

Interactive Financial Services



# Customizable Web Server (CWS) Guide

**Note**

Before using this information and the product it supports, be sure to read the general information under Notices .

## **Draft for Beta - Fourth Edition (April 2002)**

This edition applies to the Version 1 Release 6 Modification 1 part of the Interactive Financial Services Solution.

Order publications through your IBM representative.

IBM welcomes your comments. A form for readers' comments is provided at the back of this publication.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 2002. All rights reserved.**

Note to U.S. Government Users: Documentation related to restricted rights: Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

---

# Contents

<b>Contents</b> .....	<b>iii</b>
<b>Figures</b> .....	<b>x</b>
<b>Trademarks and Service Marks</b> .....	<b>xi</b>
<b>Notices</b> .....	<b>xii</b>
<b>About This Book</b> .....	<b>xiii</b>
What This Book Contains .....	xiii
Terminology .....	xiii
Who Should Read This Book .....	xiii
What You May Find Useful Before Reading This Book .....	xiii
Where to Find More Information .....	xiv
Do You Have Comments, Problems, or Suggestions? .....	xiv
<b>Typographic Conventions</b> .....	<b>xv</b>
<b>Summary of Changes</b> .....	<b>xvi</b>
<b>Chapter 1. Customizable Web Server (CWS) Overview</b> .....	<b>1</b>
Components of the CWS Customization .....	1
Web Browser .....	1
The Customizable Web Server (CWS) .....	1
Transaction Delivery Manager .....	2
The Financial Institution .....	2
Understanding the CWS Environment .....	3
The Development Environment .....	3
<b>Chapter 2. Migrating to V1R6M1 and WebSphere 4</b> .....	<b>4</b>
Benefits of Using WebSphere 4.0 .....	4
Migrating from V1R6 with WAS 2 to V1R6 with WAS 4 .....	4
Software Prerequisites .....	5
CWS Distribution Format .....	5
Installation and Configuration of the CWS .....	6
Updates to CWS Customer Java Code .....	6
IfsSystem.properties File .....	6
Character Encoding Support .....	6
Migrating from V1R5 to V1R6 .....	6
NLS/DBCS .....	7
Code-to-Text Translations .....	7
Resource Bundle .....	8
User Profile Changes .....	9
Transaction API Changes .....	9
MRM/CWS Integration .....	9
NLS Enablement .....	11
Gold Manager Wide Character Support .....	11
Disable JavaScript .....	11
MQ Maximum Number Handles .....	11

<b>Chapter 3. Installing CWS Components .....</b>	<b>12</b>
Installation Overview .....	12
Install Software Prerequisites .....	12
AIX Performance Tuning .....	12
DB2 Setup .....	13
Creating a CWS Instance .....	15
Other Considerations .....	24
Running Multiple CWS Instances .....	24
Important Ports .....	25
Sample IHS Configuration File .....	25
Establishing A Secure HTTP Connection .....	25
WebSphere Hints and Tips .....	27
WebSphere Topography .....	27
CWS Parameter Setups and Recommendations .....	27
CWS Performance Recommendations .....	27
SNA Configuration .....	27
Steps for Customizing SNA .....	27
Update SNA files .....	28
MQ Configuration for a Test Environment .....	28
MQ Configuration for a Production Environment .....	29
 <b>Chapter 4. Writing Java Servlets for the IFS CWS .....</b>	 <b>32</b>
Before You Begin .....	32
Java Servlets .....	32
Servlet Advantages .....	33
The Servlet API .....	33
HttpServletRequest Object .....	34
HttpServletResponse Object .....	34
Service Method .....	34
Using Cookies .....	35
Session Management .....	36
JavaServer Pages (JSP) .....	37
Using IfsHttpServlet .....	37
Initialization .....	37
Creating and Reusing Gold Objects within Multiple Servlet Pages .....	38
Settings Fields within Gold Transaction Objects .....	39
Removing Objects from the Http Session Cache .....	41
Handling Errors following Transaction Execution .....	41
Controlling Page Navigation .....	42
Debugging Page Servlets .....	43
Sorting Gold Objects by Fields .....	43
Using IFS Gold Objects .....	44
Required Inputs for Security Classes .....	45
Preventing Multiple Executions of a Transaction Instance .....	46
Self-Defining Fields .....	46
Customization Methods .....	48
Debugging Transactions .....	51
Unit Testing Page Servlets .....	54
Reauthentication/Relogon .....	54
Using Resource Bundles for Gold Message Return Codes to Text Translation .....	55
Normal Transaction Behavior .....	57
Error Handling .....	57
Technical Details .....	57
National Language Enablement .....	58
Resource Bundle .....	58
Using Resource Bundles for Localization .....	58
Resource Bundle Names .....	58
Auto-Generation of CTT Properties Files .....	59

Code-to-Text Translations .....	59
Dynamic Session Locale .....	60
Using the NLS Utility Class - IfsNLS .....	61
Locating a Resource Bundle: IfsNLS.getBundle() vs ResourceBundle.getBundle() .....	61
Textual Representation of Gold Primitives in Localesensitive Manner .....	63
JDBC Considerations.....	66
JDBC Configuration.....	67
Host Database Table Changes .....	67
Procedures to Establish Connection to DB2 Database using JDBC .....	69
Considerations in Using Multiple User Ids .....	71
Financial Impacts.....	71
GtAddSecondaryConsumer .....	71
GtChangeSecondaryConsumer .....	72
GtCloseSecondaryConsumer.....	73
GtResetSecondaryConsumer .....	73
GtSecondaryConsumerInquiry .....	73
Potential Issues and Problems .....	73
Instance Variables of a Servlet.....	73
Duplicate Transaction Problem .....	74
Using Object Cache.....	75
<b>Chapter 5. Troubleshooting Java in the CWS.....</b>	<b>76</b>
FI-Developed Servlets .....	76
Java CWS API Classes .....	76
Traditional CWS Debugging .....	77
CWS Trace Files .....	77
Gold Trace Files .....	77
CWS Message Log.....	77
Core Files .....	77
Java Specific Debugging Capabilities.....	77
IfsSystem.properties and ifs.system.debug .....	77
javacore.txt file.....	78
Specific Error Messages .....	78
78 - Timeout Has Occurred .....	78
404 - File Not Found.....	78
801 - Unarchitected Error Code .....	78
<b>Chapter 6. CWS Developer's ToolKit .....</b>	<b>79</b>
Introduction .....	79
Test Data Flow .....	80
Using the Basic Tool .....	80
Prerequisites.....	81
WebSphere Installation .....	81
Installing the CWS Toolkit .....	81
CWS Configuration File.....	83
Configure the CWS Toolkit Server Programs .....	83
Running the Toolkit.....	84
Using the Java Servlet Examples.....	84
Enabling Server Encryption.....	84
Using the Enhanced Simulation Tool.....	85
DB2 Configuration .....	85
CWS Configuration File for Enhanced Tool .....	87
Running the Enhanced Toolkit .....	87
Basic Data Management .....	87
Data Management using Approach or Excel.....	88
Development and Test Environment .....	88
The Test Environment.....	88

CWS Trace and Log Facilities.....	89
CWS Message Log.....	89
CWS Internal Execution Trace.....	89
Gold Messages Trace.....	90
Code to Text Problems.....	93
Logging for FISim.....	94
Debug Process.....	94
Debugging.....	94
Debug Functions.....	95
Debugging Tips.....	95
Support Pages.....	96
Gold Transactions Serviced By Enhanced Simulation.....	96
Predefined Responses.....	99
Important Points To Consider.....	100
Environment variables.....	100
User Configuration Files.....	100
Response Data Files.....	101
Repeat Items.....	102
Goldmine.....	102
<b>Appendix A. Web Server Program (WSP) Error Messages and Codes.....</b>	<b>104</b>
WSP Component Return Codes.....	104
WSP PFM Return Codes.....	104
Reason Codes Used in the Gold Header for the PFM Server.....	104
Web Server Return Codes.....	105
CWSAPI Request Codes.....	106
Gold Manager Codes.....	106
Codes Processing Java Request.....	106
Internal Errors in Gold Manager - Locating Atoms.....	107
Gold Manager Error Codes While Creating the Output Response Buffer.....	107
Internal Conversion Errors (GMNIBBLE).....	107
Gold Manager Initialization Codes.....	107
Errors While Logging Off.....	108
Extra Error Codes.....	108
<b>Appendix B. IFS Web Page Performance/Response Time Tuning Recommendations.....</b>	<b>109</b>
Introduction.....	109
Environment.....	109
Service Provider.....	109
Hosted Applications.....	109
IFS Response Times.....	109
Response Time Considerations.....	109
Other Response Time Factors.....	110
Networks.....	110
What Does All This Mean?.....	110
Backend Latency.....	110
Page Design.....	110
Graphics.....	110
Multiple Transactions on a Page.....	112
Large Messages.....	113
Web Caching.....	113
Stay on Budget.....	113
Client Side JavaScript.....	113
Testing Pages.....	114
Page Implementation.....	114
Gold Manager Caching.....	114
Web Server Caching.....	115

<b>Appendix C. Gold Message Interface Tool (GMIT)</b>	<b>116</b>
Introduction	116
The Test Data Flow	116
Glossary	117
Getting Started	117
Prerequisites	117
Installing GMIT	118
MQSeries and GMIT	118
Adding a Queue Manager	118
Starting a Queue Manager	119
Adding Default Objects	119
Defining Queues	120
Adding Local Queues	120
Adding the Remote Queue	121
Adding Alias Queues	121
Adding Channel Definitions	122
Adding Process Definitions	123
Starting a Channel Initiator	123
Displaying Channel Status	123
goldmine	123
Modifying Request Data Files	123
Running goldmine	126
goldbank	127
Modifying the goldbank Configuration File	127
Running goldbank	128
Request Data Files	129
Messages and Codes	135
goldbank Messages	135
goldmine Messages	136
MTI Codes	137
 <b>Appendix D. FISim (Financial Institution Simulator)</b>	 <b>138</b>
Introduction	138
The Test Data Flow	139
Installing FISim	140
Prerequisites	140
Installation	141
MQSeries and FISim	142
All Local MQSeries Configuration	142
Remote MQ Series Configuration for FISim	146
Remote MQ Series Configuration for CWS Web Server	151
Using FISim	156
DB2 Configuration	156
Running FISim	158
Basic Data Management	159
Data Management using Approach or Excel	159
Altering Response Codes	160
Command Descriptions	160
Examples	161
Entering Commands Directly Through DB2	161
Troubleshooting	162
FISim Trace Files and Logs	162
FISim Returns a BRC of 247 Due to SQLCODE of -954	162
Predefined Responses	163
Important Points to Consider	163
Environment Variables	164
Response Data Files	165
Repeat Items	165

Goldmine .....	166
<b>Appendix E. IFS PvC Starter Kit .....</b>	<b>167</b>
Introduction .....	167
Solution Architecture .....	167
Functions Supported .....	168
System Requirements .....	169
IFS PvC Support Installation.....	169
Configuration on CWS for IFS PvC Support .....	169
Solution Design .....	170
User Interface Design.....	170
IFS PvC Servlets Design .....	170
Data Security Consideration.....	172
NLS Enabling.....	172
Class Definition .....	172
Common Services .....	172
User Authorization .....	175
Account Inquiries.....	176
Fund Transfers .....	178
Customer Services .....	182
IFS PvC Servlets Normal Page Flow .....	183
IFS User Interface on Handheld Device .....	185
Logon Screen .....	185
Main Function Screen .....	186
Fund Transfer Function Screen.....	188
Services Function Screen .....	191
Logoff Screen .....	192
<b>Appendix F. Basic WebSphere 4 Administration .....</b>	<b>194</b>
WebSphere 4.0.2 Overview .....	194
Starting WebSphere Administration Server .....	194
Launching the Graphical Administrative Client.....	194
Windows Instructions.....	195
Stopping WebSphere.....	195
Starting an Application Server .....	195
Stopping an Application Server .....	196
The wscp.sh Command Line Utility .....	196
AIX.....	197
<b>Appendix G. Migrating from Java 1.1.8 to Java 1.3.0.....</b>	<b>198</b>
Java API Changes .....	198
WebSphere 4.0.2 Supported APIs .....	198
Changes in the JCWSAPI .....	198
Common Migration Problems and Resolutions .....	203
The Invoker Servlet .....	203
Cannot Forward After Obtaining Stream .....	204
Effective Call To sendRedirect() Method .....	206
JSP .91 Syntax not supported.....	207
Unsupported API Calls .....	210
Recompilation and Deprecated Methods .....	211
Miscellaneous Problems .....	211
<b>Appendix H. Building And Deploying CWS Web Applications In WebSphere 4.....</b>	<b>213</b>
J2EE Overview .....	213
J2EE Packaging and Deployment Concepts and Terms .....	213
Structure Of The ifs.war File .....	215



<doc_root> .....	215
<doc_root>/META-INF .....	215
<doc_root>/WEB-INF .....	216
Extending The Base ifs.war File .....	218
Uninstall/Reinstall Approach .....	220
Copy/Replace Approach .....	221
<b>Glossary of Terms and Abbreviations .....</b>	<b>222</b>
<b>Bibliography .....</b>	<b>230</b>
Interactive Financial Services Library .....	230
Other Documents .....	230
CTW HELP .....	230
Other Publications .....	230
<b>Reader's Comments – We'd Like to Hear from You .....</b>	<b>231</b>

---

## Figures

Figure 1. Basic Areas of Customization.....	1
Figure 4. Java Servlet and JSP programming in IFS CWS .....	32
Figure 5. CWS Toolkit Environment .....	79
Figure 6. CWS Toolkit Data Flow.....	80
Figure 7. CWS Toolkit Directory Structure.....	83
Figure 8. GMIT Program Flow .....	117
Figure 9. Logoff.dat.....	126
Figure 10. Logoff.out.....	127
Figure 11. goldbank.config.....	128
Figure 12. logoff.fgresp .....	129
Figure 13. FISim Configurations without a Core Controller .....	138
Figure 14. FISim Configurations with a Core Controller .....	139
Figure 15. FISim Data Flow with a Core Controller .....	140
Figure 16. FISim Directory Structure .....	141
Figure 17. IFS Architecture for PvC Support .....	168
Figure 18. Flow chart of IFS PvC servlet.....	171
Figure 19. The Inheritance Hierarchy of IFS PvC Servlets.....	172
Figure 20. Logon Screen .....	186
Figure 21. Main Function Screen.....	186
Figure 22. Accounts Function Screen.....	186
Figure 23. Account Summary Screen .....	187
Figure 24. Account Details Screen .....	187
Figure 25. Transaction Search Screen .....	187
Figure 26. Transaction History Screen .....	188
Figure 27. Balance Inquiry Screen .....	188
Figure 28. Interest Rates Screen.....	188
Figure 29. Fund Transfer Screen.....	189
Figure 30. Add Fixed Transfer Screen.....	189
Figure 31. Add Variable Transfer Screen .....	189
Figure 32. Search Transfer Screen .....	190
Figure 33. Change Fixed Transfer Screen .....	190
Figure 34. Change Variable Transfer Screen .....	190
Figure 35. Delete Fixed Transfer Screen.....	191
Figure 36. Delete Variable Transfer Screen .....	191
Figure 37. Transfer Inquiry Screen .....	191
Figure 38. Service Function Screen .....	191
Figure 39. Check Copy Order Screen .....	192
Figure 40. Check Book Reorder Screen.....	192
Figure 41. Stop Cheque Payment Screen .....	192
Figure 42. Logoff Screen .....	193

---

## Trademarks and Service Marks

The following terms used in this book are trademarks or service marks of the companies indicated, in the United States or other countries or both:

AIX	IBM Corporation
CICS	IBM Corporation
CICS/ESA	IBM Corporation
DB2	IBM Corporation
FlowMark	IBM Corporation
IBM	IBM Corporation
IBM Global Network	IBM Corporation
IMS	IBM Corporation
Integrion	Integrion
Language Environment	IBM Corporation
Lotus Notes	IBM Corporation
NetView	IBM Corporation
Money	Microsoft Corporation
MQ	IBM Corporation
MQSeries	IBM Corporation
MVS/ESA	IBM Corporation
MVS	IBM Corporation
OS/2	IBM Corporation
Quicken	Intuit Inc.
RACF	IBM Corporation
SOMobjects	IBM Corporation
SP	IBM Corporation
SP2	IBM Corporation
System/390	IBM Corporation
VTAM	IBM Corporation
WebSphere	IBM Corporation

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation. Java is a trademark of Sun Microsystems, Inc. Lotus Notes is a registered trademark of Lotus Development Corporation.

Other company, product, and service names may be trademarks or service marks of other companies.

---

## Notices

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785 U.S.A

Those interested in enabling programs to exchange information with IFS using the Gold Standard Message Specification should contact:

IFS Publications Coordinator  
IBM Manassas  
10511 Battleview Parkway  
Manassas, VA 20109  
United States of America

---

# About This Book

---

## What This Book Contains

This book contains information about the Interactive Financial Services (IFS) Customizable Web Server (CWS). It describes how a financial institution can:

- Customize IFS through the Customizable Web Server (CWS)

## Terminology

This book refers to:

- Consumer financial institutions (CFIs) as *financial institutions (FIs)*
- The following terms are used interchangeably: *customer, end user, subscriber, and consumer*. These terms mean the same thing; namely, the customer of a financial institution and the user of the financial institution's electronic solution (IFS).
- Gold Standard for Electronic Financial Services as *standard message*

---

## Who Should Read This Book

This book is for financial institution personnel, application programmers, system programmers, and anyone else who plans to customize IFS and integrate it with a financial institution's existing systems.

## What You May Find Useful Before Reading This Book

Before you read this book, it may be helpful for you to have some knowledge of or experience with some of the following hardware and software products and services, depending on which IFS components you'll be working with:

- End-user interface:
  - Windows NT operating system
  - Netscape Communicator
  - Microsoft Internet Explorer
- Customizable Web server:
  - HTML
  - IBM WebSphere Application Server
  - Sun Microsystems, Java Programming Language (Java)
  - Java Servlets
  - Java Server Pages (JSPs)
- Standard Messages
- IFS Interface Module:
  - System/390 (S/390) CMOS processors
  - The Multiple Virtual Storage (MVS) operating system
  - The Customer Information Control System (CICS) program
  - The IBM (DB2) Universal Database program
  - The Message Queuing Series (MQSeries) program
  - The Application Integration Feature (AIF) of the Flowmark for MVS/Enterprise Systems Architecture (MVS/ESA) program, including:

AIF mapping

The messaging application programming interface (MAPI)

- RISC System/6000 (SP2) processors

- The Advanced Interactive eXecutive (AIX) operating system

## **Where to Find More Information**

Where necessary, this book refers to information in other publications. Additional information is also available through the Internet.

## **Do You Have Comments, Problems, or Suggestions?**

Your suggestions and ideas can contribute to the quality and the usability of this publication. If you have suggestions for improving this publication or if you have any problems using it, please send us your comments by mail. See *Communicating Your Comments* located at the end of the book.

---

# Typographic Conventions

The following typographic conventions are used in this library:

Convention	What It Means
<b>Bold</b>	<b>Bold</b> words and characters represent system elements (commands, for example) that you use literally.
<i>Italic</i>	<i>Italic</i> words and characters represent variable values that you specify. <i>Italics</i> are also used as follows: <ul style="list-style-type: none"><li>• For book titles</li><li>• For textual emphasis</li><li>• When a glossary term is first introduced in the text.</li></ul>
<i>Constant width</i>	<i>Constant width</i> words and characters represent examples of information that the system displays.
[ ]	Brackets enclose optional items in command format and syntax descriptions.
{ }	Braces enclose a list of items you choose from in command format and syntax descriptions.
	Vertical bars represent the word <code>or</code> . They separate items in a list of choices. Vertical bars can also represent a pipe in Unix.
...	Ellipses indicate that you can specify the preceding item one or more times.
< >	Angle brackets ( <code>less than</code> and <code>greater than</code> ) enclose the name of a key on the keyboard.
<Ctrl-x>	<Ctrl-x> represents a <i>control-character sequence</i> . For example, <Ctrl-c> means: hold down the control key <Ctrl> while pressing <c>. <b>Note:</b> On some AIX platforms <Ctrl-x> is an actual signal (interrupt).
<Return>	<Return> represents the key on your workstation keyboard labeled with the word <b>Return</b> or <b>Enter</b> , a left arrow, or both (word and arrow).

---

# Summary of Changes

## Summary of Changes for Interactive Financial Services Customizable Web Server Guide Version 1 Release 6 Fourth Edition

The Fourth Edition of the document reflects the change from WebSphere 2.03 to WebSphere 4.0.x as the environment supporting CWS for V1R6M1.

