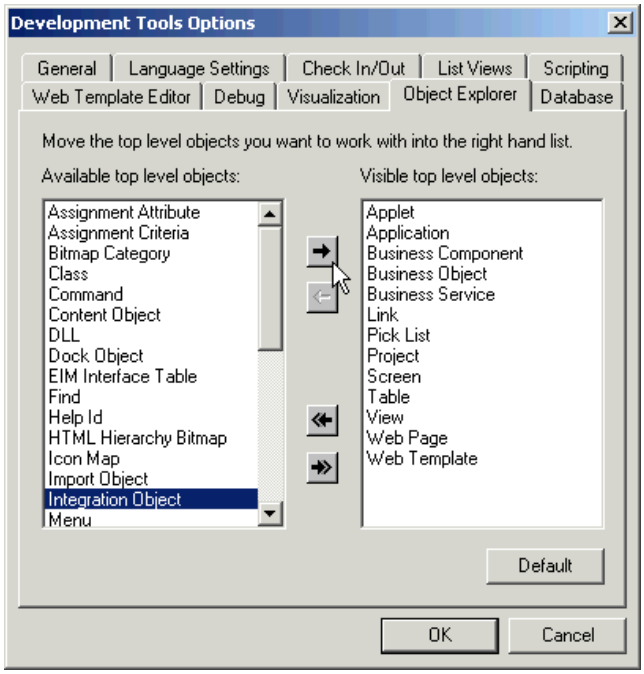


Figure 50. The Development Tools Options dialog box - enabling Integration Objects

3 Click **OK**.

4 From the **File** menu select **New Object…**. The **New Object Wizards** dialog is displayed.

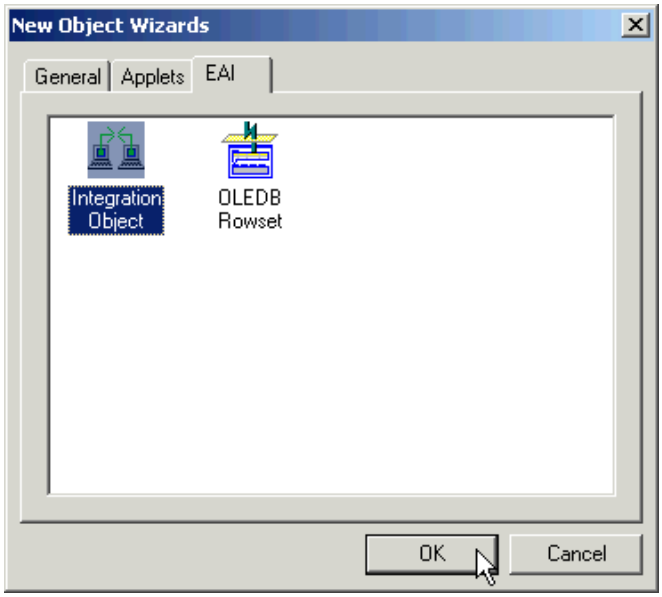


Figure 51. New Objects Wizard

5 Click on the **EAI** tab.

6 Select **Integration Object**.

7 Click **OK**. This will start the Integration Object Builder.

Creating an Integration Object in the Integration Object Builder

Once the Integration Object Builder is started, you can now create the internal Integration Object. The procedure provided below creates an Integration Object for the FINCORP account.

To create the Integration Object:

1 In the first step of the Integration Object Builder wizard, select **JCDA Integration** as the Project to use for this Integration Object, and select **EAI Siebel Wizard** as the source system of the new Integration Object.