### New Information

- Chapter 2, Migrating to V1R6M1 and WebSphere 4
- Appendix F, Basic WebSphere 4 Administration
- Appendix G, Migrating from Java 1.1.8 to Java 1.3.0
- Appendix H, Updating the IFS WAR File

### Changed Information

- Extensive information on administering and configuring CWS for WebSphere Application Server 4.0 have been added including sections on WebSphere administration, deploying CWS on WebSphere, and migrating Java code.
- All references to the Netscape IPlanet Server product have been removed as this server product is no longer supported.
- Customizable Web Server (CWS) Overview
- Extensive information on migrating to v1R6M1

### Moved Information

- Chapter 2, Customizable Web Server (CWS) Overview moved to Chapter 1

## Summary of Changes for Interactive Financial Services Customizable Web Server Guide Version 1 Release 6 Third Edition

This was an internal edition only. These changes are included in the fourth edition.

### Changed Information

- Chapter 2, Customizable Web Server (CWS) Overview
- Chapter 4, Writing Java Servlets for the IFS CWS
- Appendix B. IFS Web Page Performance/Response Time Tuning Recommendations

## Summary of Changes for Interactive Financial Services Customizable Web Server Guide Version 1 Release 6 Second Edition

### Changed Information

- Chapter 3, Installing CWS Components
- Chapter 4, Writing Java Servlets for the IFS CWS
- Appendix E, IBM IFS PvC Starter Kit
- Bibliography

### Deleted Information

- Index



## Summary of Changes for Interactive Financial Services Customizable Web Server Guide Version 1 Release 6 First Edition

This First Edition supports Version 1 Release 6 Modification 1 of the Interactive Financial Services solution. Version 1 Release 6 technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

In addition to the changes indicated below, this book has been reorganized and has a new title (IFS Customizable Web Server (CWS) Guide). Maintenance, terminology and editorial changes have also been included.

### New Information

- Chapter 1, CWS Migration Considerations from V1R5M1 to V1R6M1
- Chapter 3, Installing CWS Components
- Appendix D, FISim (Financial Institution Simulator)
- Appendix E, IBM IFS PvC Support

### Changed Information

- iPlanet Web Server, Enterprise Edition 4.0SP4 has replaced the Netscape Enterprise Server
- Chapter 2, Customizable Web Server (CWS) Overview
- Chapter 4, Writing Java Servlets for the IFS CWS
- Chapter 5, Troubleshooting Java in the CWS
- Chapter 6, CWS Developer's Toolkit
- Appendix A, Web Server Program (WSP) Error Messages and Codes
- Appendix B, IFS Web Page Performance/Response Time Tuning Recommendations
- Appendix C, Gold Message Interface Tool (GMIT)

### Moved Information

- Chapter 1, Introduction to IFS. This information can be found in the *IFS Overview and Release Changes Guide*.
- Chapter 6, Customizing the Messaging Server. This information will be available in a stand-alone IFS Messaging Server Guide.
- Appendix A, The Interactive Financial Services Infrastructure: Software. This information can be found in the *IFS Planning Guide*.
- Appendix B, IFS Gold Message Descriptions and Usage. This information can be found in the IFS Gold Standard Message Transaction Specification.
- Appendix E, IFS Business Response Codes. This information can be found in the IFS Gold Standard Message Transaction Specification.
- Appendix H, IFS Service Provider Adapter 2.1.A Guide for 98.2.3 and V1R5. The Service Provider Adapter information will be available in a stand-alone Service Provider Adapter Guide.
- Appendix I, A Well Behaved Service Provider. The information will be available in the stand-alone Service Provider Adapter Guide.
- Appendix K, OFX/GOLD Translator Messaging Mapping and Service Provider Adapter (SPA) Development Guide. This OFX information will be available in a stand-alone OFX Guide.
- Appendix L, Master OFX Support Matrix. This OFX information will be available in a stand-alone OFX Guide.

### Deleted Information

- Chapter 4, Customization Via the JavaScript API. With Release V1R6, IFS no longer supports JavaScript.
- Chapter 7, Bill Presentment - Business Flows
- Chapter 8, Bill Presentment - Retrieval of Bill Image
- Appendix F, JavaScript API Support. With Release V1R6, IFS no longer supports JavaScript.



---

## Chapter 1. Customizable Web Server (CWS) Overview

Interactive Financial Services (IFS) provides each Financial Institution (FI) with the capability to customize the look and feel of its system through customized development for the Customizable Web Server (CWS). By developing for the CWS, a FI can provide its end-users (customers) with a system whose user interface reflects the unique style and functions desired by that FI. This chapter describes the relevant components in customization, the CWS environment, and significant design issues.

**Note:** Any new page development must be Java pages and not JavaScript pages. It is highly recommended to convert the JavaScript pages to Java pages as soon as possible. See Chapter 4. Writing Java Servlets for the IFS CWS for more information.

---

### Components of the CWS Customization

Before beginning customizing, the FI must understand the IFS architecture. Figure 1 illustrates the most basic components of this customization.

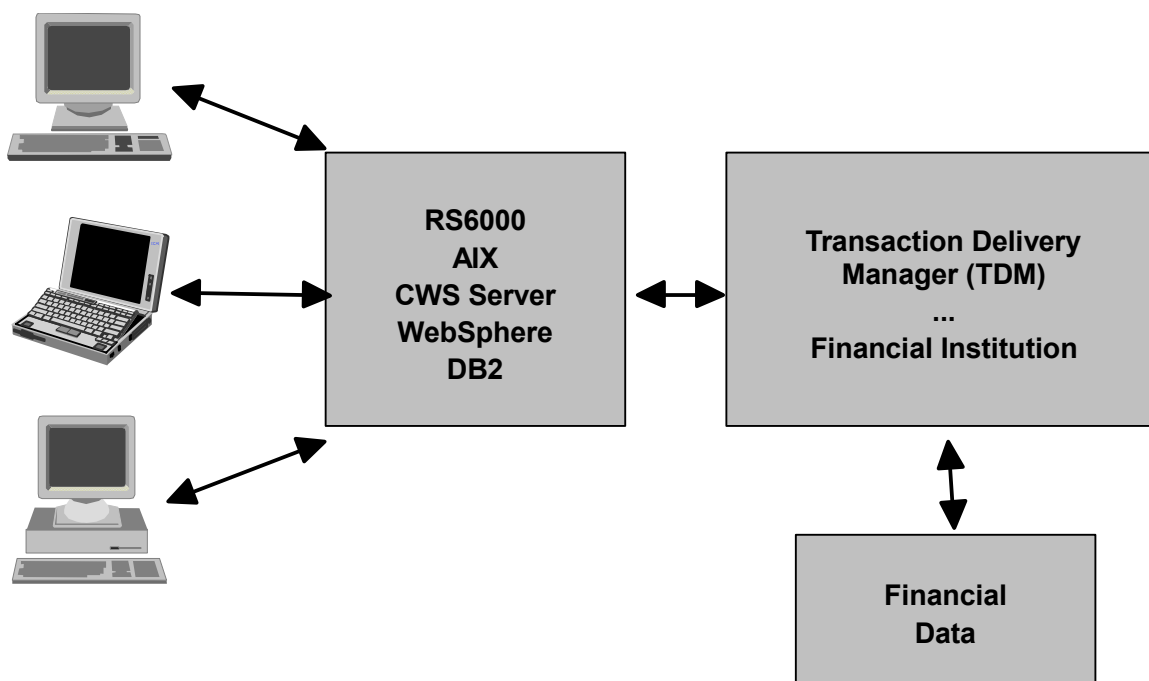


Figure 1. Basic Areas of Customization

### Web Browser

End users who have never connected to the Internet, can, as an option at the financial institution, be provided with either a starter package complete with current browser release or with start-up requirements.

See the *IFS Planning Guide* for details on the software requirements for IFS.

### The Customizable Web Server (CWS)

The CWS encompasses all hardware and software required to serve the IFS Web pages to the Internet. Working with the CWS to permit customized look, feel, and function of the financial institution is the primary focus of this chapter.

The primary method of customizing the CWS server is through the programming interface. Besides familiarity with standard messages the programmer should understand the Application Programming Interface JCWSAPI for Java. This section explains some of the considerations when using the API.

Additional information on the JCWSAPI can be found in the JavaDoc in the cws/docs/jcwsapi/JavaDocs directory of the v1r6m1\_yymmdd.tar.Z file.

The FI may choose to work with IBM Global Services or another third party to customize the Web server on a contract basis. Or, they may choose to use a graphic designer to design the Web pages, and another service provider to implement the Web pages. The graphic designer must understand the flexibility and limitations of building dynamic Hypertext Markup Language (HTML) pages.

**Web Interface:** The Web interface gives end users a means to access the financial institution to perform desired financial institution transactions. This interface is implemented with HTML pages delivered to the customer's browser upon the selection of a Uniform Resource Locator (URL).

Java code is contained within the HTML pages, to allow access to FI data, customer preferences, and other database or legacy system information. The CWS, using IBM WebSphere Application Server (WebSphere) and the Java Virtual Machine (JVM), provides the tools necessary to interpret Java code and send data to and access data returned with standard messages.

Except for temporary cached information obtained during its normal operation, the CWS does not store customer data of any kind. This reduces the chance of data theft, preserves the status of the FI's data center as the primary data source, and eliminates the need for database replication. The architecture isolates functions to machines that are most efficient at their dedicated tasks, which makes processing very efficient.

**JCWSAPI and Java:** The Java CWSAPI (JCWSAPI) provides a programming interface between the HTML pages and the software that communicates with the Transaction Delivery Manager (TDM). The objective of the JCWSAPI is to make every unique data element in the standard message data definition available in its atomic form to the Java Servlet Writer. These data elements are presented to the HTML page in the form of methods on Java objects. By creating an instance of a particular JCWSAPI Java object, the customizer of a Web page can embed any standard message data element in the HTML file at the desired location.

After the JCWSAPI is invoked, and the data or transaction request has been made and processed, data elements (or errors) are retrieved from the FI by the TDM, and are placed into the Java object. These methods can be displayed on an HTML page.

## Transaction Delivery Manager

The Transaction Delivery Manager (TDM), also known as the Core Controller (CC), is responsible for retrieving data elements from the FI based on requests made by standard messages. The data is then passed with MQSeries to the standard message manager and logged. There can be one TDM for many Web servers. However, each FI's transactions and the FI data center links are separate from all others.

Normally all data will be stored at the FI's data center, if the FI chooses to use strip files rather than a direct connection to the TDM. The TDM will maintain customer profile information, account information, and transaction records.

The TDM also passes standard messages (such as structured messages), to the messaging server where they are handled by the FI's customer service messaging work queue.

## The Financial Institution

The FI is responsible for handling the standard messages sent to existing systems. Therefore, the FI must make any necessary modifications to its back end to interoperate with the IFS system; IFS will, however, work closely with the FI by providing tools necessary to translate standard messages to messages usable by the FI's systems. Each FI data item must be matched up to a standard message field; for example, the bank must choose the appropriate account balance item to be displayed to the consumer on the appropriate Web page.

Most FIs will choose to establish a direct link between the FI and the TDM. Frequently this link will be frame relay, using protocols such as SNA with LU 6.2 (a minority of financial institutions may choose IP for the link). Other financial institutions may utilize strip files pulled from the FI's data. In the former case, data will be real time; in the latter, financial data on the TDM and the financial institution's existing systems will not be synchronized until both databases are updated.

---

## Understanding the CWS Environment

Before beginning to customize, it is important to understand the CWS environment.

### The Development Environment

**AIX:** IFS uses the IBM AIX operating system on an IBM e-server pSeries or RS/6000 hardware platform. As IBM's premier UNIX operating system, AIX was designed especially for commercial client-server environments requiring scalability, multi-tasking and multi-user functionality. The developer uses AIX to install and manage components as well as compile, test, and debug various subsystems.

**IBM WebSphere Application Server Software:** The IBM WebSphere Application Server software allows the user to deploy and manage various Enterprise and Web applications across multiple systems. The Administrative console lets an administrator check the status of many servers and deploy an application such as CWS to any of the servers. Versions of WebSphere also include support for Enterprise Java Beans, XML, CORBA and IBM MQ Series.

**Java Servlets:** CWS uses Java Servlets to perform many tasks on the server. This includes validating customer input, communicating with the TDM, and formatting data before it is sent to the client web browser. Typical high level banking tasks include logging on and off, checking financial records, transferring funds, and requesting information. Servlets are often used to present customized, dynamic web pages to the customer.

**Java Servlet Pages (JSPs):** CWS uses JSPs to help with presentation oriented tasks. These JSPs can contain customized tags that can aid in rapid development of customized web pages presenting financial institution data.

---

## Chapter 2. Migrating to V1R6M1 and WebSphere 4

This chapter discusses what is involved in migrating your V1R5 or V1R6 with WebSphere Application Server version 2 to V1R6 with WebSphere Application Server version 4.

### Benefits of Using WebSphere 4.0

Delivery of V1R6M1 provides the capability of running the CWS as a web application inside of the WebSphere 4.0.2 environment. As a result, all of the benefits and features of the Java/2 Enterprise Edition version 1.2 (J2EE) specification are now available to developers of CWS applications including the following APIs (Application Programming Interfaces):

- Java Servlet 2.2
- Java Server Pages 1.1
- Enterprise Java Beans 1.1
- JDBC (Java Database Connectivity) 2.0
- JTA/JTS (Java Transaction API) 1.1
- JNDI (Java Naming and Directory Interface) 1.2.1
- JAF (JavaBeans Activation Framework) 1.0
- XML4J (XML parser) 3.1.1
- XSL (Extensible Stylesheet Language) 2.0
- RMI-IIOP (Remote Method Invocation) 1.0
- JMS (Java Messaging Services) 1.0.1
- Java Mail 1.1

In addition to the support for the above APIs, the following features are available in WebSphere 4.0.2:

- Full J2EE compliance
- Web services support
- Database connection management pooling
- XML parsing
- Expanded database support
- Built-in web server
- Firewall support
- Multiple application servers
- Operating system, LDAP or custom authentication
- Directory services

There are many other features of WebSphere 4.0.2 not included in the list above. For a full description of these features please refer to Chapter 2 of the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*. For more information about the J2EE 1.2 specification, please refer to Chapter 3 of the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*, or Sun's on-line documentation found at <http://java.sun.com/j2ee/>.

---

### Migrating from V1R6 with WAS 2 to V1R6 with WAS 4

First time users of CWS may skip to Chapter 3. Installing CWS Components. However, previous users of CWS must perform the following steps when upgrading to V1R6M1:

1. Software prerequisites
2. CWS distribution format
3. Installation and Configuration of the CWS
4. Updates to CWS Customer Java code
5. Character Encoding Support

Each of these issues is discussed in detail below.

## Software Prerequisites

CWS V1R6M1/WebSphere 4 has the following software prerequisites:

- AIX 4.3.3 Fix pack 9 or AIX 5.1
- WebSphere 4.0.2 Advanced Edition
- IBM HTTP Server (IHS) 1.3.19.1 (comes bundled with WebSphere 4.0.2)
- DB2 7.2.3 (DB2 7.1 Fix pack 5)
- MQSeries 5.2 Configured to use the system management tool (smitty)

### AIX

AIX must be upgraded to the appropriate level before CWS prerequisite software can be upgraded or installed. Systems running AIX 4.3.3 must be upgraded to recommended maintenance level 9 (4.3.3.0.09). Instructions on how to download and install fixes for AIX can be found on-line at <http://techsupport.services.ibm.com/server/nav?fetch=pm>.

### WebSphere

CWS requires WebSphere 4.0.2 Advanced Edition to function properly. Previous versions of CWS used WebSphere 2.0.3; unfortunately, there is no supported upgrade path from version 2.0.3 to 4.0.2. WebSphere 2.0.3 should be uninstalled from the system and WebSphere 4.0.2 installed. WebSphere 4.0.2 installation instructions are included with the WebSphere 4.0.2 distribution and in chapter 10 of the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*.

### IBM HTTP Server

Netscape s iPlanet HTTP Server has been replaced in V1R6M1/WebSphere 4 with IBM HTTP Server (IHS) 1.3.19.1. IBM HTTP Server (IHS) is included with WebSphere 4.0.2 and is installed as an optional component during WebSphere 4.02 installation. For more information on installing IHS please refer to the installation documentation included with WebSphere 4.0.2 and chapter 10, section 10.3 of the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*.

### DB2

WebSphere 4.0.2 stores all configuration information inside of a DB2 database. Therefore, a connection to a database must be made available to WebSphere through either a full installation of DB2 UDB (Universal Database) on the local system or by using DB2 Connect to communicate with a remote database server.

**Note:** The choice of using DB2 UDB or DB2 Connect is beyond the scope of this chapter, please refer to Chapter 3. Installing CWS Components for more information about DB2 topology.

If DB2 UDB or DB2 Connect is currently installed on your system it may be upgraded to the appropriate level by downloading and installing FixPaks. Both DB2 UDB and DB2 Connect must be at level 7.2.3, which is the same as 7.1 FixPak 5. More information, and downloads, can be found on-line at <http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/win02unix/support/download.d2w/report>.

### MQSeries

MQSeries must be at level 5.2 for the CWS to function properly. Installations of MQSeries may be upgraded to level 5.2 by downloading and installing FixPaks. More information about upgrades can be found on-line at <http://www-3.ibm.com/software/ts/mqseries/support/>.

**Note:** existing MQSeries queue managers and queues may be used with WebSphere 4.0.2 if MQSeries has been updated to version 5.2.

## CWS Distribution Format

The CWS is now packaged inside of a Web Archive (WAR) file as a J2EE web application. Distributing the CWS in this format provides support for J2EE functionality and simplifies the installation process. More

information about the J2EE and WAR files can be found in Appendix H. Building And Deploying CWS Web Applications In WebSphere 4.

## Installation and Configuration of the CWS

Installation of CWS has changed significantly in WebSphere 4.0.2 and detailed, step-by-step, instructions are provided in Chapter 3. Installing CWS Components.

Changes have been made to the CWS configuration file and it is strongly discouraged to use configuration files from older versions of CWS. However, the majority of options and values have remained unchanged. It is recommended to read through the sample CWS configuration file before making updates. Configuration of the CWS is described in detail in Chapter 3. Installing CWS Components

## Updates to CWS Customer Java Code

Changes were made to the JCWSAPI to support the move from WebSphere 2.0.3 to WebSphere 4.0.2. As a result, code that uses the JCWSAPI must be recompiled in the WebSphere 4.0.2 environment, and in some cases, updated. More information about the changes, and possible migration problems, can be found in Appendix G. Migrating from Java 1.1.8 to Java 1.3.0.

## IfsSystem.properties File

The values for all *public static* variables in the **com.ibm.ifs.gold.IfsSystem** class can now optionally be set in the Java properties file **IfsSystem.properties**; which can be found in the `WEB-INF/classes` directory. `IfsSystem.java` should no longer be edited and recompiled, if variable values need to be changed the appropriate updates should be made to `IfsSystem.properties` and CWS restarted. Please read the instructions in the `IfsSystem.properties` file for more information.

## Character Encoding Support

The process of configuring CWS to use different character encodings has changed from WebSphere 2 to WebSphere 4. To use an encoding such as UTF-8, set the JVM (Java Virtual Machine) system property **client.encoding.override** to UTF-8. This process is further explained in the *Update the Application Server Configuration* section in Chapter 3.

---

## Migrating from V1R5 to V1R6

This section will address only the changes that need to be made to the customer's existing servlets and the server config files written for V1R5, in order to run on V1R6.

Delivery of V1R6M1 provides the capability for implementing National Language Support/Double Byte Character Set and Small Business Enhancements. For a description of these enhancements, see the *IFS Overview and Release Changes Guide*.

Table 1. V1R5M1 to V1R6M1 CWS Migration Steps

Set ulimit setting for CWS	See <i>Preparing to Install the Customizable Web Server (CWS) on AIX</i> for the ulimit settings.
Customizable Web Server (CWS)	Follow the directions presented with the Release Tar file. Additionally, see <i>Installing and Configuring the CWS</i> .
CWS Configuration File	Make changes to the configuration file as described in <i>Resource Bundle</i> . Also refer to the <i>Sample CWS Configuration File</i> .
CWS Toolkit	A new CWS Toolkit was shipped. See <i>Installing the CWS Toolkit</i> .
Ifs.war/tools/startweb script	See <i>NLS Enablement</i> .
NLS/DBCS Enablement	See <i>NLS/DBCS</i> .
Implement NLS on the CWS	Refer to <i>Writing Java Servlets for the IFS CWS</i> .



Handling code-to-text translations	See <a href="#">Code-to-Text Translations</a> .
User Profile Changes	Refer to <a href="#">User Profile Changes</a> .
Transaction API Changes	Refer to <a href="#">Transaction API Changes</a> .
Integrate CWS information into the MRM	Refer to <a href="#">MRM/CWS Integration</a> .
Removed Java Classes	Refer to <a href="#">Removed Java Classes</a> .
Attribute Names Changes	Refer to <a href="#">Attribute Name Changes</a> .
Wide Character Support	Refer to <a href="#">Gold Manager Wide Character Support</a> .
Disable JavaScript	Refer to <a href="#">Disable JavaScript</a> .
IBM HTTP Server 1.3.19.1	Netscape Enterprise Server has been replaced by IBM HTTP Server 1.3.19.1 and IBM WebSphere 4.0.2 Advanced Edition
WebSphere - Install IBM WebSphere 4.0.2 Advanced Edition	Refer to <a href="#">WebSphere</a> .
Java Servlet Development Kit	Refer to <a href="#">Writing Java Servlets for the IFS CWS</a> .
Writing Java Servlets	Refer to <a href="#">Writing Java Servlets for the IFS CWS</a> .
Property Files	Refer to <a href="#">Resource Bundle</a> .
Implement Multiple User IDs	Refer to <a href="#">Considerations in Using Multiple User Ids</a> .
MQ Handles	Refer to <a href="#">MQ Maximum Number Handles</a> .
SNA Configuration	No changes required.
MQ Configuration	No changes required.