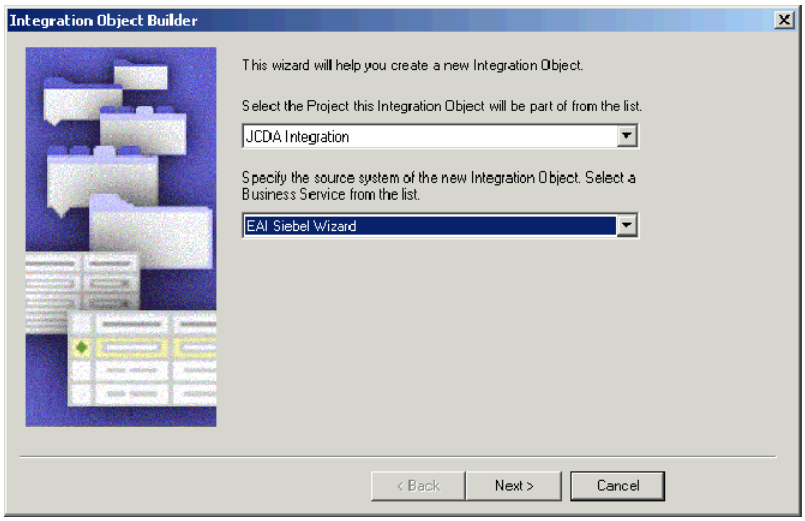


Figure 52. Integration Object Builder - first step

2 Click **Next**.

3 In the next step, select **FINCORP Account** as the source object for the new Integration Object, and select **JCDA FINCORP Account IO** as the name of the new Integration Object.

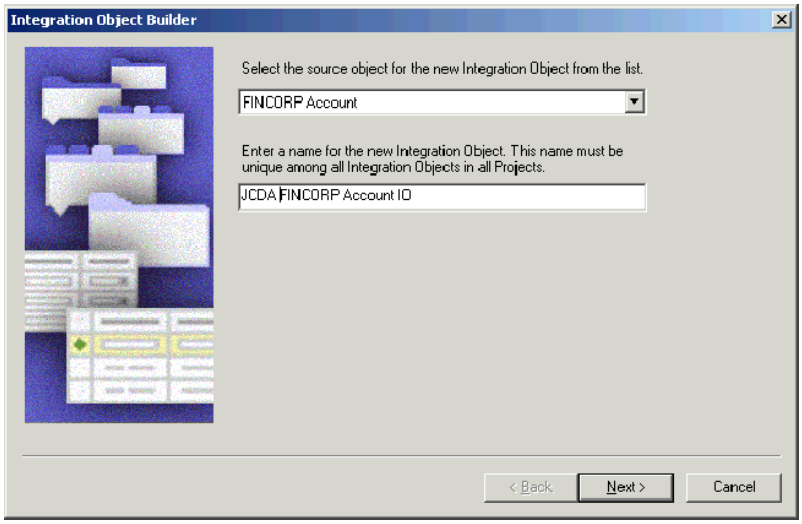


Figure 53. Integration Object Builder - second step

4 Click **Next**.

5 In the next step, ignore the messages and click **Next**.

6 In the next step, click on the [+] to expand the **FINCORP Account** branch.

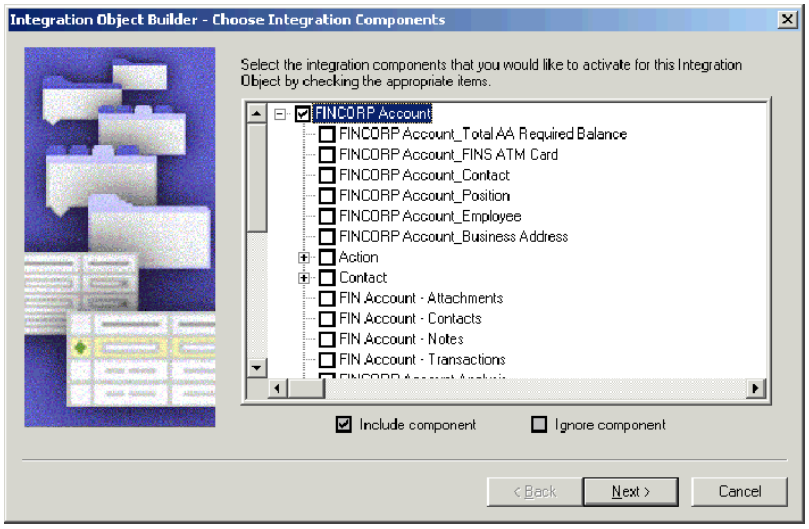


Figure 54. Integration Object Builder - Choose Integration Components

7 Click on the ☑ next to **FINCORP Account** to deselect all entries.

8 Click on the ☐ next to **FINCORP Account** to select only **FINCORP Account**. All other entries should not be checked at this point.