## NLS/DBCS

In V1R6, many new functions have been introduced to support the NLS/DBCS enablement. The IFS Web Server now supports a locale object, called the **session locale object**, for each user session. This locale object is created and cached in the user session. The session locale object can be dynamically set based on each client's request during runtime. The contents of the web pages are created using the current session locale. This allows end users to view the same web page in different locales. Also, there is DBCS support using UTF-8 charset. Multiple currency is enabled. Refer to [Writing Java Servlets for the IFS CWS](#) for details about these new functions.

## Code-to-Text Translations

In previous releases, every transaction class (Gt\*.java) or data class (Gs\*.java) that had a field holding a non-descriptive code also had a corresponding translated text field. The name of this field had the same name as the code, but appended with `_text` . Like other fields, this field was also extracted from the response buffer of the GOLD transaction and had corresponding setters and getters to set and get the field values respectively.

In the current release, a mechanism has been defined to handle code-to-text translations using the getters of the `text` fields i.e., `get*_text()` methods of transaction and data classes to encapsulate the locales, resource bundles, and the encoded values inside these methods. All physical fields `*_text` of transaction and data classes are removed. The translated descriptive text of encoded values are now retrieved in classes using `get*_text()` methods based on the current session locale. For instance, the `GsAccount` class has a method called `getType_text()` which gives the descriptive text for a type of account whose value is obtained by the method `getType()`. The `getType_text()` method uses `getType()` as the key for the lookup in the auto-generated resource bundle using the locale provided (see sample code below).

```
public class GsAccountS extends IfsObject
{
    ...

    protected String type;
    public void setType(String type)
    {
```

```

this.type = type;
}

public String getType()
{
return type;
}

public String getType_text() throws IfsException
{
return IfsNLS.getText("IfsAcctSubTpC1Resources", getIfsSession().getLocale(),getType())
}

...
}

```

All the `get*_text()` methods use the static method of a utility class called `IfsNLS` to retrieve the locale-specific text encoded in a resource bundle using the code (key) and the locale. The description of this class is defined in [Writing Java Servlets for the IFS CWS](#) .

The following two changes need to be made to the customer s existing code or server configuration for the code to text translation:

- The property files need to be generated and stored on the web server for the resource bundles used in the `get*_text()` methods. How to generate such property files will be explained in the next section.
- All `get*_text()` methods throw an `IfsException`. Any servlet that calls the `get*_text()` method needs to wrap the method in a try/catch block.

## Resource Bundle

Gold messages can contain fields that are defined as `code-to-text` fields. Such a field in the message normally contains a coded value which can be correlated to a text phrase. In V1R5, `ListResourceBundle` classes, which are generated based on the code to text tables in the Core Controller, are used for the code to text translations. In that design and implementation, any future changes made to the contents of the resource bundles require recompilation of the resource bundle classes.

In V1R6, in order to support easier maintenance of the code to text table contents, the resource bundles are changed to `PropertyResourceBundle`. For the naming convention and the contents of the property files, please see [Writing Java Servlets for the IFS CWS](#) . The property files replace the previous resource bundle classes, `IfsBrcErrorTextBundle`, `Ifs3tkErrorTextBundle`, `IfsCwsErrorTextBundle`, and `IfsSystemErrorResourceBundle`. In addition, many other property files, one for each code to text field type, are created during the web server startup time. The building of the property files is under the control of several variables in the CWS server config file:

- The first entry, `wsp_ckt_active`, will indicate whether the Code-to-Text to `PropertyResourceBundles` function is active. If `wsp_ckt_active=0` then no property files will be created. A value of 1 indicates that the files will be created. Since WebSphere 4 doesn t keep files from a previous installation of an Enterprise Application, you will need to create the files (set `wsp_ckt_active` to 1) the first time the application is started after you deploy (or redeploy) the application.
- The second entry, `wsp_ckt_prb_path`, will indicate the pathname in which to store the property files. The new entry will be `wsp_ckt_prb_path=path` If `path` begins with a `/` it will be treated as an absolute path (from root). If the path does not begin with `/` it will be treated as a relative path from the application deployment path.
- The third entry, `wsp_ckt_iso_length` indicates the length of the ISO fields used to return the country code and language code in the CODERFSH transaction. Currently only a value of 3 is supported (`wsp_ckt_iso_length=3`).

No mechanism to dynamically update the property files after server startup is provided. The server will need to be stopped and restarted to receive recent code-to-text table updates. Note that the startup time of the CWS will now be dependent on the quantity of CTT tables downloaded.

The above are necessary changes to make the existing customer code and server config files to run under the V1R6. In order to use other new functions, such as the dynamic session locale object, UTF-8 charset support, new currencies, `IfsNLS` class, and the formatting of currency, date, and time, please see [Writing Java Servlets for the IFS CWS](#) .

## User Profile Changes

Release V1R6 introduced changes to the User Profile in support of Small Businesses. One change to the User Profile is a new field for a Primary Consumer ID. **The User Profile that is returned by the FI as a response to a Logon or Self-Registration message must include a non-blank and non-null value for this Primary Consumer ID.** The Primary Consumer ID value returned at Logon or Self-Registration is placed in the gold message header of every gold message sent to the FI.

**Note:** The Primary Consumer ID value is placed in the gold message for **every** gold message executed, not just the new set of messages for small businesses.

## Transaction API Changes

In V1R6, the execute method of every gold transaction now throws an `lfsException`. This requires changes to every servlet developed with IFS gold transactions prior to release V1R6. Specifically, any servlet that calls the execute method of a gold transaction must do so within a try/catch block.

## MRM/CWS Integration

The integration of CWS information into the Message Repository Manager (MRM), and the corresponding formalization of business rules into the MRM, has introduced variations between the V1R5 and V1R6 IFS CWS Java Class public interfaces. The result is a more heterogeneous interface, though many variances within the interface are still preserved. The changes are categorized as follows:

- Removed Java Classes
- Attribute Name Changes

### Removed Java Classes

Table 2. Removed Java Classes

Java Class	Resolution	Comments
<b>GsAcctId</b>	Use <b>GsAccount</b>	The Java class <b>GsAcctId</b> serves the same purpose as the hand-written class <b>GsAccount</b> : both represent account identifiers. Using two distinct types for the same information introduces unnecessary complexity and incompatibility between fields within the interface, due to Java's strong type checking. Therefore all uses of the previously generated class <b>GsAcctId</b> has been replaced with <b>GsAccount</b> .  Note the following differences between <b>GsAcctId</b> and <b>GsAccount</b> : <ul style="list-style-type: none"><li>• The attribute <b>goldNdx</b> in <b>GsAcctId</b> is removed from <b>GsAccount</b>.</li><li>• The attribute <b>subType</b> in <b>GsAcctId</b> is renamed to <b>accountSubType</b> in <b>GsAccount</b>.</li><li>• The attribute <b>clrgHseFild</b> in <b>GsAcctId</b> is renamed to <b>clearinghouseFIID</b> in <b>GsAccount</b>.</li></ul>
<b>GsDestFiDtl</b> <b>GsGetDestFiClhsTpDtl</b> <b>Gs</b> classes in general	Enable appropriate <b>Gt</b> class in MRM	A <b>Gs</b> class is produced only if it is used by an appropriately enabled <b>Gt</b> class in the MRM. This includes Gt classes generated by the MRM CWS Emitter (marked <code>Emit Java</code> in the MRM) and hand-written classes (marked <code>Java Enabled</code> ).
<b>GtApplicationLogon</b>		System message, not appropriate for CWS Java Classes
<b>GtLogoffCTW</b>		CTW message, not appropriate for CWS Java Classes
<b>GtLogonCTW</b>		CTW message, not appropriate for CWS Java Classes
<b>GtPayeeInquiryData</b>	Use <b>GtPayeeInquiry</b>	The MRM tool design supports the generation of a single java class to represent each transaction. <b>GtPayeeInquiry</b> and <b>GtPayeeInquiryData</b> are two separate java classes representing the same transaction. The purpose of <b>GtPayeeInquiryData</b> was to represent a particular use of the payee inquiry transaction, thereby simplifying the coding process. <b>GtPayeeInquiryData</b> is a degenerate form of <b>GtPayeeInquiry</b> .
<b>GtSelfDef</b>	Use <b>GtSelfDefiningInfo</b>	Redundant/duplicate message.
<b>GtServerShutdown</b>		System message, not appropriate for CWS Java Classes
<b>GtServerStartup</b>		System message, not appropriate for CWS Java Classes

## Attribute Name Changes

The following table lists all changes between the V1R5 and V1R6 attribute names. The scope of this section does not include new fields added to the Java Classes in V1R6.

Note that the focus of this table is the V1R5 attributes in the second column; IFS Java Classes may occur in multiple rows in the first column. If the number of occurrences of a class is more than one, then the occurrence and total number of occurrences of that class are shown as (occurrence/total) following the name.

Table 3. Attribute Name Changes

Java Class(es)	V1R5 Attribute	V1R6 Attribute	Comments
Various	account_packed	<i>removed</i>	Removed with change from <b>GsAcctId</b> to <b>GsAccount</b> .
Various	*_packed	<i>removed</i>	Internally used field, removed where not required for internal use. These fields should probably be marked <b>private</b> .
<b>GsBusInvPymtRec</b>	invLineNunber	invLineN <u>u</u> mber	Spelling correction
<b>GsGrntrInfoDtl</b>	reqGrntrIncDtl	grntrIncDtl	Only request message fields are prefixed by <u>req</u> .
<b>GslraAcctDtl</b>	ifsPDistransitRoutingNumberID transitRoutingNumberID	<i>removed</i> <i>removed</i>	Descope in the MRM.
<b>GsStaffPrflItem</b>	staffTitleCode_text	<i>removed</i>	The new attribute enables generation of a code-to-text translation method. (See Note 1)
<b>GtAuthorizeAccountRegistration</b> <b>GtDisclosureAcceptance</b>	reqConsumerID reqChangeableConsumerID	<i>removed</i> <i>removed</i>	Autofilled (See Note 2)
<b>GtBillPayVerification</b> (1/2) <b>GtCreditCardDispute</b> <b>GtReportBillPayProblem</b> (1/2)	reqHomePhone reqWorkPhone	<i>removed</i> <i>removed</i>	Autofilled (See Note 2)
<b>GtBillPayVerification</b> (2/2) <b>GtReportBillPayProblem</b> (2/2)	reqFaxNumber	<i>removed</i>	Autofilled (See Note 2)
<b>GtCSRConsumerStart</b> <b>GtUserProfile</b>	cnsmrInform	consumerInformation	Conform with use of this field in other contexts.
<b>GtCancelStopCheckPayment</b> <b>GtCheckCopy</b> <b>GtCustomerEmailToFI</b> <b>GtFIEmailToCustomer</b> <b>GtGPCustomerMessage</b> <b>GtSalesSlipCopyRequest</b> <b>GtSmallBusStopRecurDraft</b> <b>GtStatementCopy</b>	reqContactfreeFormText	reqFreeFormText	Conform with use of this field in other contexts.
<b>GtGetDestClearinghouseProfile</b> <b>GtGetSecuritiesDescriptions</b> <b>GtInvestmentAccountHistory</b> (1/2)	moreF moreF_text totTxnMchVal totTxnSndVal crsr	moreFlag moreFlag_text totalTransactionsMatched totalTransactionsSent cursor	Changing to the V1R5 names will bring these commonly used fields out-of-sync with the names used in other messages for the same elements. Common name simplifies future maintenance using the MRM.
<b>GtInvestmentAccountHistory</b> (2/2) <b>GtStopCheckPaymentInquiry</b>	reqMaxNbrRecVal reqLstNbrRecVal reqCrsr	reqMaxNumberRecords reqLastNumberRecords reqCursor	Changing to the V1R5 names will bring these commonly used fields out-of-sync with the names used in other messages for the same elements. Common name simplifies future maintenance using the MRM.
<b>GtInvestmentOrderCancel</b> <b>GtInvestmentOrderCancelChange</b> (1/2) <b>GtInvestmentOrderStock</b> (1/2)	respCodeCl respCodeCl_text fiRfrNbrId srvcPrvdRfrId fiProcInd fiProcInd_text	responseCode responseCode_text referenceNumber serviceProviderReference fiProcInd fiProcInd_text	Changing to the V1R5 names will bring these commonly used fields out-of-sync with the names used in other messages for the same elements. Common name simplifies future maintenance using the MRM.
<b>GtInvestmentOrderCancelChange</b> (2/2) <b>GtInvestmentOrderStatusInquiry</b> <b>GtInvestmentOrderStock</b> (2/2)	cnsmrId chCnsmrId permUserId	consumerID changeableConsumerID permUserID	Changing to the V1R5 names will bring these commonly used fields out-of-sync with the names used in other messages for the same elements. Common name simplifies future maintenance using the MRM.

Notes:

1. The rules for establishing/generating a Code-To-Text translation for an attribute are:

A valid-value must be defined for the element (in the MRM), in the context of at least one message (valid values are defined in message context).

The attributes immediate context of use must be in a reply message.

2. For V1R5 the set values (via the public set methods for these attributes were ignored, and the values were (re)populated in the CWS - much like the autofilled attributes. In previous releases the values set using the public interface were ignored and overwritten within CWS, therefore an external interface is not necessary. The choice was made to preserve the V1R5 behavior, as opposed to the interface, of the IFS Java Classes. Therefore these attributes have been formally marked as `autofilled` in the MRM tool. The previous state that allowed such fields to exist has been eliminated from generated code.

## NLS Enablement

The `ifs.war/tools/startweb` script must assign and export the value of the environment variable `MTINLS`. This is currently set to the location of the `cws/nls/lib` subdirectory under the CWS instance installation.

## Gold Manager Wide Character Support

This function added the following configuration options to the config file:

```
#-----  
# required parameters  
#-----  
# these parameters are inserted into the gold header for startup functions:  
web_country_code=USA  
web_language_code=ENG
```

These parameters are inserted into the Gold header during startup functions.

## Disable JavaScript

This function removes JavaScript support from the CWS.

## MQ Maximum Number Handles

The maximum number of MQ handles must be increased because of the increase in the number of Gold messages that can potentially be transferred over different queues.

This number needs to be increased in the Operating System. The default is 256 and should be raised to 600 via the `smitty` tool.

---

## Chapter 3. Installing CWS Components

---

### Installation Overview

This chapter outlines the software requirements and steps needed to install CWS V1R6M1 using IBM WebSphere 4.0.2 in your IFS environment. The instructions that are given were created and tested on an IBM RS/6000 running AIX 4.3.3 FixPak 9 and AIX 5.1 and are specific to the environments of those systems (i.e. software installation roots, location of home directories, etc.). The environment of the system on which you are installing should be taken into consideration when running any of the specific commands listed below. The following are the general steps that are required to install CWS on AIX:

- Install Software Prerequisites
- AIX Performance Tuning
- DB2 Setup
- Creating a CWS Instance

The following instructions detail the process of installing CWS to connect to FISim using the *All Local MQSeries Configuration* described in appendix D. Information on using other configurations, such as the *Remote MQSeries Configuration* can also be found in appendix D.

**Note:** If you are migrating an existing CWS instance please read Chapter 2. Migrating to V1R6M1 and WebSphere 4 before continuing.

### Install Software Prerequisites

CWS requires the following software to be installed for proper functioning. See the installation instructions and manuals for each of the prerequisites listed below for more information on installing and configuring each respective product.

- AIX 4.3.3 Fix pack 9 or AIX 5.1
- WebSphere 4.0.2 Advanced Edition
- IBM HTTP Server (IHS) 1.3.19.1 (comes bundled with WebSphere 4.0.2)
- DB2 7.2.3 (DB2 7.1 FixPak 5)
- MQSeries 5.2 Configured to use the system management tool (smitty)

**Note:** The *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook* gives very detailed instructions on installing WebSphere 4.0 advanced edition on AIX. Please reference this material before installing.

### AIX Performance Tuning

The AIX system ulimits, paging space, and bound processes must all be set correctly to achieve maximum performance.

**ulimit Settings:** The system limits for all parameters must be set to unlimited. This is true for both the soft limits and hard limits.

Hard limits for the system are reported by:

```
ulimit -H
```

Which should report :

```
Unlimited
```

For the UID of the webserver the soft limits can be inspected by:

```
ulimit -a (assuming the user running this command is the UID of the webserver)
time(seconds)          unlimited
file(blocks)           unlimited
data(kbytes)           unlimited
```

```
stack(kbytes)          unlimited
memory(kbytes)         unlimited
coredump(blocks)       unlimited
nofiles(descriptors)   unlimited
```

See the *AIX System Manual* for more information on the *ulimit* command.

**Paging Space:** Sufficient AIX paging space must be available. The actual amount will vary by the amount of memory installed in the system as well as by the number of CWS instances and other applications run by the customer, including whether the customer is using early or late paging space allocation (using the PSALLOC environment variable). The following formula should yield sufficient paging space for most environments.

$$\text{Total paging space} = 512 \text{ MB} + (\text{memory size} - 256 \text{ MB}) * 1.25$$

See *The Paging Space Overview* in *AIX Version 4.3 System Management Guide: Operating System and Devices* and the *IBM Redbook AIX 64-bit Performance in Focus (SG24-5103-00)* for more information.

**Bound Processes:** Run the Java and web server process as **unbound** on all servers.

Subsequent chapters will address installing the CWS and configuring the different server types, as well as installing the CWS Toolkit.

## DB2 Setup

### Product Level

WebSphere Application Server version 4.0.2 requires DB2 version 7.2.3 to function properly, for clarification, DB2 7.2.3 is really DB2 7.1 plus official FixPak 5. You can run the ,db2level- command to determine what version of DB2 you are using. For DB2 v7.2.3 the db2level command should return the following product signature:

```
# db2level
DB21085I  Instance "db2inst1" uses DB2 code release "SQL07023" with level
identifier "03040105" and informational tokens "DB2 v7.1.0.55", "s011211" and "U480359".
```

You want the ,code release- to be ,SQL07023- when you run the db2level command.

You must also run the /usr/lpp/db2\_07\_01/java12/usejdbc2 script after installing or updating to DB2 7.2.3. This script installs the JDBC 2.0 support needed by WebSphere Application Server 4.0.2.

### Configuring the Database Manager to Use Shared Memory

WebSphere 4.0.2 requires the DB2 database manager to use extended shared memory; this can be accomplished by following the steps outlined below:

*Assuming the db2instance owner's userid is db2inst1*

Log in as the DB2 instance owner, db2inst1, using the su command as follows:

```
# su - db2inst1
```

Note that when you log in as db2inst1, the command prompt changes from the # symbol to a dollar sign (\$) to indicate a change in your login identity.

If this is the first time that you have logged in as the DB2 instance owner, you may be prompted to change the password. Enter a new password and press Return. When prompted, type the new password again and press Return.

**Note:** DB2 requires a password of *eight* or fewer characters.

Next, turn on the use of extended shared memory by setting the EXTSHM environment variable using the following commands:

```
$ db2stop
```

```
$ export EXTSHM=ON
$ db2set DB2ENVLIST=EXTSHM
$ db2start
```

These commands must be executed each time you stop and start the DB2 database manager. If DB2 is configured to start when the system is rebooted you can add the above commands to the `/etc/rc.db2` file.

## Creating a Database for WebSphere Application Server

Perform the following steps to create a database named `was` and set the DB2 application heap size:

**Note:** If WebSphere 4.0.2 is already installed on your system then you don't have to create the WAS database. Follow the steps below substituting the name of the currently installed WebSphere administrative database for `was`.

1. Login as the DB2 instance owner, `db2inst1`.
2. Create a database named `was` by using the `db2 create database` command, as follows:

```
$ db2 create database was
```

This process can take several minutes to complete.

3. Set the application heap size by using the `db2 update db config` command, as follows:

```
$ db2 update db config for was using applheapsz 256
```

If an application heap size of 256 does not work for your system, increase it to 512.

4. Stop and start the database for your changes to take effect.

```
$ db2stop
$ export EXTSHM=ON
$ db2set DB2ENVLIST=EXTSHM
$ db2start
```

## Potential Problems

### TCP/IP Loopback

If using local DB2 databases for data access by session clients, in some cases, multiple connections for session clients cannot be established successfully and you may see errors in the WebSphere application server's `stdout.txt` log and the graphical administration client like the following:

```
7c9ae126 SeriousEventD F ADMS0027E: Could not initialize persistent storage for serious events.
Got exception COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] SQL1224N A database agent could
not be started to service a request, or was terminated as a result of a database system
shutdown or a force command. SQLSTATE=55032
```

To avoid stale connections when there are large numbers of session clients, catalog the DB2 databases to use a TCP/IP loopback by following these instructions:

1. Set up a TCP/IP port in `/etc/services` file, if a port for remote DB2 clients has not been established yet. If the port has been claimed you should see a line like the following:

```
db2cdb2inst1 50000/tcp # Connection port for DB2 instance db2inst1
```

2. Ensure that the TCP/IP communication protocol has been specified in the `DB2COMM` registry parameter.
  - To check the current setting of the `DB2COMM` parameter, enter `db2set DB2COMM`.
  - To update the `DB2COMM` registry variable to include TCP/IP, use the `db2set` command.

For example:

```
db2set DB2COMM=existing_protocol_names, tcpip
```



3. Update the SVCENAME database manager configuration parameter to the connection service name as defined in /etc/services (step 1).

*For example:*

```
db2 update dbm cfg using svcename connection_service_name
connection_service_name would be db2cdb2inst1 from the example in step 1.
```

4. Catalog the loopback node. For example:

```
db2 catalog tcpip node node_name remote 127.0.0.1 server
connection_service_name
```

5. Catalog the database as follows:

```
db2 catalog db database_name as database_alias
db2 uncatalog db database_name
db2 catalog db database_alias as database_name at node node_name
```

This allows you to implement a TCP/IP loopback without needing to change the application to connect to the new alias.

6. Stop DB2 and start it again to refresh the directory cache.

### Reinitialize WebSphere Administrative Database

If you experienced a problem connecting to the database when starting WebSphere for the first time, and as a result updated the database configuration or created a TCP/IP loopback, then the WebSphere administrative database may not have been initialized. In order for WebSphere to reinitialize the database you need to update the /usr/WebSphere/AppServer/bin/admin.conf file and set the following properties values to true:

```
com.ibm.ejs.sm.adminServer.createTables=true
install.initial.config=true
```

This causes WebSphere to create and initialize the database tables that are necessary for it to function properly. This process can also be used to repair a broken WebSphere instance without having to reinstall the entire application.

**Warning:** This process will destroy all changes that have been made to WebSphere since installation. For example, all application servers and enterprise applications will be removed. This should only be attempted if the administrative database was never created in the first place or when attempting to repair a corrupted administrative database.

## Creating a CWS Instance

The following steps describe the process of installing, configuring and testing a CWS instance on a system with all of the software prerequisites using the *All Local MQSeries Configuration* and FISim.

### Create An Instance AIX userid

All CWS instances are executed under unique shared AIX userids with the following values:

- Add userid to group mqm and db2iadm1 (MQSeries and DB2 administrators)
- All other fields are default values.

This can be accomplished by executing the following instructions:

1. Logon as root
2. At the command line type

```
# smitty users
```

3. Select *Add a User*
4. Specify a value for the *User NAME* field (will be referenced as <userid> for the rest of this chapter), and the groups *db2iadm1* and *mqm* for the *Group SET* field.
5. Press enter
6. Press F10 to exit back to the command line
7. Type:
 

```
# passwd <userid>
```

 and enter a password for the new user

## Create Directory Structure - Logs Directory

CWS, IHS, WAS 4 and FISim all produce a number of different log and error files. It is a good idea to place these files in a common directory for quick examination and cleanup. The following commands will produce the recommended directory structure that will be referenced throughout the rest of this chapter:

1. Logon as user <userid>
2. 

```
$ cd
```
3. 

```
$ mkdir -p CWS/logs
```

## Create a WebSphere Application Server

The CWS was designed and packaged to run inside of a WebSphere 4 application server environment. For each CWS instance that will be installed on a system you must create a new application server that will run under the userid created in step 1. To create the application server use the following instructions:

1. Open a WebSphere administrative console (see Appendix F)
2. Click Console->New->Application Server...
3. Click the *General* tab
4. Value for the *Application Server name* text field is <userid>
5. Value for the *Working directory* text field is /home/<userid>/CWS/logs
6. Click the *Advanced* tab
7. Value for the *User ID* text field is <userid>
8. Value for the *Group ID* text field is the groupid that the WebSphere files and directories are owned by, (usually system). The groupid can be determined by running the command
 

```
ls -l /usr/WebSphere
```

 the groupid is the text in the fourth column, for example
 

```
drwxr-xr-x    20  root system 1024 Jan 21 15:24 AppServer
```
9. Click the *OK* button.
10. A dialog box containing command completed successfully will appear. Click on ok to complete this step.

## Create a WebSphere Virtual Host

Each instance of CWS must have its own virtual host defined in WebSphere that specifies which hostnames and ports an application will listen for requests on. Each instance of CWS listens for requests on all possible hostnames but only one port. To create the virtual host, use the following instructions:

1. Open a WebSphere administrative console (see Appendix F )
2. Click *Console->New->Virtual Host...*
3. Click the *General* tab.
4. Value for the *Name* text field is <userid>
5. Click the *Add* button.
6. Click in the text field under the *HostAliases* label type `\* : <port>` where <port> is an integer between 1 and 65535 and is the port this CWS instance will listen on
7. Click the *OK* button.
8. A dialog box containing command completed successfully will appear. Click on ok to complete this step.

## Create an IHS Virtual Host

Each instance of CWS must have its own virtual host defined in IHS. A virtual host specifies which hostnames and ports IHS will listen for requests on and eventually forward to the corresponding CWS instance. To create the virtual host, use the following commands/instructions:

1. Logon as user `root`
2. Change into IHS configuration directory  
**# `cd /usr/HTTPServer/conf`**
3. Back up the current configuration file  
**# `cp httpd.conf httpd.back`**
4. Open the `httpd.conf` file for editing using your favorite editor
5. Search for the text *Listen 80*
6. If the line is preceded with a pound sign ( # ) then uncomment the line by removing the # character
7. In the line immediately beneath *Listen 80* type `,Listen <port>` where <port> is the same integer specified in step 4.
8. Define the virtual host by searching for the text `</VirtualHost>` and immediately beneath it place the following:

```
<VirtualHost [hostname]:[port]>
  ServerAdmin [admin_email]
  DocumentRoot
/usr/WebSphere/AppServer/installedApps/[userid].ear/ifs.war/docs
  ServerName [hostname]
  ErrorLog /home/[userid]/CWS/logs/http-error.log
  TransferLog /home/[userid]/CWS/logs/http-access.log
</VirtualHost>
Where:
```

*[hostname]* is the fully qualified hostname of the system (i.e. *Host.some\_domain.com*)  
*[port]* is the port specified in step 4  
*[admin\_email]* is the Internet e-mail address of the server administrator (i.e. [Someone@somewhere.com](mailto:Someone@somewhere.com))  
*[userid]* is the userid the CWS instance is running as

9. Save the file

10. Restart IHS

```
# /usr/HTTPServer/bin/apachectl restart
```

11. A confirmation that restart is complete appears with the following message:

```
# /usr/HTTPServer/bin/apachectl restart: httpd restarted
```

## Update and Deploy the ifs.war File

The CWS is now distributed as a Web Archive (WAR) file named *ifs.war* which is found in *v1r6m1\_yymmdd.tar.Z* under *cws/bin*. This base WAR file must be updated to include your custom CWS application (see Appendix H for more details). The updated *ifs.war* file must be deployed into a unique WebSphere application server for each instance of CWS that you wish to run on a given system. To deploy the *ifs.war* file use the following instructions:

1. Update the distributed base *ifs.war* file to include your custom CWS application (see Appendix H for more details).
2. Logon as user *root*
3. Open a WebSphere administrative console (see Appendix F)
4. Click *Console->Wizards->Install Enterprise Application*
5. Click the *Install stand-alone module (\*.war, \*.jar)* radio button
6. Click the *Browse...* button to search for the *ifs.war* file or enter the fully qualified path to the *ifs.war* file including *ifs.war* in the *Path* text field. Make sure that the *ifs.war* file is not on an NFS mounted directory, if it is make a local copy on your local subdirectory.
7. Value for the *Application Name* text field is *<userid>*
8. Do not change the text field labeled *Context root for web module*.
9. Click the *Next>* button (repeat this 7 times until Virtual Host selection)
10. Click the *Select Virtual Host...* Button
11. Select the virtual host you created in step 4 and click the *OK* button
12. Click the *Next>* button
13. Click the *Select Server...* Button
14. Select the Application Server you created in step 3 and click the *OK* button
15. Click the *Next>* button
16. A confirmation box appears saying that the application will be installed on the following nodes with installed directory setting with each node as */usr/WebSphere/AppServer/installedApps/<user id>.ear*.

17. Click the *Finish* button.
18. A confirmation box will appear saying the command completed successfully. Click **OK**.
19. In the left pane expand the *WebSphere Administrative Domain* item
20. In the left pane expand the *Nodes* item
21. In the left pane right click on the <hostname> item under *Nodes* and select the *Regen Webserver Plugin* menu option
22. On the lower pane, an event message saying ,Plugin Regeneration Completed Successfully- confirms that the ifs.war file is now considered to be deployed into the WebSphere environment.

## Update the Application Server Configuration

The CWS application server that was created in step 3 may optionally be updated to explicitly specify the location of the CWS configuration file and supported character set. By default, CWS will attempt to use the file /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc/config. To specify a different file you can set the Java system property **ifs.config.file** using the graphical administration client. To update the application server use the following instructions:

1. Open a WebSphere administrative console (see Appendix F)
2. In the left pane expand the *WebSphere Administrative Domain* item
3. In the left pane expand the *Nodes* item
4. In the left pane expand the <host> item where <host> is the hostname of the system
5. In the left pane expand the *Application Servers* item
6. In the left pane click on the <userid> item underneath *Application Servers*
7. In the right pane click on the *General* tab
8. Change the *Node startup state* combo box to *Stopped*
9. Click the *JVM Settings* tab
10. In the *System Properties* box click the *Add* button
11. **To Update Configuration File Location**

Two text fields are created, the value for the *Name* field is **ifs.config.file** and the value for the *Value* field is the path to the CWS configuration file. If the path that is entered begins with a / it is considered to be an absolute path; if the / is omitted the path is considered to be relative to the deployment directory /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc/config. Note that the etc/config is the addition to the absolute path.

### To Update Supported Character Set

Two text fields are created, the value for the *Name* field is **client.encoding.override** and the value for the *Value* field is the character set CWS will use (e.g. UTF-8, ISO-8859-1, etc.).

### To Update The Java Library Path

Two text fields are created, the value for the *Name* field is **java.library.path** and the value for the *Value* field is the path to the java servlets. If the path that is entered begins with a / it is considered to be an absolute

path; if the / is omitted the path is considered to be relative to the deployment directory /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/bin. Note that the bin is the addition to the absolute path

12. Click the *Apply* button.
13. On the event message of the lower pane, a ,command completed successfully- should appear.

## Change AIX Permissions

When WebSphere installs a WAR file it is exploded into the /usr/WebSphere/AppServer/installedApps directory. Each file contained in the WAR is copied to the file system with a default set of AIX permissions. The default permissions given to files must be changed for some of the tools and executables to work properly. To change the permissions of the CWS files use the following commands/instructions:

1. Logon as user `root`
2. Input the following commands:

```
# cd /usr/WebSphere/AppServer/installedApps/  
# chmod -R 755 <userid>.ear  
# chmod 777 <userid>.ear/ifs.war/WEB-INF/classes/com/ibm/ifs/resource  
# chmod 777 <userid>.ear/ifs.war/ebiller_logos  
# cd /usr/WebSphere/AppServer/temp  
# mkdir -p <hostname>/<userid>/<userid>/ifs.war  
# chmod -R 775 <hostname>/<userid>
```

*Where:*

*<userid> is the userid created in step 1*  
*<hostname> is the hostname of the system*

Note: The hostname of the system can be determined using the hostname command

```
# hostname
```

Note: The last command changes the permissions of the /usr/WebSphere/AppServer/temp/<hostname>/<userid> for members of the group to be able to write to the directory it is owned by. It is recommended to limit membership of said group to trusted users only.

## Initialize FISim Database (if applicable)

Skip this section if you are not using FISim. FISim stores some sample accounts in a DB2 database that can be used to test the CWS installation. FISim can be setup to read configuration information from shell environment variables or the command line. Use the following instructions to setup FISim.

1. Logon as user `<userid>`
2. Create a bin directory for the user with the following command:

```
$ mkdir ~/bin
```

3. Create a new file named cwsWas4 in the bin directory with the following lines (update for your environment as appropriate)

```
# Environment for CWS and WebSphere Application Server 4.0.2  
export USERID=${LOGIN}  
export WAS=/usr/WebSphere/AppServer  
export CWS=${WAS}/installedApps/${USERID}.ear/ifs.war
```

```

export JAVA_HOME=/usr/WebSphere/AppServer/java
export DB2DIR=/usr/lpp/db2_07_01
export DB2INSTANCE=db2inst1
export DB2INSTHOME=/home/db2inst1
export DB2INSTLIB=${DB2INSTHOME}/sqllib
export DB2JAVA=${DB2INSTLIB}/java

# Fisim and Mqseries queues
export FISQMGR=${USERID}qm
export FISDB2INST=${USERID}
export FISDB2USER=${USERID}
export FISXMIT=${USERID}.xmit
export FISIM_LOG_PATH=/home/${USERID}/CWS/logs

# Misc. environment variables
export PATH=${CWS}/tools:${WAS}/java/bin:${PATH}
export LIBPATH=/usr/lib:${JAVA_HOME}/lib:${CWS}/bin
export LD_LIBRARY_PATH=${LIBPATH}:${DB2INSTLIB}/lib
export CLASSPATH=${CWS}/WEB-INF/lib/jcwsapi.jar:${CWS}/WEB-INF/classes:${CWS}/bin:${WAS}/lib/j2ee.jar

# Add DB2 admin environment Fisim
. ${DB2INSTLIB}/db2profile

```

4. Save and exit the file.
5. Read the environment file using the following command:

```
$ . ~/bin/cwsWas4
```

6. Create the FISim database using the following commands (this step can be skipped if you are using a database that has previously been setup for FISim use):

```

$ db2 create db <userid>
$ db2 connect to <userid>
$ db2 create bufferpool <userid>pool size 1000 pagesize 16k
$ db2 disconnect current
$ db2stop (*Note: if db2 will not stop because other applications are currently accessing a database
you can use the ,db2 list applications- command to determine the active connections)
$ db2start
$ db2 connect to <userid>
$ db2 "create tablespace <userid>space pagesize 16k managed by system using
('<userid>space') bufferpool <userid>pool"
$ db2 disconnect current
$ db2stop
$ export EXTSHM=ON
$ db2set DB2ENVLIST=EXTSHM
$ db2start
$ cd /usr/fis/data (assuming Fisim install root is /usr/fis)
$ ./setupfis -d <userid> -u <userid> -p <password>

```

## Configure MQSeries

CWS and FISim use MQSeries queues to communicate. The following example uses the *All Local MQSeries Configuration* from Appendix D. to configure the MQSeries queues use the following commands/instructions:

1. Create MQSeries queue manager and queues using smitty by following the instruction in the *All Local MQSeries Configuration* in Appendix D or by executing the following commands as user <userid> at a shell prompt:

```
$ /usr/bin/crtmqm -c 'Queue manager for <userid> CWS instance' -u
'SYSTEM.DEAD.LETTER.QUEUE' -h '512' <userid>qm
$ /usr/bin/strmqm <userid>qm
$ /usr/lpp/mqm/smit/chamqattr "define" QLOCAL=='<userid>.xmit' DESCR=='FISim
transmission queue' USAGE='XMITQ' <userid>qm
$ /usr/lpp/mqm/smit/chamqattr "define" QLOCAL=='<userid>.reply' DESCR=='FISim
reply queue' <userid>qm
$ /usr/lpp/mqm/smit/chamqattr "define" QREMOTE=='<userid>'
RQMNAME=='<userid>' XMITQ=='<userid>.xmit' <userid>qm
```

## Update the CWS Configuration File

An example CWS configuration file is provided with the base ifs.war file but must be updated to reflect your environment. By default, CWS will attempt to read from the configuration file

/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc/config. You can specify a different configuration file by following the instructions in step 7.

To update the CWS configuration file use the following commands/instructions:

1. Logon as user root
2. Backup the example configuration file

```
# cd /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc
# cp config.sample config
```

3. Open the file ,config- for editing and change the following parameter values:

```
web_server_name=<userid>
wsp_exe_name=/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/bin/web
srvr
wsp_sharedmem_file=/home/<userid>/CWS/logs/<userid>.sharedmem
wsp_trace_file=/home/<userid>/CWS/logs/<userid>.trace
wsp_log_file=/home/<userid>/CWS/logs/<userid>.msglog
wsp_gold_trace=/home/<userid>/CWS/logs/<userid>.goldtrace
mq_request_qmgr=<userid>qm
mq_remote_qmgr_cc=<userid>
mq_reply_queue_cc=<userid>.reply
wsp_encrypt_file=/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc
/pubkey.cbsf
```

**Note:** These are the minimum set of parameters that must change in order for CWS to function properly with an All Local MQSeries configuration. Other parameters may be changes as needed in accordance with the environment in which CWS will be running. For example, you can change the web server type by changing the values of the following parameters:

### Web server types:

**WSP** To enable a WSP server, set wsp\_enable\_web=1. WSP cannot be enabled if CTW is enabled.

**CSR** To enable a CSR server, set wsp\_enable\_web\_server=1 and set Csr\_web\_server=1.

**CTW** To enable a CTW server, set wsp\_enable\_ctw=1. CTW cannot be enabled if WSP is enabled.

**SELFREG** To enable self registration, set selfreg\_server=1 if the server is to only be used as a self registration server or set selfreg\_server=2 if the server is to be used as both a self registration server and a WSP server.

More information about the available parameters is available in the sample CWS configuration file.



## Sample CWS Configuration File

A sample CWS configuration file is distributed with CWS. The sample provides information about the available mandatory and optional CWS configuration parameters. The default path to the file is `/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc/config.sample`.

## Starting and Stopping CWS

At this point CWS has been installed and configured. The preferred way to start and stop CWS is to use the WebSphere Advanced Administrative Console (see Appendix F). Use the following instructions to start and stop the newly created CWS instance:

### To Start CWS

1. Start WebSphere (see Appendix F)
2. Logon as user `<userid>`
3. Start the back end and the MQSeries queue manager. For instance, if you are using FISim:  

```
$ . ~/bin/cwsWas4 (source the environment file)
$ /usr/bin/strmqm <userid>qm (start the MQSeries queue manager)
$ cd /usr/fis/bin
$ ./runfis -d <userid> -u <userid> -p <password> -l & (start FISim)
```
4. Start CWS with the Administrative Console (see Appendix F), or with the `startweb` command:  

```
$ cd /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/tools
$ startweb
```

### To Stop CWS

Use the Administrative Console to stop the CWS instance, or use the `stopweb` command:

1. Logon as user `<userid>`
2. Execute the following commands:

```
$ . ~/bin/cwsWas4 (source the environment file)
$ cd /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/tools
$ stopweb
```

`ifs.war/tools/lswb`, `startweb`, and `stopweb` are tools for controlling the CWS server located in the `ifs.war/tools` directory under the web application instance directory. There are many command line options for selecting server instances, configuration files, runtime and HTTP server directories. Execute these commands with the `-H` option to view extended help and command line options.

The `startweb` and `stopweb` scripts use the current path to help determine which instance of CWS to start and stop. You should change to the tools directory of the cws instance before running `startweb` and `stopweb`.

## Test Environment Workaround

It is possible to install the base `ifs.war` file without updating it with a custom CWS application. The following is a workaround that allows the base `ifs.war` file to be installed by manually copying the example servlets into the `ifs.war` directory after deployment. This approach is not recommended, but may be used to set up a quick test environment. Use the instructions below to install and run the example Servlets after installing the base `ifs.war`:

1. Stop CWS

2. Logon as user root
3. Change into the application server s WEB-INF directory

```
# cd /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/WEB-INF
```

4. Open the file *ibm-web-ext.xmi* for editing
5. Change the property *serveServletsByClassnameEnabled=false* to *serveServletsByClassnameEnabled=true*
6. Save the file
7. Copy the test JSP files into the parent directory. Note that this step is done automatically by WebSphere when the ifs.war file is deployed.

```
# cp /V1R6M1/cws/testdocs/examples/*.jsp ..
```

Assuming the CWS tar file has been extracted into the /V1R6M1 directory.

8. Copy the test class files into the classes directory. Note that this step is done automatically by WebSphere when the ifs.war file is deployed.

```
# cp /V1R6M1/cws/testdocs/examples/*.class classes
```

Assuming the CWS tar file has been extracted into the /V1R6M1 directory.