9 Click **Next**.

10 In the final step, ignore the messages and click **Finish**.

Activating Fields

To complete the setup for the **JCDA FINCORP Account IO** Integration Object, all unused fields must be set to **Inactive**. In this example, only the **Account Number** and **Account Alias** fields are used.

1. In the Object Explorer select the **Integration Object** branch.
2. Locate and select the **JCDA FINCORP Account IO** Integration Object. Click on the **J** at the bottom of the list to limit the list to only Integration Objects starting with J.
3. Expand the **Integration Object** branch by clicking on the [+].
4. Expand the **Integration Component** branch by clicking on the [+].
5. Select the **Integration Component Field** branch.
6. Double click on **Name** in the column header of the **Integration Objects Fields** list. This will lock this column so that when the list is scrolled horizontally the name of the fields will still be visible.
7. Scroll to the right until the **Inactive** column is next to the **Name** column.
8. On the first row, mark the field as **Inactive** by checking the **Inactive** attribute. To mark the row (field) **Inactive** click on the **Inactive** attribute. A check mark will appear.
9. To mark the remaining entries as **Inactive** use the down-arrow button to move to the next row then hit the space bar to set the **Inactive** field. Mark all fields (rows) as **Inactive**.
10. Scroll back to the top of the list.
11. Mark **Account Alias** and **Account Number** as Active by removing the check mark under the **Inactive** column. This is seen in Figure 55.

	W	Name	Inactive
>		AA Avg Balance	✓
		AA Avg Monthly	✓
		AA Fee Billed	✓
		AA Period Date	✓
		AA Statement Date	✓
		AA Surplus/Deficit	✓
		AA Total Charge	✓
		AA Total Required Balance	✓
		AA YTD Charge	✓
		ABA Number	✓
		Account Address Calc	✓
		Account Alias	
		Account Alias Number	✓
		Account Branch ABA	✓
		Account Branch Id	✓
		Account Branch Name	✓
		Account Branch Number	✓
		Account Category	✓
		Account Holder	✓
		Account Name	✓
		Account Number	
		Account Product	✓

Figure 55. Inactive fields

Activating Component Keys

Finally, the Component Keys that will not be used must be marked as Inactive. This example uses a Component Key containing only **Account Number**. The remaining five Component Keys will be marked as **Inactive**.

Follow these steps to set up the component keys:

1. Expand the Integration Component Key branch by clicking on the [+].
2. Click on the **Integration Component Key Field** branch. The component keys are displayed in the top list and the associated key fields are displayed in the bottom list.
3. Double-click on the **Name** column header in the Integration Component Keys list (top view). This will lock the **Name** column.
4. Scroll to the right until the **Inactive** column is next to the **Name** column.
5. Select each record in the **Integration Component Keys** list and note the key fields for that component key. Find the first component key that contains only **Account Number** as a key field.
6. Mark all other component keys as Inactive. Scroll down to make sure there are no additional component keys. The result is seen in Figure 56.

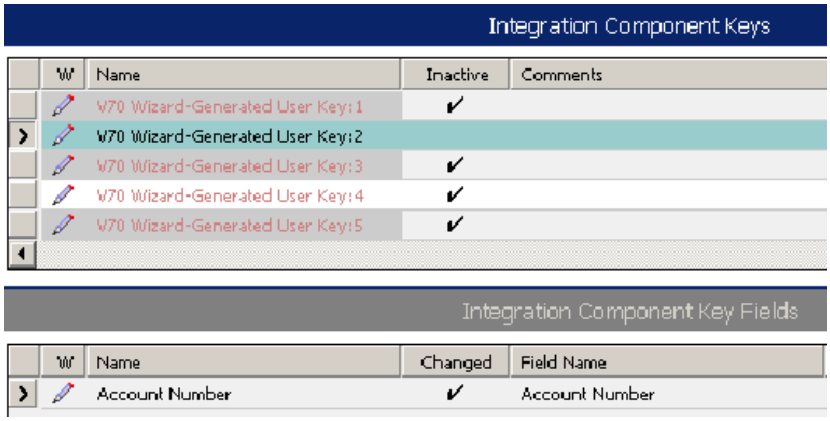


Figure 56. Inactive component keys

Creating the External Integration Object

For creating the external Integration object, you will need the DTD file created during JI Code Generation in MapMaker for the desired method in your JI Service. The following procedure utilizes the DTD file for the Get_Checking method of the M_Banking JI service:

7. Start the Integration Object Builder. See "Starting the Integration Object Builder in Siebel Tools" on page 158.
8. In the first wizard step, select **JCDA Integration** as the Project to use for this Integration Object, and select **EAI DTD Wizard** as the source system of the new Integration Object. A third pull-down list box appears.