9. Copy the images into the docs directory. Note that this step is done automatically by WebSphere when the ifs.war file is deployed.

```
# cp -R /V1R6M1/cws/docs/images ../docs
# cp -R /V1R6M1/cws/docs/ifs.gif ../docs
```

Assuming the CWS tar file has been extracted into the /V1R6M1 directory.

10. Start CWS
11. Connect with a web browser by opening the URL <http://server:port/servlet/Logon> where *server* is the hostname of the system and *port* is the same port specified in step 4.
12. You will be prompted to input a username, PIN and password, enter the following

```
Username:      ARF10nn (01 < nn < 10)
PIN:           10nn
Password:      10nn
```

nn must be the same for the username pin and password. Values shown are the default for Fisim.

After a successful test you should reverse the changes that you just made by stopping CWS, setting `serveServletsByClassName=false` in the *ibm-web-ext.xmi* file and removing the test files that were copied. Leaving the *ibm-web-ext.xmi* as is and/or not removing the test files will allow them to be accessed by other users and could be a potential security risk in a production environment.

---

## Other Considerations

### Running Multiple CWS Instances

Running multiple instances of CWS has been greatly simplified within the WebSphere 4 environment. No longer do you need to install multiple instances of an HTTP server or WebSphere. IHS and WebSphere 4

can be configured to handle multiple requests for different CWS instances on the same system by listening on different ports.

To install another instance of CWS on the same system follow the instructions above beginning with step 1 and continuing to step 14 with the caveat that no two CWS instances can run on the same port. That is, follow the same instructions that you did to create the first instance but create a different userid in step 1 and specify a different port in step 4.

The new instance does not have to be the same server type as the previous one. For example, the first installed CWS instance may have been configured as a CTW server and the second as a CSR server.

You can even deploy base ifs.war files that have been updated with different custom CWS applications. For example, you could have updated one base ifs.war file to include the user interface for one financial institution while the second has a user interface from a totally different financial institution.

## Important Ports

There are 4 ports of interest when running CWS on WebSphere 4, these ports must be unique on the same machine:

1. IHS administration server port - defined in

*/usr/HTTPServer/conf/admin.conf*

2. CWS instance port - defined in both

*/usr/HTTPServer/conf/httpd.conf*  
*/usr/WebSphere/AppServer/config/plugin-cfg.xml*

3. WebSphere administration server port - defined in

*/usr/WebSphere/AppServer/bin/admin.conf*

4. CWS instance WebSphere HTTP transport port - defined in

*/usr/WebSphere/AppServer/config/plugin-cfg.xml*

## Sample IHS Configuration File

One of the necessary steps when installing and configuring a CWS instance is updating the IHS configuration file to listen for requests on a specified port and creating virtual hosts. A sample IHS configuration file is distributed with CWS with the following path :

*/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/etc/httpd.conf.sample.*

The sample *httpd.conf* file provides example virtual hosts for two fictitious users: *user1* and *user2*. The virtual hosts illustrate how to configure the same CWS instance to listen for requests on two different ports; one configured for *http* and the other for *https*.

## Establishing A Secure HTTP Connection

In today's computing environment security is a very important consideration. CWS can be configured to listen for connections on secure ports using SSL (secure socket layers), better known as https. To accomplish this you must first configure IHS to use SSL; this process is thoroughly outlined in the IHS InfoCenter documentation that is bundled with IHS or can be found on-line at <http://www-4.ibm.com/software/webservers/htpservers/doc/v1319/index.html>. The InfoCenter documentation describes the process of enabling SSL using the IHS Administration Server GUI interface. To use the Administration Server GUI interface use the following steps:

1. Logon as user *root*

2. Create an IHS administration server userid and password

```
# /usr/HTTPServer/bin/htpasswd -m /usr/HTTPServer/conf/admin.passwd  
<login_name>
```

You will be prompted to create a password for this user.

3. Start the IHS administration server

```
# /usr/HTTPServer/bin/adminctl start
```

4. In a web browser load the Administration Server by going to the URL <http://server:8008>

Note that 8008 is the default port for the IHS administration server and *server* is the hostname of the system.


5. You will be prompted to enter a user id and password, use the one you created in step 2.

To enable a secure connection you must follow the ,how-to- in the InfoCenter documentation. To find the how-to use the following instructions:

1. Use a web browser to load the InfoCenter.
2. In the left pane click on the ,IBM HTTP Server- link to expand it.
3. In the left pane click on the ,How to- link to expand it.
4. In the left pane click on the ,Get started- link to expand it.
5. In the left pane click on the ,With secure connections- to load the how-to in the right pane.

Read through the documentation on certificates and follow the steps in the ,Setting up SSL using the IBM Administration Server- section.

Finally, you have to configure WebSphere to listen for requests on the secure port. To accomplish this you must update the Virtual Host you defined for the CWS instance using the following instructions:

1. Open a WebSphere administrative console (see Appendix F)
2. In the left pane click on the  next to the WebSphere *Administrative Domain* item.
3. In the left pane click on the *Virtual Hosts* item to highlight it.
4. In the top right pane click on the virtual host you created for the CWS instance requiring a secure connection.
5. In the middle right pane click on the *General* tab
6. In the middle right pane click on the *Add* button which creates a new text field under the *\*Host Aliases* label.
7. In the new text field enter the text *\*:<port>* where *<port>* is the port you assigned for the secure connection.
8. In the middle right pane click on the *Apply* button.
9. Stop and start the CWS application (See Appendix F)

## WebSphere Hints and Tips

Some of the basic WebSphere administrative topics are briefly discussed in Appendix F, such as starting and stopping WebSphere Administration Server and starting and stopping application servers. For more information about WebSphere administration and problem resolution please refer to the InfoCenter documentation that comes bundled with the product or can be found on-line at <http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.html> and the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*.

## WebSphere Topography

The installation and configuration process outlined above was very simple in the fact that each WebSphere instance had its own administrative domain and each system had a full installation of DB2 7.2.3. Both WebSphere and DB2 can be configured using different topologies for load balancing and centralization and backup of data. The appropriate choice for WebSphere and DB2 topographies is dependent upon many factors, including the hardware resources available to your organization, expected traffic, etc., and is beyond the scope of this document. More information about WebSphere and DB2 topologies can be found in chapter 5 of the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*.

## CWS Parameter Setups and Recommendations

The values for all *public static* variables in the **com.ibm.ifs.gold.IfsSystem** class can now optionally be set in the Java properties file **IfsSystem.properties**; which can be found in the `WEB-INF/classes` directory. `IfsSystem.java` should no longer be edited and recompiled, if variable values need to be changed the appropriate updates should be made to `IfsSystem.properties` and CWS restarted. Please read the instructions in the `IfsSystem.properties` file for more information.

## CWS Performance Recommendations

Parameter	Setting	Analysis
wsp_max_sessions	TBD	TBD
wsp_worker_threads	TBD	TBD
wsp_trace_flag	0	It is recommended that web server internal tracing not be done on production web servers unless needed for diagnostic purposes.
wsp_gold_trace_active	0	It is recommended that gold tracing not be done on production web servers unless needed for diagnostic purposes. It is recommended that development and test web servers trace all gold messages.
wsp_ping_timeout	TBD	TBD

---

## SNA Configuration

Refer to the SNA Installation manual for more information.

---

## Steps for Customizing SNA

For SNA configuration, the following three files should be modified or created.

- `/etc/sna/sna_dmn.cfg`
- `/etc/sna/sna_node.cfg`

- `/etc/sna/sna_tps.cfg`

Remember to be logged in as ,root- when you do this.

Whenever these files are edited, follow the steps below to restart SNA to make the new settings effective:

1. Enter SMIT by typing:  
`smitty SNA`
2. Select ,*Manage SNA Resources-*
  - Select ,*Stop SNA Resources-*
  - Select ,*Stop SNA-*
3. Return to the section ,*Manage SNA Resources-*
  - Select ,*Start SNA Resources-*
  - Select ,*Start SNA-*
  - Select ,*Start Node-*
  - Select ,*Start SNA Link Station-*

## Update SNA files

The following files need to be edited:

- `sna_dmn.cfg`
- `sna_node.cfg`
- `sna_tpse.cfg`

## MQ Configuration for a Test Environment

MQSeries is used to route messages between CWS and the core controller. The following MQ objects must be defined for production (see Figure 2):

- Local Queue Manager
- Local Transmit Queue to fictitious TDM
- Local Target Queue for CWS
- Remote Queue

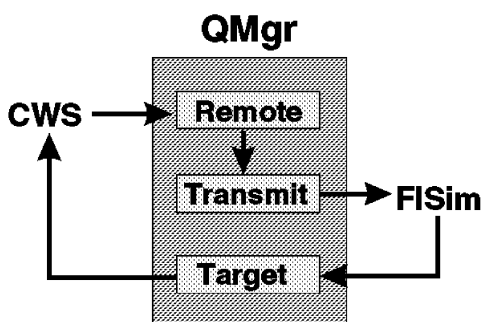


Figure 3. CWS with FISim emulation of both a TDM and an FI.

## Define and Start Queue Manager

- Set @queue manager name to `QM1`  
`crtmqm QM1`
- Start the queue manager  
`strmqm QM1`

## Local Transmission Queue

- Set @queue name to `rq.XMIT`
- Set @usage to `XMITQ`
- All other fields are default values.

```
runmqsc QM1
define QLOCAL ('rq.XMIT') +
USAGE (XMITQ)
```

## Local Target Queue

- Set @queue name to a.REPLY
- All other fields are default values.

```
runmqsc QM1
define QLOCAL ('a.REPLY')
```

## Remote Queue

**Note:** A remote queue is used by CWS to map a remote queue manager to a local transmit queue.

- Set @queue name to remote queue manager name.
- Set @name of remote queue manager to remote queue manager name.
- Set @transmission queue to the name of the transmission queue defined in ,Local Transmission Queue-.
- All other fields are default values.
- NOTE: QREMOTE and RQMNAME must be the same.

```
runmqsc QM1
define +
QREMOTE ('rq') +
RQMNAME ('rq') +
XMITQ ('rq.XMIT')
```

## MQ Configuration for a Production Environment

MQSeries is used to route messages from CWS to the core controller, logger and OLSS. The following MQ Objects must be defined for production (see Figure 3):

- Queue Manager
- Local Transmission Queue to TDM
- Process Definition
- Local Target Queue for CWS
- Local Target Queue for core controller, logger and IGATE
- Remote Queue
- Sender Channel
- Receiver Channel

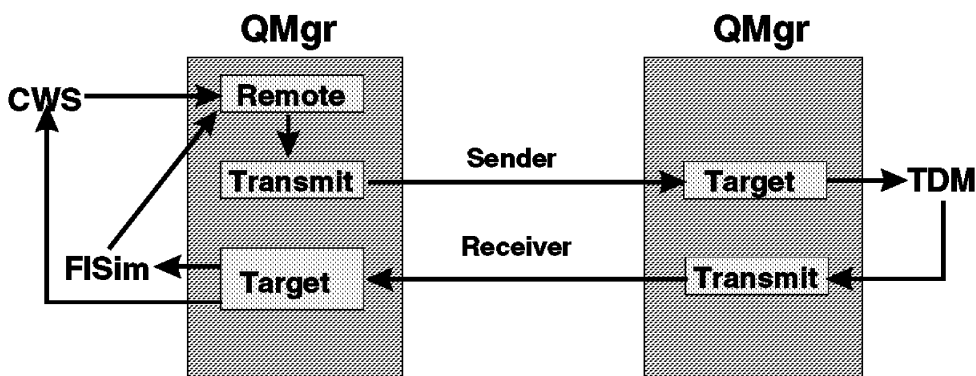


Figure 4. CWS and TDM with FISim emulation of an FI.

Figure shows two queue managers on AIX development system to facilitate comprehension of message flow between the IFS components. Only one queue manager on the AIX development system is required.

## Define and Start Queue Manager

- Set @queue manager name to QM1

*crtmqm QM1*

- Start the queue manager

*strmqm QM1*

## Local Transmission Queue to TDM

- Set @queue name to rq.XMIT
- Set @usage to XMITQ
- Set @trigger process to process definition defined in ,Process Definition-
- All other fields are default values.

*runmqsc QM1*

*define +*

*QLOCAL ('rq.XMIT') +*

*PROCESS ('rq.SEND.PROCESS')+*

*USAGE (XMITQ)*

## Process Definition

- Set @process name to rq.SEND.PROCESS
- Set @application type to UNIX
- Set @user data to sender channel name (QM1.TO.rq)
- All other fields are default values.

*runmqsc QM1*

*define process('rq.SEND.PROCESS') +*

*APPLTYPE (UNIX) +*

*APPLICID ('QM1.TO.rq')*

## Local Reply Queue for CWS

- Set @queue name to a.REPLY
- All other fields are default values.

*runmqsc QM1*

*define QLOCAL ('a.REPLY')*

## Remote Queue

**Note:** A Remote Queue is used by CWS to map a remote queue manager to a local transmit queue. QREMOTE and RQMNAME must be the same.

- Set @queue name to remote queue manager name.
- Set @name of remote queue manager to remote queue manager name.
- Set @transmission queue to the name of the transmission queue defined in ,Local Transmission Queue to TDM-
- All other fields are default values.

*runmqsc tQMdnmm*

*define +*

*QREMOTE ('rq') +*

*RQMNAME ('rq') +*

*XMITQ ('rq.XMIT')*

## Sender Channel

- Set @channel name to QM1.TO.rq
- Set @channel type to SDR
- Set connection name to ????
- Set @transmission protocol to LU62
- Set @name of transmission queue to the name of the transmission queue defined in ,Local Transmission Queue to TDM-
- All other fields are default values.

*runmqsc QM1*

*define +*

*CHANNEL ('QM1.TO.rq') +*



```
CHLTYPE (SDR) +  
CONNNAME ('????') +  
TRPTYPE (LU62) +  
XMITQ ('rq.XMIT')
```

## Receiver Channel

- Set @channel name to rq.TO.QM1
- Set @channel type to RCVR
- Set @transmission protocol to LU62
- All other fields are default values.

```
runmqsc QM1  
define +  
CHANNEL ('rq.TO.QM1') +  
CHLTYPE (RCVR) +  
TRPTYPE (LU62)
```

---

## Chapter 4. Writing Java Servlets for the IFS CWS

**Beta Warning:** This chapter contains code snippets that will not work with V1R6M1/WAS 4. See Appendix G for updates to jcwsapi.

---

### Before You Begin

Writing Java Servlets for the IFS CWS is not a tutorial for writing Java servlets. There are numerous books on the market for that purpose. This chapter specifically addresses the utilities in the IFS CWS Toolkit that were designed for use in writing Servlets to be used on the IFS platform. It is assumed that the developers will already be familiar with Java, Java Servlets and web development. It is also assumed that the developers will have read other IFS publications and will have some understanding of the IFS Customizable Web Server (CWS) architecture. One choice of software to use for Java Servlet development is [Beta TBD correct development tool is to be determined]. This, along with the String Data supplied in the IFS CWS Toolkit will enable the developer to perform preliminary unit testing of the Servlets developed. See , Unit Testing Page Servlets- on page 54 for more details.

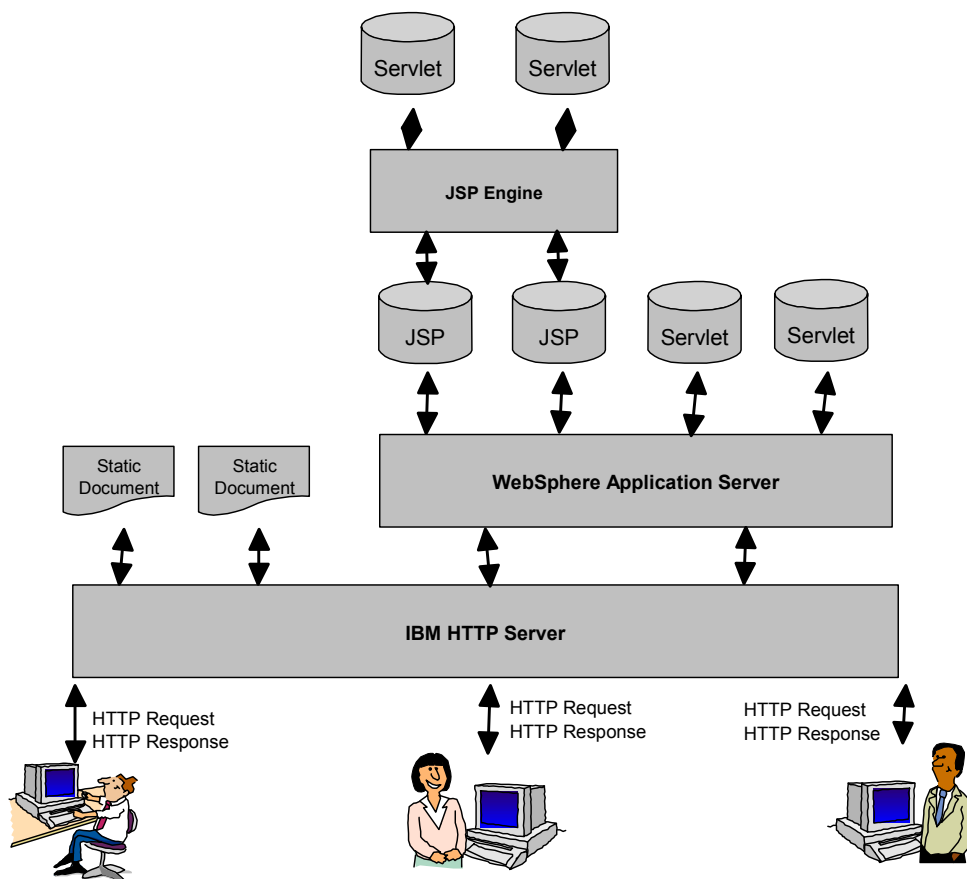


Figure 2. Java Servlet and JSP programming in IFS CWS

---

### Java Servlets

A servlet is a program written in the Java programming language that runs on the server, as opposed to the browser (applets). Java servlets provide web developers with a simple, consistent mechanism for extending the functionality of a web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side without a face. Servlets are designed to work within a request/response processing model. In such a model, a client sends a request message to a server and the server responds by sending back a response message. Java servlets have made many web applications possible. Although servlets are mostly used to generate dynamic HTML pages, they can be used to do many

other things, such as reading or writing information to/from a database, generating images using Java 2D API, parsing/generating XML documents, and sending serialized objects to and reading serialized objects from other applications.

Servlets are the Java platform technology of choice for extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building web-based applications, without the performance limitations of CGI programs. Servlets offer a number of advantages over other approaches, including portability, power, efficiency, extensibility, and elegance.

## Servlet Advantages

**Portability:** Because servlets are written in Java and conform to a well-defined and widely accepted API, they are server- and platform-independent. They are highly portable across operating systems and server implementations. With servlets, one can truly ,write once, run everywhere.-<sup>1</sup> This makes it possible to select a ,best of breed- strategy for servers, platforms, and tools.

**Power:** Written in Java, servlets can take advantage of Java s memory management and have access to the entire family of the Java APIs, including *networking and URL access, multithreading, image manipulation, data compression, database connectivity, internationalization, remote method invocation (RMI), and Enterprise JavaBeans (EJB), among others.*<sup>1</sup> Therefore, they enjoy all the benefits of the mature Java language - portability, performance, reusability, and crash protection.

**Efficiency:** Servlet invocation is highly efficient. Once a servlet is loaded, it generally remains in the server s memory as a single object instance. Thereafter, the server invokes the servlet to handle a request using a simple, lightweight method invocation. Separate threads handle multiple concurrent requests, so servlets are highly scalable.<sup>1</sup>

**Extensibility:** The Servlet API and servlet code are object oriented and so support encapsulation and inheritance. The Servlet API includes methods and classes to handle many routine chores of servlet development, such as request parameter parsing, cookie handling, and session tracking. The API is designed to be easily extensible. It can be extended and optimized by the financial institution using class inheritance. As a matter of fact, `IFSHttpServlet`, included in the IFS CWS Toolkit is extended from `HttpServlet` and has added a set of utility methods to handle common servlet functions, such as creating objects based on request parameters, caching objects, setting request parameters, page navigation, error handling, sort, etc. These utility methods will be discussed in detail in later sections.

## The Servlet API

The basic Servlet is not protocol-specific. However, because a large proportion of servlets are likely to use HTTP as the basis of their communication, the `HttpServlet` - a subclass of the Servlet class is provided to give additional support methods that are useful when handling the HTTP protocol. Focus is on `HttpServlet` and other HTTP related classes.

HTTP is a simple, stateless protocol. A client, such as a web browser, makes a request, the web server responds, and the transaction is done. The main behavior of the servlet is to respond to a request and perform the requested HTTP command, called a method, such as GET, POST, PUT, DELETE, etc.

The Servlet interface defines methods to initialize a servlet, to receive and respond to client requests, and to destroy a servlet and its resources. These are known as life-cycle methods, and are called by the web server in the following manner:

1. Servlet is created and then initialized via an `init()` method.
2. The `service()` method can be called zero or more times to handle client requests.
3. Servlet is destroyed via `destroy()` method.

---

<sup>1</sup> Java Servlet Programming by Jason Hunter with William Crawford. Copyright November 1998, Pages 11-12.

The `service()` method carries out a single request from the client. The method implements a request and response paradigm. It accepts two parameters: a request object and a response object. The request object contains information about the service request, including parameters provided by the client. The response object is used to return information to the client.

## HttpServletRequest Object

An `HttpServletRequest` object extends the `servlet` object. It encapsulates information about a single client request, including request parameters, implementation-specific attributes, and an input stream for reading binary data from the request body. It provides additional functionality specifically for HTTP servlets, including support for cookies, session tracking, access to HTTP header information, and parsing HTTP form data.

The following are some of the frequently used request methods.

```
javax.servlet.ServletRequest
{
    public abstract String      getParameter(String paramName);
    public abstract Enumeration getParameterNames();
    public abstract String []   getParameterValues(String paramName);
    public abstract String      getRemoteAddr();
}

javax.servlet.http.HttpServletRequest
{
    public abstract String      getRequestURI();
    public abstract Enumeration getHeaderNames();
    public abstract int         getIntHeader(String headerName);
    public abstract Cookie []   getCookies();
    public abstract HttpSession getSession(boolean creationFlag);
}
```

## HttpServletResponse Object

HttpServlet objects use `HttpServletResponse` objects to send MIME encoded data back to the client. The `ServletResponse` class is extended to allow manipulation of HTTP protocol-specific data, including response headers, status codes, setting cookies, sending redirects, etc. The servlet engine creates this object and passes it to the servlet's `service()` method. To send binary data, use the `ServletOutputStream` returned by `getOutputStream()`. To send character data, use the `PrintWriter` returned by `getWriter()`. The output's MIME type can be set using the `setContentType()` method.

The following are some of the frequently used request methods.

```
javax.servlet.ServletResponse
{
    public abstract ServletOutputStream getOutputStream() throws IOException;
    public abstract PrintWriter        getWriter()        throws IOException;
    public abstract void                setContentType(String type);
    public abstract void                setContentLength(int length);
}

javax.servlet.http.HttpServletResponse
{
    public abstract void addCookie(Cookie cookie);
    public abstract void setStatus(int statusCode);
    public abstract void sendError(int statusCode) throws IOException;
    public abstract void sendRedirect(String url)  throws IOException;
}
```

## Service Method

The `service()` method is the heart of the servlet. It handles the setup and dispatching to all the `doXXX()` methods. Therefore, a concrete subclass of `HttpServlet` usually does not override the `service` method. `IFSHttpServlet` extends the `HttpServlet` and overrides the `service()` method to provide some common service functions for concrete servlets, including setting the locale object for the session, initialization of the OID table, and authentication checking. An `HttpServlet` can override the `doGet()` and `doPost()` methods to handle GET and POST requests, respectively. These methods are where one can use the `HttpServletRequest` API

to process request parameters, use JDBC API to access the enterprise databases, and generate the dynamic HTTP page to send back to the client through HttpServletResponse API.

It is important to realize that there can be multiple service requests being processed at once. If your service method requires any outside resources, such as files, databases, or some external data, you must ensure that resource access is thread safe (to be thread safe don't create any class variables outside the doGet or doPost method).

## Example:

The following servlet generates a complete HTML page to say ,Hello World- and ,Current time is:... , where the time is the dynamic content.

```
import java.io.*;
import java.servlet.*;
import java.servlet.http.*;

public class HelloWorldServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Hello World</BIG>");
        out.println("<BIG>Current time is " + (new Date()).toString() + "</BIG>");
        out.println("</BODY></HTML>");
    }
}
```

---

## Using Cookies

A cookie is a named piece of data used for session tracking. A web server sends a cookie to a browser that can later be read back from that browser. Since HTTP connections are stateless, a cookie can be used to store persistent information across multiple HTTP connections.

To set cookie information, do the following:

- create a cookie which contains a name, value, and some other attributes
- set the content type of the HttpServletResponse response
- add the cookie to the response
- send the output.

The cookie must be added after setting the content type, but before sending the output, as the cookie is sent back as part of the HTTP response header.

```
Cookie theCookie = new Cookie ("ID", "123");
response.setContentType("text/html");
response.addCookie(theCookie);
```

It is necessary to remember that all cookie data are strings. Information such as int data must be converted to a String object. By default, the cookie lives for the life of the browser session. To enable a cookie to live longer, call the setMaxAge(interval) method. When positive, this allows the number of seconds a cookie exists to be set. A negative setting is the default and destroys the cookie when the browser exits. A zero setting immediately deletes the cookie.

Retrieving cookie data is a little awkward. It is not possible to ask for the cookie with a specific key. It is necessary to ask for all cookies, then find the specific one you are interested in.<sup>2</sup>

---

<sup>2</sup> <http://java.sun.com/products/servlet/index.html>

The following code finds the setting of a single-valued cookie:

```
Cookie theCookie = null;
Cookie cookies[] = request.getCookies();
if (cookies != null)
{
    for(int i=0, n=cookies.length; i < n; i++)
    {
        if (cookie[i].getName().equals("ID"))
        {
            // the required cookie is found
            theCookie = cookies[i];
            break;
        }
    }
}
```

The IFS system uses cookies to communicate between servlets and the IFS Server during a user session. The cookie can be provided by the servlet, or it will be automatically generated by the toolkit as part of transaction execution and saved in the HttpSession. During the execution of a transaction, the system will first look for the cookie from the HttpSession object using a name that matches the `IfsSystem.cwsCookieName`. If it finds the cookie it is used for communication. If a cookie of that name is not found in the session, the system will look for the cookie from the browser.

If this is the first transaction processed in a session, and the cookie is not found in either place, a new cookie with a default value is used. By IFS rules this first transaction must be either a logon or self-registration transaction. If this first transaction is processed successfully the system-generated cookie is stored in the HttpSession.

When the user session ends, the cookie is destroyed, thus requiring the user to logon again to establish a new session and cookie. If your business logic requires servlets to save the cookie in the browser after communications has been established, the cookie value must be reset before attempting to create a new user session. If not, the IFS system will attempt to use this old cookie to communicate with the server, and the server will fail because it will not know anything about the old cookie. To reset the cookie value, use the default cookie value of `NoCookie`.

```
if (ifsSession.getSessionKey() != "NoCookie")
{
    ifsSession.setSessionKey("NoCookie");
}
```

The cookie name used to communicate within the IFS system is defined in two places, the `IfsSystem` object and in the CWS config file. The `IfsSystem` object uses a static variable, `cwsCookieName`, to store the name of the cookie. The CWS config file uses a parameter called `cookie_name`. The default value for the cookie name is `IBANK`.

---

## Session Management

A session consists of a series of requests after authentication from the same browser over a fixed period of time. Cookies are usually used to identify a session and each cookie's size is limited to 4096 bytes. Therefore, in order to share application-specific data among requests, HTTP servlets allow one to maintain session information with the HttpSession class. *The HttpServletRequest provides the current session with the getSession(boolean) method. If the boolean parameter is true, a new session will be created when a new session is detected. This is, normally, the desired behavior. In the event the parameter is false, then the method returns null if a new session is detected.*<sup>2</sup>

This is an example of how to get the session:

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    HttpSession session = request.getSession(true);
    ...
}
```

Once an HttpSession has been created for a consumer, subsequent calls to `request.getSession(true)` will return the same HttpSession up until the session expires. If a parameter of `true` is used and the session

becomes invalid, a new one will be created. After accessing an `HttpSession`, it is possible to maintain a collection of key-value-paired information, for storing any sort of session-specific data. The developer automatically has access to the creation time of the session with `getCreationTime()` and the last accessed time with `getLastAccessedTime()`, which describes the time the last servlet request was sent for this session.

To store session-specific information, use the method:

```
public void putValue(String key, Object value)
```

To retrieve the information, ask the session using the method:

```
public Object getValue(String key)
```

**Note:** Both of these methods require, as inputs, an `HttpSession` object and an object identifier (OID). There is a public override of `putObject` that takes an `HttpSession` object.

When the consumer logs off the servlet should invalidate the session by using the method `invalidate()`.

`HttpServlet` provides an enhanced object caching and retrieving mechanism for session management that will be described later.

---

## JavaServer Pages (JSP)

The preferred way to generate dynamic HTML pages is to use JavaServer Pages (JSP). JSP pages are the pages created by the web developer that include JSP technology-specific tags, declarations, and possibly scriptlets, in combination with other static (HTML or XML) tags. A JSP technology-based page has the extension `.jsp`; this signals to the web server that the JSP technology-enabled engine will process elements on this page.<sup>2</sup> All JSP implementations support a Java programming language-based scripting language, which provides inherent scalability and support for complex operations.

JSP technology-based pages are compiled into servlets, so theoretically, the power of JSP pages to support web-based applications is equivalent to that of servlets. However, JSP technology was designed to simplify the process of creating pages by separating web presentation from web content. In many applications, the response sent to the client is a combination of template data and dynamically generated data. In this situation, it is much easier to work with JSP technology-based pages than to do everything with servlets.<sup>2</sup>

---

## Using HttpServlet

The IFS CWS toolkit provides an extension to the `HttpServlet` class called `HttpServlet`. The `HttpServlet` provides additional functions to the page servlet developer making it easier to create and initialize objects, store and retrieve objects in the http session, share objects between multiple web pages, sort objects, control page navigation, process IFS gold transactions errors, and debug page servlet code. Since `HttpServlet` is an abstract class, a concrete servlet class extends `HttpServlet` to take advantage of these functions. Example uses for most of the methods listed below can be found in the IFS toolkit example pages.

## Initialization

Each servlet page should consider the `HttpServlet` static boolean variable called `debugOn` and perform appropriate diagnostic messages when it is set to true.

Additionally, for the first rendered servlet page, at the beginning of the `doXXX` (`doGet`, `doPost`, etc.) method the first servlet page should call the `HttpSystem` `initHttp` static method and the `HttpServlet` `setDefaultHeader` method. The `initHttp` method performs important initialization action for the IFS Java framework. The `setDefaultHeader` sets the content type to `text/html` and the following parameters and values in the http header.

- Pragma to No-cache
- Cache-Control to no-cache
- Expires to 0

## Creating and Reusing Gold Objects within Multiple Servlet Pages

Gold objects can be created and processed from any servlet page. Once created, these objects can be shared with other servlet pages by utilization of the http session cache. References to objects within the http session cache are passed between the pages as http request parameters. The http request parameter names and values may be in either basic or compound formats. A basic format is one in which you have a single field name or single value such as `type=CDA`. A compound format is one in which a gold transaction or object is followed by a `,.-` then one of the fields of the object or transaction. If the field of the object or transaction is also an object, it can also be followed by a `,.-` and one of its fields. Any of the fields within this compound format may represent an array of the fields followed by a `,.-` and a positive integer value representing the element of the array. There are no limits to the number of `,.-` and fields that can be listed within a compound format. Again, both the name and value or either the name or value for an http request parameter can be in compound format.

In addition to the compound format, an http request parameter can contain a composite value. A composite value is one in which the value begins with `,??-` followed by two or more `name=value` pairs each separated by the `,&-`. Here is an example of an http request parameter with a compound name and a composite value: `,userProfile.accountDetail.0. account=??type=CDA&number=12345678&nickname=MyAccount-`.

Objects are placed in the session cache via the `putObject` method and retrieved from the cache with the `getObject` method. Both of these methods require, as inputs, an `IlsSession` object and an object identifier (OID). The OID is used to maintain uniqueness of objects within the session cache, and the `IlsSession` object identifies which http session. In addition, the `putObject` method requires the object to be stored. An OID can be automatically generated by the IFS system, or specifically provided by a page servlet developer. In either case, the OID must begin with two underscore characters, `,__-`. OIDs can be passed between servlet pages as http request parameters.

In addition to the `putObject` and `getObject` methods, objects can be both stored and retrieved from the http session cache with the `createInstance` method. The `createInstance` method requires three input parameters, an `IlsSession` object, a fully qualified object name, and an http request parameter name that will associate to an OID value. Method implementation consist of:

- Retrieve an OID value from the http request using the supplied http request parameter name
- If the http request parameter value is not an OID or it is a composite value (a value that begins with two question mark characters (??) followed by two or more `name=value` pairs )
  - Object is created using the fully qualified object name
  - Object is returned
- If the http request parameter value retrieved is an OID
  - Object is found in the http session cache using OID value

Retrieved object is returned

- Object is not found in the http session cache and the OID is not a compound type (an OID that contains `,.-` characters separating fields of the transaction the OID represents)

Object is created according to the fully qualified object name

Newly created object is stored in the http session cache using the OID value

Object is returned.

In each of these scenarios, the method returns an instance of an IFS gold object.

The benefits of using the `createInstance` method versus the combination of the `putObject/getObject` methods are twofold. First, in error free conditions, the `createInstance` method always returns an IFS object. The `getObject` method will only return an object when one is found in the http session cache. Second, if the servlet page that contains the `createInstance` method is loaded repeatedly, as could be the case with an impatient viewer, the object is not going to be created, discarded, and garbage collected over and over again. The gold object will be created and stored in the session cache the first time the page is loaded. Any repeated loading of this page retrieves the same object from the session cache.

In this same impatient viewer example, creating a gold object the conventional way using `new` and storing the object in the session cache with the `putObject` method creates two unwanted side effects. First a gold object will be repeatedly created and stored in the cache every time the page is loaded. Second, if the OID



value used to store the new object in the session cache is the same, the previous gold object will be marked for garbage collection. However, if a new OID is used each time the gold object is stored in the cache, each object will remain in the session cache even though only the last one saved is used.

## Examples:

Using the `putObject/getObject` combination of methods from a page a gold object can be created, and stored in the http session cache. For this illustration, the OID used to store the object in the cache will be a user-defined value. This OID value will be passed to the next page as an http request parameter.

Page 1:

```
IfsSession session = new IfsSession(request, response);
GtUserProfile userProfile = new GtUserProfile(session);
putObject(session, "__sharedUserProfile", userProfile);
response.sendRedirect("Page2?oid=__sharedUserProfile");
```

Page 2:

```
IfsSession session = new IfsSession(request, response);
String oidParm = (String) getRequestParameter(session, "oid");
GtUserProfile up = (GtUserProfile) getObject(session, oidParm);
```

The next example will use the `createInstance` method to create and store the gold object. Notice the object in this example is not created until the second page. Also, the OID used to store the object will be generated by the `IfsHttpServlet` `getNewOid` method within the first page and passed as an http request parameter. A form example will be used to pass the OID as a hidden field to Page 2.

Page 1:

```
IfsSession session = new IfsSession(request, response);
String oidValue = getNewOid(session, "GtUserProfile");
PrintWriter pw = response.getWriter();
pw.println("<FORM METHOD='post' ACTION='servlet/Page2'>");
pw.println("<INPUT TYPE='submit' NAME='test' VALUE='Test'>");
pw.println("<INPUT TYPE='hidden' NAME='oid' VALUE='" + oidValue + "'>");
pw.println("</FORM>");
```

Page 2:

```
IfsSession session = new IfsSession(request, response);
GtUserProfile up = null;
up = (GtUserProfile) createInstance(ifsSession, "com.ibm.ifs.gold.auto.GtUserProfile", "oid");
```

## Settings Fields within Gold Transaction Objects

Once a gold transaction object has been created and before it is executed, it may be necessary to set several of its fields with input data. The source of the input data for a transaction's fields can be a form from a browser window, or output fields from a previously executed gold transaction stored in the http session cache. In either case the data for the fields typically comes from a previous page. The `IfsHttpServlet` provides a method called `setFields` to facilitate the setting of a transaction's fields. This method attempts to match the name of each input field of the transaction with each name from the http request parameters. The method can drill down compound http request parameter names and values, as well as composite values, to set the appropriate fields of the specified transaction object. If a match is found, the value of the field is retrieved from the http request parameter and set into the transaction.

It is important to note here that when dealing with compound http request parameter names and values, it is only necessary to save the top object in the http session cache and not each individual field within the compound or composite formats. The `setFields` method is smart enough to traverse down the hierarchy starting with the top-level object.

## Examples:

This example will use simple field names from a form on Page 1. Page 2 will create the `GtLogon` transaction object and have its fields, `userID`, `pin`, and `passwordID` set from the previous page.

Page 1:

```
IfsSession session = new IfsSession(request, response);
String oidValue = getNewOid(session, "GtLogon");
PrintWriter pw = response.getWriter();
pw.println("<FORM METHOD='post' NAME='my_logon' action='servlet/Page2'>");
```

```

pw.println("<TABLE WIDTH='350' BORDER='0' CELLPADDING='0' HEIGHT='100'>");
pw.println("<TR>");
pw.println("<TD NOWRAP HEIGHT='27' WIDTH='27%'><B>Customer ID:</B></TD>");
pw.println("<TD>");
pw.println("<INPUT TYPE=text NAME=userID SIZE=30 VALUE=' '>");
pw.println("</TD>");
pw.println("</TR>");
pw.println("<TR>");
pw.println("<TD NOWRAP HEIGHT='27' WIDTH='27%'><B>PIN:</B></TD>");
pw.println("<TD>");
pw.println("<INPUT TYPE=password NAME=pin SIZE=30 VALUE=' '>");
pw.println("</TD>");
pw.println("</TR>");
pw.println("<TR>");
pw.println("<TD NOWRAP HEIGHT='27' WIDTH='27%'><B>Password:</B></TD>");
pw.println("<TD>");
pw.println("<INPUT TYPE=password NAME=passwordID SIZE=30 VALUE=' '>");
pw.println("</TD>");
pw.println("</TR>");
pw.println("</TABLE>");
pw.println("<INPUT TYPE=hidden NAME=logon VALUE=' ' + oidValue + ' '>");
pw.println("<P>");
pw.println("<INPUT TYPE='image' VALUE='logon' SRC='/images/enter.gif'");
pw.println("WIDTH='85' HEIGHT='25' ALIGN='CENTER'");
pw.println("BORDER='0'>");
pw.println("</P>");
pw.println("</FORM>");

```

Page 2:

```

IfsSession session = new IfsSession(request, response);
GtLogon myLogon = null;
myLogon = (GtLogon) createInstance(ifsSession, "com.ibm.ifs.gold.GtLogon", "logon");
setFields(session, "myLogon", myLogon);

```

This next example will demonstrate using hierarchical field names within Page 1 for a transaction that will be created in page 2. Page 2 will create a GtAccountDetail transaction and using setFields the transaction's account field will have its type, number, nickname, and accountSubType fields set from the previous page.

Page 1:

```

IfsSession session = new IfsSession(request, response);
GtUserProfile up = new GtUserProfile(session);
String oidValue = getNewOid(session, "GtAccountDetail");
up.execute();
PrintWriter pw = response.getWriter();
pw.println("<FORM METHOD='post' ACTION='/servlet/Page2'>");
pw.println("<INPUT TYPE='submit' NAME='test' VALUE='Test'>");
pw.println("<INPUT TYPE='hidden' NAME='acctDetail.account.type' VALUE=' ' +");
pw.println("up.getAccountDetail()[0].getAccount().getType() + ' '>");
pw.println("<INPUT TYPE='hidden' NAME='acctDetail.account.number' VALUE=' ' +");
pw.println("up.getAccountDetail()[0].getAccount().getNumber() + ' '>");
pw.println("<INPUT TYPE='hidden' NAME='acctDetail.account.nickname' VALUE=' ' +");
pw.println("up.getAccountDetail()[0].getAccount().getNickname() + ' '>");
pw.println("<INPUT TYPE='hidden' NAME='acctDetail.account.accountSubType' VALUE=' ' +");
pw.println("up.getAccountDetail()[0].getAccount().getAccountSubType() + ' '>");
pw.println("<INPUT TYPE=hidden NAME=accountDetail VALUE=' ' + oidValue + ' '>");
pw.println("</FORM>");

```

Page 2:

```

IfsSession session = new IfsSession(request, response);
GtAccountDetail acctDetail = null;
acctDetail = (GtAccountDetail) createInstance(ifsSession, "com.ibm.ifs.gold.auto.GtAccountDetail",
"accountDetail");
setFields(session, "acctDetail", acctDetail);

```

This last example will use basically the same data to demonstrate specifying hierarchical field names with composite values within Page 1 for a transaction that will be created in page 2. Page 2 will create a GtAccountDetail transaction and using setFields the transaction's account field will have its type, number, nickname, and accountSubType fields set from the previous page.

Page 1:

```

IfsSession session = new IfsSession(request, response);
GtUserProfile up = new GtUserProfile(session);
String oidValue = getNewOid(session, "GtAccountDetail");
up.execute();
PrintWriter pw = response.getWriter();

```

```

pw.println("<FORM METHOD='post' ACTION='/servlet/Page2'>");
pw.println("<INPUT TYPE='submit' NAME='test' VALUE='Test'>");
pw.println("<INPUT TYPE='hidden' NAME='acctDetail.account' VALUE='??type=" +
    up.getAccountDetail()[0].getAccount().getType() + "&number=" +
    up.getAccountDetail()[0].getAccount().getNumber() + "&nickname=" +
    up.getAccountDetail()[0].getAccount().getNickname() + "&accountSubType =" +
    up.getAccountDetail()[0].getAccount().getAccountSubType() + ">");
pw.println("<INPUT TYPE=hidden NAME=accountDetail VALUE=" + oidValue + ">");
pw.println("</FORM>");
Page 2:
IfsSession session = new IfsSession(request, response);
GtAccountDetail acctDetail = null;
acctDetail = (GtAccountDetail) createInstance(ifsSession, "com.ibm.ifs.gold.auto.GtAccountDetail",
"accountDetail");
setFields(session, "acctDetail", acctDetail);

```

## Removing Objects from the Http Session Cache

Currently, there are two ways to dispose of objects that have been put into the session cache. The simplest way is when the current session ends. When the http session ends, all of the objects within the cache are removed. The session can end for various reasons such as executing the GtLogoff transaction, an explicit call to `httpSession.invalidate()` or by experiencing a session timeout condition. A session timeout condition will occur when an http session is idle for some period of time. This period of time is programmable, and controlled via the `session.invalidationtime` parameter of the WebSphere `session.properties` file.

The controlled way to remove objects from the http session cache is via the `removeObject` method. Either the actual object to be removed or its OID is needed in order to remove the object from the cache.

### Example:

This example will remove the `GtUserProfile` from the session cache, using the actual object.