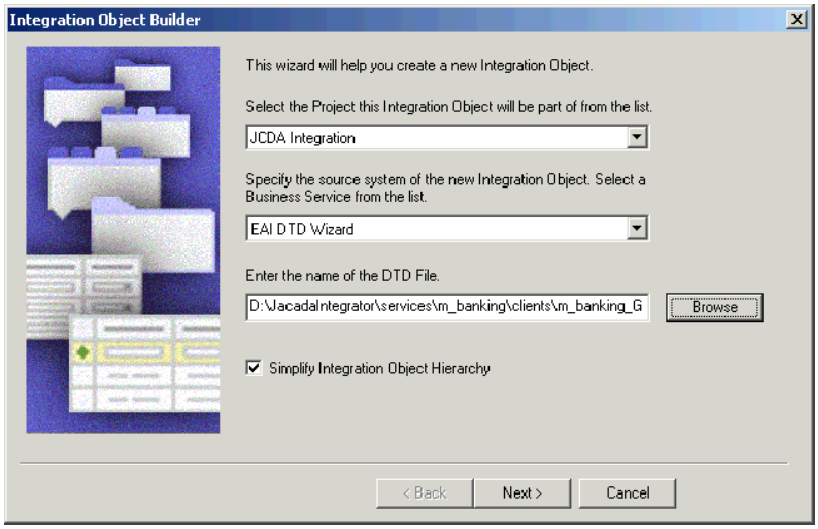


Figure 57. Integration Object Builder - first step

9 Locate the DTD for the **Get_Checking** method of the **M_Banking** JI service.

10 Click **Next**.

11 In the next step you select the root tag of the data structure represented by the DTD. Select **GetChecking** from the drop-down list.

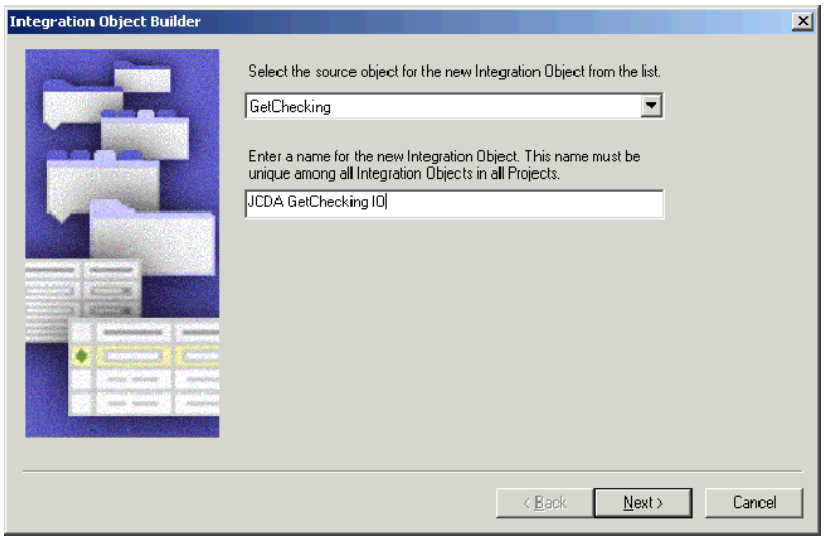


Figure 58. Integration Object Builder - second step

12 Enter **JCDA GetChecking IO** as the name of the new Integration Object.

13 Click **Next**.

14 In the next step you are asked to choose which elements from the DTD are to be used for integration. Make sure all elements are checked.