```

Any Page:
IfsSession session = new IfsSession(request, response);
GtUserProfile up = new GtUserProfile(session);
up.execute();
putObject(session, up);
.
.
.
removeObject(session, up);

```

## Handling Errors following Transaction Execution

After executing a transaction, its processing state must be checked to verify the contents of the returned data. The `handleErrors` method provides a convenient way to verify the state of a transaction and, if necessary, to forward processing control to a set of pages designed to display this information. If a transaction state is bad, the method redirects the browser to one of three locations specified with the call to `handleErrors`. However, if the transaction state is good the method just returns.

The `handleErrors` method requires as input, the transaction to check, the name of an error page servlet, the name of an information page servlet, and the name of a redo servlet. All of these pages are passed an http request parameter called `txn` and the value is the OID that can be used to retrieve the transaction from the http session cache. Warning here about redirection. Java servlets can be redirected using a method of the response object. However, if anything is written to the response writer before attempting a redirection, the redirection will fail. Therefore, it is wise to put all logic at the beginning of the servlet before any statements are written to the response object.

Most all executed transactions are returned a Business Response Code (BRC). Most BRCs fall into one of three major classifications, Fatal Error, Information, or Redo. An error condition indicates that some major problem occurred and processing should be rerouted to an error page where the error information can be displayed. Information conditions require a separate page for viewing the information returned before processing can continue. A redo condition typically means that a viewer entered incorrect information and reentry is required. Therefore, the redo page is usually the same page that executed the transaction. Error Handling will be discussed in some further detail in a later section.

## Example:

This example demonstrates a transaction execution and error processing.

```
Page 1:
IfsSession session = new IfsSession(request, response);
GtUserProfile up = new GtUserProfile(session);
up.execute();
handleErrors(up, "ErrorPage", "InfoPage", "Page 1");

//If processing returned here no problems were detected during execution of the transaction
ErrorPage:
IfsTxn txn = (IfsTxn) getRequestParameter(req, "txn");
if (null!=txn)
{
    pw.println("errorCode = " + txn.getErrorCode() + "<br>");
    pw.println("brc = " + txn.getBrc() + "<br>");
    pw.println("status = " + txn.getStatus() + "<br>");
    pw.println("compCode = " + txn.getCompCode() + "<br>");
    pw.println("reasonCode = " + txn.getReasonCode() + "<br>");
    pw.println("errorFunctionName = " + txn.getErrorFunctionName() + "<br>");
    pw.println("errorModuleName = " + txn.getErrorModuleName() + "<br>");
    pw.println("message = " + txn.getMessage() + "<br>");
    pw.println("gmMessage = " + txn.getGmMessage() + "<br>");
    pw.println("senseData = " + txn.getSenseData() + "<br>");
    pw.println("senseDataLength = " + txn.getSenseDataLength() + "<br>");
}
else
{
    String message = (String) getRequestParameter(req, "Exception");
    if (null!=message)
        pw.println("Exception occurred, text value is " + message);
}
```

## Controlling Page Navigation

There are times when page navigation needs to be restricted, as in locking the browser to a particular page until all required information is provided. The browser is locked to a page with the lockNavigation method. Once navigation is locked, the IfsHttpServlet prevents a consumer from going to any servlet page other than the locked page. If an attempt is made, the browser will automatically be rerouted back to the locked page. A page can be unlocked using the unlockNavigation method. This method returns the browser to normal page navigation. When navigation is locked to a particular page, the page developer must remember to unlock navigation from the locked page.

There are two additional helper methods concerning page navigation isNavigationLocked and getLockedPage. getLockedPage returns the name of the page that is currently locked.

## Example

This example locks navigation to a particular page, and unlocks navigation from the locked page.

```
Page 1:
IfsSession session = new IfsSession(request, response);
//Logon BRC indicated the consumer must change his password. Lock consumer to that page.
lockNavigation(session, "ChangePassword");
ChangePassword:
IfsSession session = new IfsSession(request, response);
GtChangePinPassword change = new GtChangePinPassword(session);

//Set fields to transaction
//Execute transaction
change.execute();
handleErrors(change, "ErrorPage", "InfoPage", "ChangePassword");

//Execution was successful, unlock navigation and continue
unlockNavigation(session);
```

## Debugging Page Servlets

Output of debug information within a page servlet is available via the debug method. Enabling the debug mode is via a static debug variable `debugOn` in each servlet page. Full enable/disable control and output location specification available through static variables in the `IfsSystem` class. Full enable of debug information means that all debug statements within all servlet pages and all transactions will be written to the output. Specific enable/disable control of servlet pages and gold transactions is available.

Debug enable options are defined in the `IfsSystem` class as `DEBUG_OFF`, `DEBUG_ON`, and `DEBUG_SOME_CLASSES`. Debug statements can be enabled/disabled via the static variable `IfsSystem.debugMode`. Debug statements can be sent to the browser window or to a file via static variables `IfsSystem.debugInfoToHtml` and `IfsSystem.debugInfoToFile`, respectively. The debug file name can be specified via the static method `IfsSystem.setDebugLogFileName`. The initial installation values are debug is set to off, output to html and file set to off, and the debug log file name is set to null. Specific enable of gold transactions and servlet pages requires setting the `IfsSystem.debugMode` to `IfsSystem.DEBUG_SOME_CLASSES` and then setting the static variable `debugOn` to true for each desired servlet page and gold transaction.

**WARNING:** If debug statements are sent to the browser window, and a debug statement is placed before a redirection command, the output of the debug statement to the browser window will cause the redirection to fail. A redirection command is executed within the `handleErrors` method if an error was detected from a processed transaction.

### Example:

This example enables debug statements, for all servlet pages and all gold transactions, sending them to a file called `/tmp/debug.log`.

```
public class Page1 extends IfsHttpServlet
{
    public static boolean debugOn = false;
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        IfsSystem.debugMode = IfsSystem.DEBUG_ON;
        IfsSystem.debugInfoToFile = true;
        IfsSystem.setDebugLogFileName("/tmp/debug.log");
        debug(res, "Inside Page1.doGet()");
    }
}
```

This example enables debug statements for the current servlet page and for the `GtAccountHistory` transaction. Statements will be sent to the browser.

```
public class Page1 extends IfsHttpServlet
{
    public static boolean debugOn = false;
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        IfsSystem.debugMode = IfsSystem.DEBUG_SOME_CLASSES;
        GtAccountHistory.debugOn = true;
        debugOn = true;
        IfsSystem.debugInfoToHtml = true;
        debug(res, "Inside Page1.doGet()");
    }
}
```

## Sorting Gold Objects by Fields

IFS gold objects within an array are sorted based on fields of the object as long as the field to sort on is one of the Java primitive types, a `String`, or one of the four gold primitives `GsDate`, `GsCurrency`, `GsDecimal`, or `GsTime`. All of the gold objects in an array must contain the name of the field that is to be used for sorting. Upon return from the sort method, the target array supplied as input contains all of the objects in the specified sorted order. Sorting can be specified for ascending or descending order. Three separate methods are provided for sorting Java primitives, gold primitives and `Strings`. If a gold primitive is used as the sort field, an `IComparator` object must be supplied as a sort parameter. An `IComparator` is nothing more than an instance of the gold primitive. For example, to sort an array of gold elements according to a field that is a currency field, the sort method requires a `GsCurrency` object instance to perform the sort.

## Example:

This example will sort an array of `GsAcctDtl` objects based on the Java `int` field of `fIProductDescriptionType` in ascending order.

```
public class Page1 extends IfsHttpServlet
{
    public static boolean debugOn = false;
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        GtUserProfile up = new GtUserProfile();
        up.execute();
        handleErrors(up, "ErrorPage", "InfoPage", "Page1");
        GsAcctDtl[] accountDetail = up.getAccountDetail();
        sort(accountDetail, "fIProductDescriptionType", true); //true means ascending
        //accountDetail is in ascending fIProductDescriptionType order.
    }
}
```

This example will sort an array of `GsAcctDtl` objects based on the gold primitive `GsCurrency` field of `ledgerBalanceAmount` in descending order.

```
public class Page1 extends IfsHttpServlet
{
    public static boolean debugOn = false;
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        GtUserProfile up = new GtUserProfile();
        up.execute();
        handleErrors(up, "ErrorPage", "InfoPage", "Page1");
        GsAcctDtl[] accountDetail = up.getAccountDetail();

        //set last parm to false to sort in descending order.
        sort(accountDetail, "ledgerBalanceAmount", GsCurrency.fromString("$1.00"), false);
        //accountDetail is in descending ledgerBalanceAmount order.
    }
}
```

This last example will sort an array of `GsAcctDtl` objects based on the `accountStatusType` field in descending order. The `accountStatusType` field is a `String`.

```
public class Page1 extends IfsHttpServlet
{
    public static boolean debugOn = false;
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        GtUserProfile up = new GtUserProfile();
        up.execute();
        handleErrors(up, "ErrorPage", "InfoPage", "Page1");
        GsAcctDtl[] accountDetail = up.getAccountDetail();

        // Create Collator object.
        Collator collator = Collator.getInstance();

        // set last parm to false to sort in descending order.
        sort(collator, accountDetail, "accountStatusType", false);
        //accountDetail is in descending accountStatusType order.
    }
}
```

---

## Using IFS Gold Objects

The IFS CWS toolkit provides transaction objects that are used to send and receive data between the servlet pages and an FI (Financial Institution). The transaction objects are built on a framework that provides customization methods, mechanism for defining additional fields for any transaction, and debug control at a transaction level. The framework supports a multitude of transactions tailored for most business needs. Even though there are many different types of transactions, the approach to using them is the same.

- Create transaction
- Set Required Fields
- Validate inputs
- Perform Request Ready Actions
- Execute

- Perform Response Ready Actions
- Error Processing
- Access returned data

Some transactions contain fields that require input before the FI can return any data. The specifics on which fields are required are set by the FI. The IFS framework does place a few restrictions on required input fields for the security transactions. These restrictions are discussed in the next section. Validation rules, request ready and response ready actions are handled via the customization methods.

A transaction's execute method provides the main behavior for the object. The data source for a transaction may be the FI or the CWS Gold Manager cache. The Gold Manager cache is a convenient place to store data that can be reused reducing the need to communicate again with the FI. A transaction can specify the source of the data via one of three flags. They are FI (default), CACHE, and CACHEFI. These static constants are defined in the `IlsTxn` object. The FI flag means to go to the FI for the transaction data. The CACHE flag means to return the transaction data from the Gold Manager cache. The CACHEFI flag means to attempt to return the transaction data from the Gold Manager cache, but if there is no current data within the cache for this transaction, retrieve the data from the FI. The actions performed during the execute method are:

- The cookie is retrieved from the session cache
- Validates input data via registered or default methods
- Creates/Populates GM Input Buffer with input data
- Performs Registered Request Ready Action
- Transfers GM Input Buffer to Gold Manager
- Performs error processing on returned data
- Parses GM Output Buffer into transaction fields
- Performs Registered Response Ready Action
- Cleans up GM Buffers

Following execution of a transaction, the `IlsHttpServlet` method `handleErrors` should be used to process any error information. For more information on error processing refer to the section titled, ,Error Handling- on 57.

---

## Required Inputs for Security Classes

The security classes are the only transactions that have required input fields. The security classes and their required fields are:

- `GtLogon`
  - `userID`
  - `pin`
  - `password`
- `GtChangePinPassword`
  - `oldPassword`
  - `oldPin`
  - `newPassword`
  - `newPin`
- `GtReauthentication`
  - `pin`
  - `password`

Some fields from the security classes have length restrictions. It is the responsibility of the servlets to provide validation. Methods of validation are discussed in the section ,Customization Methods- on page 48. The fields and max length values are:

- User ID - 30 bytes
- Encrypted data fields such as pin, password, consumer data - 117 bytes
- Country code - 3 bytes
- Language code - 3 bytes
- Product Name - 32 bytes
- Product Release Version - 12 bytes
- Client OS - 16 bytes
- Certificates - 4000 bytes

---

## Preventing Multiple Executions of a Transaction Instance

Before executing a transaction, the developer can set the execution mode. Three execution states are available NOT\_EXECUTE, EXECUTE, and EXECUTE\_SMART. All three of these values are defined as static variables in the lfsTxn object. The default execution state is EXECUTE. Provided there are no errors returned from input validation, or request ready actions, this state ensures that the transaction will be sent to the FI for processing. An execution mode value of NOT\_EXECUTE means the transaction will never be processed. The final execution mode available is the EXECUTE\_SMART. Again, if there are no validation or request ready action errors the transaction will be sent to the bank for processing **only** if it has not already been processed

Remembering the impatient viewer example discussed earlier, if the servlet page that has been double-clicked by the consumer contains a transaction execute statement, the transaction will be processed multiple times. However, setting the execution mode to EXECUTE\_SMART can prevent this type of problem.

---

## Self-Defining Fields

The ability to define, transmit and receive additional data fields within a transaction is known as self-defining fields. This enables a bank to pass additional information between a servlet page and the FI without having to create a new or extended transaction. Using self-defining fields, in lieu of, an extended or new transaction may be more desirable provided the number of fields to add are relatively small in number.

Each self-defining field must be defined using a GsGmSelfDefiningFieldDef object. Each GsGmSelfDefiningFieldDef object is placed in an array in the order the field is expected and the array is loaded into the transaction, using the setReqSelfDef method, for transmission to the FI. A transaction may contain either request, or response or both types of self-defining fields. A request self-defining field is a field that transmits additional data to the FI. A response self-defining field is a field that receives additional data from the FI. These types are defined in the lfsObject as lfsObject.SD\_RESPONSE and lfsObject.SD\_REQUEST. A transaction object owns Field definitions. Ownership values for a transaction can be obtained from a transaction via the getSD\_OWNER method.

The possible data types for self-defining fields are defined in the lfsObject as:

- SD\_TYPE\_STRUCT
- SD\_TYPE\_LONG
- SD\_TYPE\_DOUBLE
- SD\_TYPE\_CHAR
- SD\_TYPE\_BINARY
- SD\_TYPE\_CURRENCY
- SD\_TYPE\_DATE

Each field is identified with a TagID. The MTI defines pre-defined field TagIDs. The application developer defines self-defined field TagIDs. The starting level for self-defined field TagIDs is defined by the MTI.

Self-defining fields may be organized within self-defining structures. If so, each field must be chained to its self-defining parent structure using the parent TagID. The Chain TagID is a fixed format of 12 characters with padding of ,0- characters to the left if necessary. A self-defining structure may contain both pre-defined and self-defined fields.

Response self-defining data is retrieved from a transaction via the getRespSelfDefData method.

### Example:

This example defines a response self-defining field structure and its fields for an account history transaction that will return data as repeating items. Notice that the definition of the structures and its fields below is the same regardless of whether the data returned is a single instance or multiple instances as would be the case for a repeating structure.

In an actual program other fields of the GtAccountHistory transaction would have to be set before executing the transaction, but for clarity within this example, they have been omitted.



```

GtAccountHistory ah = new GtAccountHistory();
GsGmSelfDefiningFieldDef[] reqSD = new GsGmSelfDefiningFieldDef[3];
reqSD[0] = new GsGmSelfDefiningFieldDef();
reqSD[1] = new GsGmSelfDefiningFieldDef();
reqSD[2] = new GsGmSelfDefiningFieldDef();
reqSD[0].setFlags(IfsObject.SD_RESPONSE);
reqSD[0].setOwner(ah.getSD_OWNER());
reqSD[0].setDataType(IfsObject.SD_TYPE_STRUCT);
reqSD[0].setStructureInstanceCount(3);
reqSD[0].setPropertyName("ExampleRepeatStructure");
reqSD[0].setTagID(56001);
reqSD[1].setFlags(IfsObject.SD_RESPONSE);
reqSD[1].setOwner(ah.getSD_OWNER());
reqSD[1].setDataType(IfsObject.SD_TYPE_CURRENCY);
reqSD[1].setPropertyName("MyPaymentTg");
reqSD[1].setTagID(56002);
//Chain child field to parent structure using parents TagID. TagID in chain must be exactly
//12 characters in length. Pad with zero characters to the left.
reqSD[1].setTagIDChain("000000056001");
reqSD[2].setFlags(IfsObject.SD_RESPONSE);
reqSD[2].setOwner(ah.getSD_OWNER());
reqSD[2].setDataType(IfsObject.SD_TYPE_DATE);
reqSD[2].setPropertyName("MyDateTg");
reqSD[2].setTagID(56003);
reqSD[2].setTagIDChain("000000056001");
ah.setReqSelfDef(reqSD);
ah.execute();

//process the self-defining data returned
//For the example below, assume a PrintWriter was previously obtained as out.
GsGmSelfDefiningFieldDef[] respSD = ah.getRespSelfDefData();
//For each field within the structure, elements 1 and 2 of the structure, get the data returned
//Test to see if the data returned is a single instance or multiple
if (null==respSD[1].getMulti())
{
    //Data returned is a single instance, get the data from the individual field of the structure
    out.println("A payment was made on " + respSD[2].getSdDate().toString());
    out.println("The payment made was " + respSD[1].getSdCurrency().toString());
}
else
{
    //Data returned contains multiple instances
    GsGmSelfDefiningFieldData[] myPaymentData = respSD[1].getMulti();
    GsGmSelfDefiningFieldData[] myDateData = respSD[2].getMulti();
    for (int i = 0; i < myPaymentData.length; i++)
    {
        //use length from either data struct, the lengths of each must be the same
        out.println("A total of " + Integer.toString(myPaymentData.length) + " have been made:");
        out.println("Payment " + Integer.toString(i + 1) + " for " +
            myPaymentData[i].getSdCurrency().toString() + " was made on " +
            myDateData[i].getSdDate().toString() + ".");
    }
}

```

The last example will define two request self-defining fields within a self-defining field structure for the logon transaction. In an actual program other fields of the GtLogon transaction would have to be set before executing the transaction, but for clarity within this example, they will be omitted.

```

GtLogon logon = new GtLogon()
GsGmSelfDefiningFieldDef[] sarray = new GsGmSelfDefiningFieldDef[3];
sarray[0] = new GsGmSelfDefiningFieldDef();
sarray[1] = new GsGmSelfDefiningFieldDef();
sarray[2] = new GsGmSelfDefiningFieldDef();

sarray[0].setFlags(IfsObject.SD_REQUEST);
sarray[0].setTagID(56001);
sarray[0].setOwner(myLogon.getSD_OWNER());
sarray[0].setDataType(IfsObject.SD_TYPE_STRUCT);
sarray[0].setPropertyName("Add1LogonInfoTg");
sarray[0].setStructureInstanceCount(3);
sarray[1].setFlags(IfsObject.SD_REQUEST);
sarray[1].setTagID(56002);
sarray[1].setOwner(myLogon.getSD_OWNER());

```

```

sdarray[1].setDataType(IfsObject.SD_TYPE_LONG);
sdarray[1].setPropertyname("TermsAndCondsFlgTg");
sdarray[1].setSdLong(1);
sdarray[1].setTagIDChain("000000056001");

sdarray[2].setFlags(myLogon.SD_REQUEST);
sdarray[2].setTagID(56003);
sdarray[2].setOwner(myLogon.getSD_OWNER());
sdarray[2].setDataType(IfsObject.SD_TYPE_CHAR);
sdarray[2].setPropertyname("HshldIdTg");
sdarray[2].setLength(44);
sdarray[2].setSdChar("10 Downing Street, London England");
sdarray[2].setTagIDChain("000000056001");
logon.setReqSelfDef(sdarray);
logon.execute();

```

---

## Customization Methods

The customization methods can be used to specify unique banking rules and policies for any given transaction. Validate, request ready action, response ready action, toString, and fromString are the type of customized methods that can be provided for a transaction. Customized methods for transactions must be registered with each specific class using the registry methods. The following registry methods are defined in the IfsObject class:

- registerMethod
- deregisterMethod
- hasRegisteredMethod

Every customized method returns a results string. If the value of the results string is ,OK-, excluding the quotes, then the method is assumed to have completed successfully. Any other value is assumed a failure and the results string is copied to the message field of the transaction. After registering a validate, request ready or response ready action for a transaction class, specific instances of that class can enable the methods via setValidateActionRequired(true), setRequestReadyActionRequired(true), and setResponseReadyActionRequired(true) respectively.

The IFS security classes GtLogon, GtChangePinPassword, and GtReauthentication come with default validation methods that check to ensure that field values are provided. If any one of these values is not provided, the default validation routines will fail. Registering customized validation rules for classes overrides the default validation behavior.

The GtLogon default validation routine checks for any data values for the user id, pin, and password fields. No lengths or data content is validated.

The GtChangePinPassword default validation routine checks for any data values for the old and new user id, old and new pin, and old and new password fields. No lengths or data content is validated.

The GtReauthentication default validation routine checks for any data values for the pin and password fields. No lengths or data content is validated.

### Example:

The following example registers validation and response ready action methods for the GtAccountHistory class. This specific response ready action builds response self-defining data for the transaction. This will enable a bank to unit test servlet pages requiring self-defining data without the Gold manager and a FI.