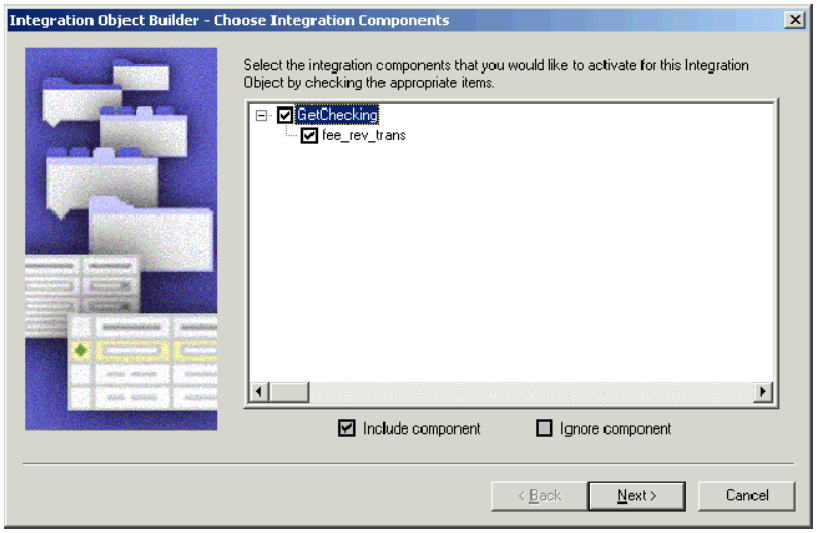


Figure 59. Integration Object Builder - Choose Integration Components

15 Click **Next**.

16 In the final step, ignore the messages and click **Finish**.

The external Integration Object is created.

Compiling the Integration Objects

The next stage is to compile the project(s) containing the Integration Objects in Siebel Tools.

To compile the Integration Objects:

1 In Siebel Tools, select **Tools > Compile Projects**. The **Object Compiler** dialog box opens.

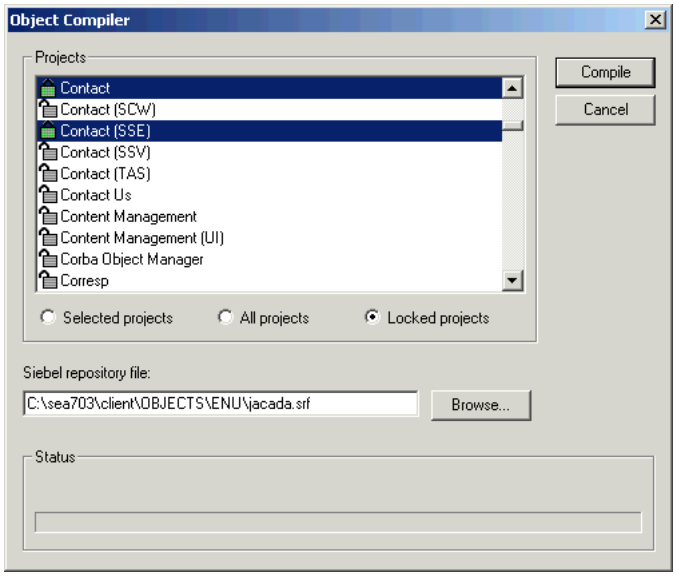


Figure 60. The Object Compiler dialog box

2 Enter the path to the Siebel Repository File in the **Siebel repository file** field. For example:

`C:\sea703\client\OBJECTS\ENU\softwareag.srf`

3 Select **Locked projects**.

4 Click **Compile**.

The Integration Objects are now compiled.

Mapping the Data

The next stage is to map the data from the internal Integration Object to the external Integration Object. In the example provided in this document, the **Siebel JCDA FINCORP ACCOUNT IO** Integration Object is mapped to the **JCDA GetChecking IO** Integration Object using Siebel Finance. The process is as follows:

1 Use the **View—Site Map** option to display all of the options available in Siebel Finance.

2 Click on the **Integration Administration Screen**. The **Integration Administration Options** list is displayed.

3 Click on **Data Maps**.

4 Create Data Maps to map between the **Siebel JCDA FINCORP Account IO** Integration Object and the **JCDA GetChecking IO** Integration Object. Create two new **Integration Object Map** entries by adding two new records with the following properties.

Integration Object Map

New | Query | Validate | 1 - 2 of 2 Query Results

Name	Source Object Name	Target Object Name	Comments
Jacada Request	JCDA FINCORP Account IO	JCDA GetChecking IO	Siebel to Jacada Integrator
Jacada Response	JCDA GetChecking IO	JCDA FINCORP Account IO	Jacada Integrator to Siebel

Figure 61. Integration Object Map - creating new records

5 Select **New Record** from the drop-down menu to add a new record. Click on the button to access the drop-down menu.

6 Edit the **Software AG Request** Data Map using the Data Map Editor. To access the Data Map Editor, select **Data Map Editor** from the **Show** drop-down list.

Show: Data Map Editor

7 Create a new record under **Integration Component Map** with the following properties:

Integration Component Map

Query | 1 - 1 of 1

Name	Source Component Name	Target Component Name
Jacada Request CM	FINCORP Account	GetChecking

Figure 62. Integration Component Map - creating a new record for the request

8 Create a new record under **Integration Field Map** with the following properties:

Integration Field Map

Query | 1 - 1 of 1

Source Expression	Target Field Name
[Account Number]	acctNo

Figure 63. Integration Field Map - creating a new record for the request

9 Select the **Software AG Response Data** map under the **Integration Object Map** applet at the top of the view.

10 Create a new record under **Integration Component Map** with the following properties:

Integration Component Map

Query | 1 - 1 of 1

Name	Source Component Name	Target Component Name
Jacada Response CM	GetChecking	FINCORP Account

Figure 64. Integration Component Map - creating a new records for the response

11 Create a new record under **Integration Field Map** with the following properties:

Integration Field Map

Query 1 - 2 of 2

Source Expression	Target Field Name
[acct_status]	Account Alias
[acct_num]	Account Number

Figure 65. Integration Field Map - creating new records for the response

The Integration Objects are now mapped.

Creating the Workflow

You must now create a workflow process. In the example provided in this document, a workflow is created in Siebel Finance.

1. Select **View > Site Map** to display all of the options available in Siebel Finance.
2. Click on **Siebel Workflow Administration**. A list of workflow administration options is displayed.
3. Click on **Workflow Processes**.
4. Create a new workflow process. This is done by adding a new record under **Workflow Processes** with the following properties:

Workflow Processes

Query Activate Revise

Name	Business Object
Jacada Integration Example	FINCORP Account

Figure 66. Workflow Process applet - creating a new workflow process record

5. Click on the **Process Designer** tab below the **Workflow Processes** applet.
6. Create the workflow using the **Workflow** palette components. This is seen in Figure 67.

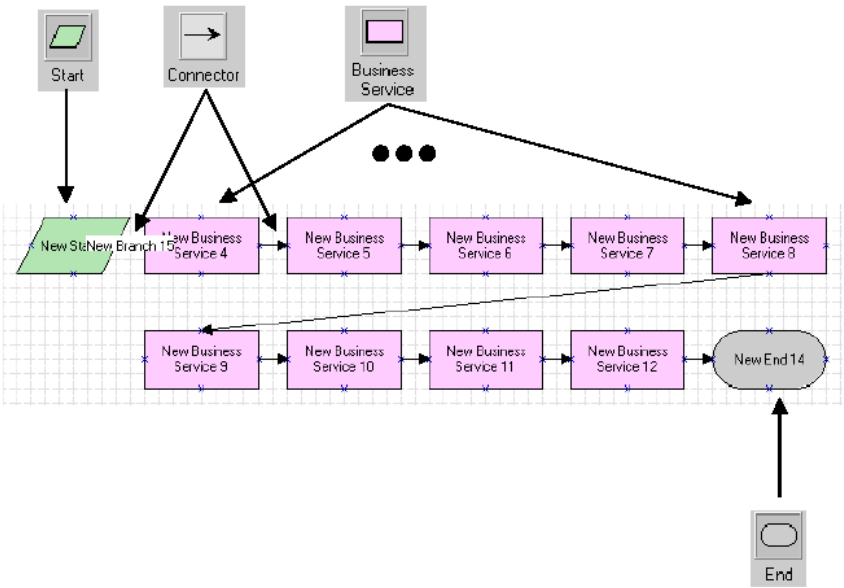
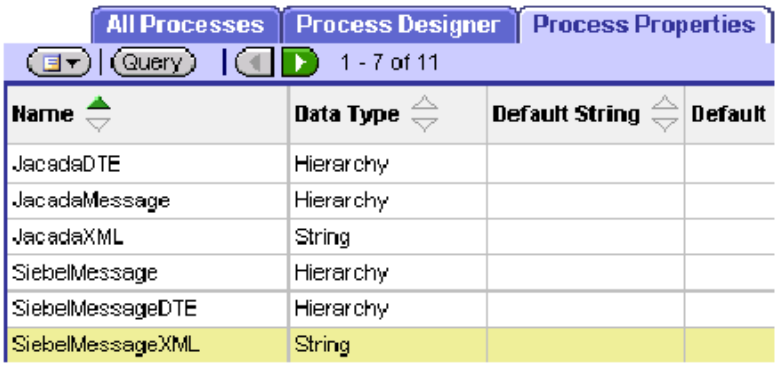


Figure 67. Creating the workflow

> **Note:** The step numbers will vary depending on your environment. The steps are renamed as each step's properties are set.

7 Define properties for the workflow. To do this, Click on the **Process Properties** tab.

8 Add the following properties:



Name	Data Type	Default String	Default
JacadaDTE	Hierarchy		
JacadaMessage	Hierarchy		
JacadaXML	String		
SiebelMessage	Hierarchy		
SiebelMessageDTE	Hierarchy		
SiebelMessageXML	String		

Figure 68. Process Properties tab - setting workflow process properties

9 Set the properties for each step in the workflow. To set the properties for a step, double-click on the desired step. This displays a view for setting the properties on the individual step. To return to the workflow designer click on the **Return to Designer** button available near the top of the view.

Testing the Workflow

To test the workflow:

1 Click on the **Process Simulator** tab and click on the **Start** button.

Figure 69. Process Simulator tab - Start button

The workflow is started.

2 After starting the workflow you can single-step through each step in the process by clicking on the **Next Step** button or run the entire workflow by clicking on the **Continue** button. If there is a problem with the workflow you will receive an error message.

When the workflow is completed, the account status from the **Major Bank** legacy application will have been written to the **Alias** field for the account number specified in the **Search Spec** of the **Get Siebel Data** step.

> **Note:** The **Alias** field was used for convenience, any existing field within **FINCORP Account** or any new field could have been used.

To view the results of the workflow follow these steps.

1 Select **View > Site Map**.
2 Select **Contacts**.
3 Select **All Contacts**.
4 Clicking on the **Query** button. This performs a query.
5 Enter **Agee** in the **Last Name** field and **Paul** in the **First Name** field.
6 Click on the **Go** button to execute the query. This should result in Paul Agee's record being selected.
7 Click on the **Consumer Services Summary** tab in the second row of tabs.
8 Select account **05852347800** by clicking on the account link 05852347800. The **Alias** field under Account Information should display **OPEN**.
9 To demonstrate the workflow again enter another value in the **Alias** field before running the Process Simulator again.

Files Used for Integration

MajorBankIO.cfg

```
environmentManager=localhost:30001
serviceName=MajorBankSvc
methodName=mthGetCheckingDetails
xmlStart=<?
logLevel=3
```

DTD for Get_Checking Method

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT GetChecking (acctNo?, checking_info?, fee_rev_trans*)>
<!ELEMENT checking_info (acct_num?, acct_status?, avail_bal?, avg_bal_12mo?,
avg_bal_1mo?, avg_bal_2mo?, cl_status?, closed_post?, curr_led_bal?,
date_open?, days_neg?, days_neg_year?, nfs_charge?, nfs_charge_year?,
nfs_return?, nfs_return_year?, ol_bal_today?, overdrawn_date?,
vip_status?)>
<!ELEMENT acctNo (#PCDATA)>
<!ELEMENT acct_num (#PCDATA)>
<!ELEMENT acct_status (#PCDATA)>
<!ELEMENT avail_bal (#PCDATA)>
<!ELEMENT avg_bal_12mo (#PCDATA)>
<!ELEMENT avg_bal_1mo (#PCDATA)>
<!ELEMENT avg_bal_2mo (#PCDATA)>
<!ELEMENT cl_status (#PCDATA)>
<!ELEMENT closed_post (#PCDATA)>
<!ELEMENT curr_led_bal (#PCDATA)>
<!ELEMENT date_open (#PCDATA)>
<!ELEMENT days_neg (#PCDATA)>
<!ELEMENT days_neg_year (#PCDATA)>
<!ELEMENT nfs_charge (#PCDATA)>
<!ELEMENT nfs_charge_year (#PCDATA)>
<!ELEMENT nfs_return (#PCDATA)>
<!ELEMENT nfs_return_year (#PCDATA)>
<!ELEMENT ol_bal_today (#PCDATA)>
<!ELEMENT overdrawn_date (#PCDATA)>
<!ELEMENT vip_status (#PCDATA)>
<!ELEMENT fee_rev_trans (amount?, description?, proc?, teller?, time?,
tran?)>
<!ELEMENT amount (#PCDATA)>
```

```
<!ELEMENT description (#PCDATA)>
<!ELEMENT proc (#PCDATA)>
<!ELEMENT teller (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT tran (#PCDATA)>
```

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 Key 173
Actions 173
AID Key 173
API 173
Applet 173
Application Programming Interface 173
Axis 95

B

Browser 173

C

C 96
CGI 173, 174
Character Encoding 174
Character Mode 174
Client 174
    functions 174
Client XML Support 63
Cookie 174

D

Data Field 174
Data Mapping 174
Data Stream 175
Data Templates 175
Data Typing 175
DLL 175
Document Object Model (DOM) 175
Document Type Definition (DTD) 175
Documentation 6
    Viewing on-line 8
DOM 175
DTD 175

E

EAServiceBean 175
ECS 175
Extensible Markup Language (XML) 181

F

Fields
    formatted 176
Formatted Fields 176
Formatting Conventions 6

G

GBBasic 176
Generating a WSDL in MapMaker 92
GUI 176

H

Host 176
HTML 176
HTTP 78
HTTP Gateway Servlet 9
    client options . . . . . 19 to 22
    installing 11
    message conventions 18
    message encoding 24
        response messages 26
    properties file . . . . . 15 to 18
    servlet initialization 14
    setting client options 18
    Tomcat 12

I

IBE 176
IBM 43
IDE 176

J

Java
runtime environment (JRE) 177
virtual machine (JVM) 177
Java Development Kit (JDK) 177
Java Runtime Environment (JRE) 177
Java Services 177
Java Virtual Machine (JVM) 177
JClient 177
JDBC 177
JDK 177
JIWSVC.war 95
JRE 177
JVM 177

L

Legacy Data 177
Legacy Host 177

M

Map 177
MapMaker 178
Methods 178
MQ Gateway 43
Error Messages 60
error messages 60
Shutting down the gateway 58
Starting the gateway 58
Stopping the gateway 58
MQSeries 43
messages 44
MQSeries Integration 113
MQSeries Queue Manager 44
Multicast 178
Multithreaded 178
Multithread-safe 178

O

Offset 178

P

Package 178
Presentation Space 178
Processing Siebel Requests 35
property file parameters 36, 37
usage 37
Protocol Agent 178
Proxy Server 178

R

request messages 24
Resource Database 179
Resource Manager 179
Resources 179
RMI 179

S

Screen 179
mapping 179
Service 179
Service Description 81
Service Discovery 87
Service Transport 78
Sessioning Support 39
Siebel Applets 128
Siebel Business Components 128

Siebel Tools 130
Siebel VBC
XML formats 29
delete 32
init 30
insert 34
pre-insert 33
query 30
update 32
XML interface 29
Siebel XML Support 29
error conditions 35
SOAP 78
SOAP Gateway
document-style messaging 95
RPC-style messaging 96
setup 95
SOCKS 179
Starting the MQ Gateway 58
System Monitor 179

T

Table Template 179
Tag 179
Telnet 180
Terminfo 180
The HTTP Gateway 9
Thread 180
Threaded 180
Thread-safe 180
TN3270 180
TN5250 180
Trail 180

U

UDDI 87
publishing a web service 105
Unicode 180

W

Web 180
Web Browser 180
Web Services 75
about 75
architecture 77
definition 75
developing 91
HTTP 78
publishing to UDDI 105
SOAP 78
testing 103
UDDI 87
WSDL 81
webMethods JI environment 177, 181
WSDL 81
generating in MapMaker 92

X

XBE 181
XML 181
XML Messaging 78
XML Spy 103
XML Support Overview 63