```

public class Page1 extends IfsHttpServlet
{
    public static boolean debugOn      = false;
    private static boolean initialized = false;

    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        PrintWriter pw = res.getWriter();
        setDefaultHeader(res);
        IfsSystem.initIfs();
    }
}

```

```

IfsSession ifsSession = new IfsSession(req,res);
GtAccountHistory ah = new GtAccountHistory(ifsSession);
GtUserProfile up = null;

if (!initialized)
{
    try
    {
        debug(res, "Creating the registration class for BankCustomizedMethods");
        Class theClass = Class.forName("BankCustomizedMethods");
        debug(res, "Created the registration class for BankCustomizedMethods");
        //Register the Response Ready Action method for GtAccountHistory transaction
        debug(res, "Registering responseReadyAction for GtAccoutHistory class");
        ah.registerMethod("responseReadyAction", theClass.getMethod("responseReadyAction",
            new Class[] {GtAccountHistory.class}));
        debug(res, "Registered responseReadyAction for GtAccoutHistory class");
        debug(res, "Registering validate for GtAccoutHistory class");
        ah.registerMethod("validate", theClass.getMethod("validate",
            new Class[] {GtAccountHistory.class}));
        debug(res, "Registered validate for GtAccoutHistory class");
    }
    catch(Exception e)
    {
        debug(res, "Exception in BankCustomizedMethods registration, value is " +
            e.toString());
    }
    initialized = true;
}

try
{
    up = (GtUserProfile)createInstance(ifsSession, "com.ibm.ifs.gold.auto.GtUserProfile",
        "__UserProfile");
    //Use Execute_Smart flag so that if the instance of user profile received came
    //from the cache and has already been executed, it will not be resent to the FI.
    debug(res, "Getting ready to set Execution mode");
    up.setExecutionMode(IfsTxn.EXECUTE_SMART);
    debug(res, "Getting ready to set execute up");
    up.execute();
    ah.setReqAccount(up.getAccountDetail()[0].getAccount());
    ah.setReqMaximumNumberOfRecords(10);
    ah.setReqTransactionType("1");
    GsGmSelfDefiningFieldDef[]reqSD = new GsGmSelfDefiningFieldDef[5];
    reqSD[0] = new GsGmSelfDefiningFieldDef();
    reqSD[1] = GsCurrency.asSelfDefiningFieldDef();
    reqSD[2] = GsCurrency.asSelfDefiningFieldDef();
    reqSD[3] = GsDate.asSelfDefiningFieldDef();
    reqSD[4] = GsDate.asSelfDefiningFieldDef();
    reqSD[0].setFlags(IfsObject.SD_RESPONSE);
    reqSD[0].setOwner(ah.getSD_OWNER());
    reqSD[0].setDataType(IfsObject.SD_TYPE_STRUCT);
    reqSD[0].setStructureInstanceCount(5);
    reqSD[0].setPropertyName("CCDt1Tg");
    reqSD[0].setTagID(56004);
    reqSD[1].setFlags(IfsObject.SD_RESPONSE);
    reqSD[1].setOwner(ah.getSD_OWNER());
    reqSD[1].setDataType(IfsObject.SD_TYPE_CURRENCY);
    reqSD[1].setPropertyName("MinPymtAmtTg");
    reqSD[1].setTagID(130);
    reqSD[1].setTagIDChain("000000056004");
    reqSD[2].setFlags(IfsObject.SD_RESPONSE);
    reqSD[2].setOwner(ah.getSD_OWNER());
    reqSD[2].setDataType(IfsObject.SD_TYPE_CURRENCY);
    reqSD[2].setPropertyName("LastStmtBalAmtTg");
    reqSD[2].setTagID(425);
    reqSD[2].setTagIDChain("000000056004");
    reqSD[3].setFlags(IfsObject.SD_RESPONSE);
    reqSD[3].setOwner(ah.getSD_OWNER());
    reqSD[3].setDataType(IfsObject.SD_TYPE_DATE);
    reqSD[3].setPropertyName("StmtEndDtTg");
    reqSD[3].setTagID(63);
    reqSD[3].setTagIDChain("000000056004");
    reqSD[4].setFlags(IfsObject.SD_RESPONSE);

```

```

reqSD[4].setOwner(ah.getSD_OWNER());
reqSD[4].setDataType(IfsObject.SD_TYPE_DATE);
reqSD[4].setPropertyNames("PymtDueDtTg");
reqSD[4].setTagID(134);
reqSD[4].setTagIDChain("000000056004");
ah.setReqSelfDef(reqSD);
ah.setResponseReadyActionRequired(true);
ah.setValidateActionRequired(true);
debug(res, "<BR>Executing the Account History transaction");
ah.execute();
debug(res, "<BR>Executed the Account History transaction");
}
catch(Exception e)
{
    debug(res, "Exception occurred, value is " + e.toString());
}
pw.println("<HTML>");
pw.println("<HEAD>");
addCopyright(res);

pw.println("<TITLE>");
pw.println("Bank Unit Test Servlet");
pw.println("</TITLE>");

pw.println("</HEAD>");
GsGmSelfDefiningFieldDef[] respSD = ah.getRespSelfDefData();
if (null==respSD)
    pw.println("<BR><BR>respSD is null<BR>");
else
{
    pw.println("<BR><BR>respSD is not null<BR>");
    pw.println("<BR><BR>The number of response fields is: " +
        Integer.toString(respSD.length));
    if (respSD[1].getSdCurrency()!=null)
    {
        pw.println("<BR><BR>The Self Defining Field Name that was retrieved is: " +
            respSD[1].getPropertyNames() + "<BR>");
        pw.println("<BR><BR>The value for that field is: " +
            respSD[1].getSdCurrency().toString("\n") + "<BR>");
    }
    if (respSD[2].getSdCurrency()!=null)
    {
        pw.println("<BR><BR>The Self Defining Field Name that was retrieved is: " +
            respSD[2].getPropertyNames() + "<BR>");
        pw.println("<BR><BR>The value for that field is: " +
            respSD[2].getSdCurrency().toString("\n") + "<BR>");
    }
    if (respSD[3].getSdDate()!=null)
    {
        pw.println("<BR><BR>The Self Defining Field Name that was retrieved is: " +
            respSD[3].getPropertyNames() + "<BR>");
        pw.println("<BR><BR>The value for that field is: " +
            respSD[3].getSdDate().toString("\n") + "<BR>");
    }
    if (respSD[4].getSdDate()!=null)
    {
        pw.println("<BR><BR>The Self Defining Field Name that was retrieved is: " +
            respSD[4].getPropertyNames() + "<BR>");
        pw.println("<BR><BR>The value for that field is: " +
            respSD[4].getSdDate().toString("\n") + "<BR>");
    }
}
pw.println("</BODY>");
pw.println("</HTML>");
}

```

```

public class BankCustomizedMethods
{
    public static String responseReadyAction(GtAccountHistory ah)
    {
        try
        {
            ah.debug("Creating GsCurrency for $100.66 and storing as first element " +

```

```

        "of self-defining array");
        ah.getRespSelfDefData()[1].setSdCurrency(GsCurrency.fromString("$100.66"));
        ah.debug("Creating GsCurrency for -$456.79 and storing as second element " +
            "of self-defining array");
        ah.getRespSelfDefData()[2].setSdCurrency(GsCurrency.fromString("-$456.79"));
        ah.debug("Creating GsDate for 09/24/1999 and storing as third element " +
            "of self-defining array");
        ah.getRespSelfDefData()[3].setSdDate(GsDate.fromString("09/24/1999"));
        ah.debug("Creating GsDate for 01/01/2000 and storing as fourth element " +
            "of self-defining array");
        ah.getRespSelfDefData()[4].setSdDate(GsDate.fromString("01/01/2000"));
        return "OK";
    }
    catch(Exception e)
    {
        ah.debug("Exception in BankCustomizedMethods responseReadyAction, value is " +
            e.toString());
        return ("Exception in BankCustomizedMethods responseReadyAction, value is " +
            e.toString());
    }
}

public static String validate(GtAccountHistory ah)
{
    try
    {
        ah.debug("Performing validation for GtAccountHistory");
        //Specific rules go here
        if (goodStatus)
            return "OK";
        else
            return "Failure Message String";
    }
    catch(Exception e)
    {
        ah.debug("Exception in BankCustomizedMethods validate, value is " + e.toString());
        return ("Exception in BankCustomizedMethods validate, value is " + e.toString());
    }
}
}
}

```

---

## Debugging Transactions

The debug control for IFS transaction objects is similar in nature to the control provided by the `IlsHttpServlet`. Refer to that section for further details on how to enable and direct statement output.

The additional information provided at the transaction level is a dump of both the input and output buffers of a transaction, and the detailed information concerning transaction parsing of the output buffer into individual fields.

### Example:

```

Thu Nov 11 11:48:04 EST 1999 <BR>Entry to executeTxn.
Thu Nov 11 11:48:04 EST 1999 <BR>Executing transaction of name com.ibm.ifs.gold.GtLogon
Thu Nov 11 11:48:04 EST 1999 <BR>Before initGMJI().
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: Done with signature GMJI
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: Done putting two spaces for number of bytes and
arguments
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: The httpServletRequest is not null
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: Done putting the ServletPath
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: The cookie is null
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: Done with get_cookies()
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: We got into the for loop
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: We got into the for loop
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: We got into the for loop
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: Done with putting getRemoteAddr
Thu Nov 11 11:48:04 EST 1999 <BR>Calling addGoldToGMJI().
Thu Nov 11 11:48:04 EST 1999 <BR>Calling invokeGoldManage().
Thu Nov 11 11:48:04 EST 1999
***** GMJI *****

```

```

Size of gmji buffer is 4096 bytes.
There are 218 bytes used in the buffer.
0000 474D4A49 000000DA 0000000E 00000004 [GMJI.....]
0010 00000013 2F6A6176 612F5072 6F636573 [....java/Proces]
0020 734C6F67 6F6E0000 00000400 00000F49 [sLogon.....I]
0030 42414E4B 3D4E6F43 6F6F6B69 65000000 [BANK=*****]
0040 00040000 000A392E 31342E33 2E363700 [.....9.14.3.67.]
0050 00000004 00000002 64000000 00040000 [.....d.....]
0060 00034745 00000000 04000000 02300000 [..GE.....0..]
0070 00000400 00000347 45000000 00040000 [.....GE.....]
0080 00023000 00000004 00000003 55530000 [...0.....US..]
0090 00000400 00000365 6E000000 00040000 [.....en.....]
00A0 00094E65 74736361 70650000 00000400 [..Netscape.....]
00B0 00000434 2E360000 00000400 00000657 [...4.6.....W]
00C0 696E4E54 00000000 04000000 0D303030 [inNT.....000]
00D0 30303030 30303030 3000          [000000000.]
***** END OF GMJI *****

```

```

Thu Nov 11 11:48:04 EST 1999
IfsTxn:invokeGoldManager calling gmJavaReq native interface. $$$
Thu Nov 11 11:48:04 EST 1999
LogoffTxn = false
Thu Nov 11 11:48:04 EST 1999 <BR>DEBUG: We are calling the gmJavaLogonReq
Thu Nov 11 11:48:04 EST 1999
IfsTxn:invokeGoldManager return from gmJavaReq native interface.
Thu Nov 11 11:48:05 EST 1999
***** GMJO *****

```

```

Size of gmjo buffer is 5388 bytes.
0000 474D4F42 30303030 30303030 30303030 [GMOB000000000000]
0010 30303030 30303161 31303030 30303030 [0000001a10000000]
0020 30303030 31613035 2D313436 38393531 [00001a05-1468951]
0030 35202020 20202020 20202020 20202020 [5          ]
0040 20202020 38362D36 37313832 31383520 [ 86-67182185 ]
0050 20202020 20202020 20202020 20202020 [          ]
0060 20203331 2D343730 39353333 38202020 [ 31-47095338 ]
0070 20202020 20202020 20202020 20202020 [          ]
0080 20202020 20202020 20202020 20202020 [          ]
0090 20202020 20202020 2020204A 6F6E6573 [      Jones]
00A0 20202020 20202020 20202020 20202020 [          ]
00B0 20202020 20202020 20467265 64202020 [      Fred ]
00C0 20202020 20202020 20202020 20202020 [          ]
00D0 20202020 20202061 68666277 70202020 [      ahfbwp ]
00E0 20202020 20202020 20202061 20203030 [      a 00]
00F0 30303030 30303030 30316135 39332D39 [0000000001a593-9]
0100 32362D36 30333320 20206163 66797477 [26-6033 acfytw]
0110 20206167 67303030 30303030 30303030 [ agg000000000000]
0120 31613639 34322052 69766572 20437420 [1a6942 River Ct ]
0130 20202020 20202020 20202020 20202020 [          ]
0140 20202020 20202020 20202020 20202020 [          ]
0150 20202020 20202020 20202020 20202020 [          ]
.
.
.
1310 20202020 30303030 30303030 30303032 [ 000000000002]
1320 30303030 30303030 30303031 30303030 [0000000000010000]
1330 30303030 30303031 61555344 39383431 [00000001aUSD9841]
1340 36352020 20202020 20202030 30303030 [65      00000]
1350 30303030 30303230 30303030 30303030 [0000002000000000]
1360 30303030 30303030 30303030 30303161 [000000000000001a]
1370 55534431 20303030 30303030 30303030 [USD1 000000000000]
1380 31613030 30303030 30303030 30323030 [1a00000000000200]
1390 30303030 30303030 3030436F 72656261 [0000000000Coreba]
13A0 6E6B3030 30303030 30303030 30316130 [nk000000000001a0]
13B0 30303030 30303030 30303142 696C6C20 [000000000001Bi11 ]
13C0 50617930 30303030 30303030 30303161 [Pay000000000001a]
13D0 61626368 7A6A7677 687A6262 7A736873 [abchzjvwzhbbzshs]
13E0 79202020 20202020 20202020 20202020 [y          ]
13F0 20202020 20202020 30303036 34393031 [ 00064901]
1400 33373532 61613030 30303030 30303030 [3752aa0000000000]
1410 30316130 30303030 30303030 30303230 [01a0000000000020]
1420 30303030 30303030 30303030 30303233 [000000000000023]
1430 33343735 35353630 30303030 30303030 [3475556000000000]

```

```

1440 30303161 30303030 30303030 30303031 [001a0000000000001]
1450 30303034 31383535 39313731 30303030 [0004185591710000]
1460 30303030 30303031 61617666 78632020 [00000001aavfxc ]
1470 20613030 30303030 30303030 30316161 [ a000000000001aa]
1480 6D70626C 6C746E72 70686368 6E782020 [mpb11tnrphchnx ]
1490 20202020 20202020 20202020 20202020 [ ]
14A0 20202020 20202020 20202020 20202020 [ ]
14B0 20202020 20202020 20202020 20202020 [ ]
14C0 20202020 20202020 20202020 20202030 [ 0]
14D0 30303030 30303030 30303549 42414E4B [000000000005IBANK]
14E0 30303030 30303030 30303332 30303030 [0000000000320000]
14F0 30303030 45424330 33454144 30303030 [0000EBC03EAD0000]
1500 30303031 45433146 41353231 [0001EC1FA521]
***** END OF GMJO *****
Thu Nov 11 11:48:05 EST 1999 <BR>Calling setErrorStatus().
Thu Nov 11 11:48:05 EST 1999 <BR>In IfsTxn setErrorStatus()
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:0, Name:GMOB_Acronym, Len:4, Value=GMOB.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:4, Name:GMOB_Return_Code, Len:4,
Value=0000.
Thu Nov 11 11:48:05 EST 1999 <BR>setErrorStatus() BRC
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:8, Name:brc, Len:3, Value=000.
Thu Nov 11 11:48:05 EST 1999 <BR>After setErrorStatus().
Thu Nov 11 11:48:05 EST 1999 <BR>Calling getGoldFromGMJO()
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:8, Name:responseCode, Len:3, Value=000.
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:11, Name:responseCode_text, Value=1.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:23, Name:responseCode_text, Len:1,
Value=a.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:24, Name:fIProcInd, Len:1, Value=1.
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:25, Name:fIProcInd_text, Value=1.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:37, Name:fIProcInd_text, Len:1, Value=a.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:38, Name:consumerID, Len:30, Value=05-
146895
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:68, Name:changeableConsumerID, Len:30,
Value=86-67182185.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:98, Name:permUserID, Len:57, Value=31-
47095338.
.
.
.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5175, Name:cnsmrInfoTx, Value=1.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5187, Name:cnsmrInfoTx, Len:1, Value=a.
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5188, Name:GoldNdx, Value=1.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5200, Name:cnsmrInfoCl, Value=418559171.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5212, Name:cnsmrInfoTx, Value=1.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5224, Name:cnsmrInfoTx, Len:1, Value=a.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5225, Name:queueQualifier, Len:8,
Value=avfxc.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5233, Name:sLASampleFlag, Len:1, Value=a.
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5234, Name:sLASampleFlag_text, Value=1.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5246, Name:sLASampleFlag_text, Len:1,
Value=a.
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5247, Name:tokenInformation, Len:80,
Value=ampb11tnrphchnx.
Thu Nov 11 11:48:05 EST 1999 <DEBUG> in getNextCookieFromGMJO
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5327, Name:cookieName, Value=5.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 <DEBUG> after getting cookieNameLength
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5339, Name:cookieName, Len:5, Value=IBANK.
Thu Nov 11 11:48:05 EST 1999 <DEBUG> after getting cookieName
Thu Nov 11 11:48:05 EST 1999 Extracting int, Offset:5344, Name:cookieValue, Value=32.
Thu Nov 11 11:48:05 EST 1999 End of getLongFromGMJO
Thu Nov 11 11:48:05 EST 1999 <DEBUG> after getting cookieValueLength
Thu Nov 11 11:48:05 EST 1999 Extracting char, Offset:5356, Name:cookieValue, Len:32,
Value=000000
00EBC03EAD00000001EC1FA521.

```

Thu Nov 11 11:48:05 EST 1999 <DEBUG> after getting cookieValue  
Thu Nov 11 11:48:05 EST 1999 <BR>Calling executeTxnCleanup().

---

## Unit Testing Page Servlets

This section will discuss a simple WinNT environment for developing servlet pages. There are other environments that can be used, but this is a simple setup that development and testing of page servlet code.

Java/2 Extended Edition and the WebSphere Application Server 4.0 or later contain most of the functions required to support page development and testing. The environment requires import statements for the `com.ibm.ifs.gold`, `com.ibm.ifs.gold.auto`, `com.ibm.ifs.resources`, and the `com.ibm.ifs.servlets` packages. This provides all of the IFS classes necessary for development.

The IFS CWS toolkit provides data for executing all gold transactions without being connected to the CWS Gold Manager. This is done with the use of string data files. These files are in the `cws/docs/jcwsapi/data` directory structure of the build. These are binary files that contain ASCII data, with no CR/LF characters imbedded in the data. String data file editing is discouraged.

To use these files, servlets must tell the IFS CWS toolkit where to find the string data files. This is done using the `ifs.system.strings.path` property in the `IfsSystem.properties` file located in `WEB-INF/classes` directory. For instance, if the string data is installed on the C: drive in the data directory, the following statement in the `IfsSystem.properties` properties file identifies the location to the Java classes:

```
ifs.system.strings.path=d:\\data
```

If the `ifs.system.strings.path` property is set to null, the IFS CWS toolkit will attempt to retrieve a transaction's data through the Gold Manager.

---

## Reauthentication/Relogon

The CWS Gold Manager maintains two separate inactivity timers for the purpose of minimizing the number of active stale user sessions. These timers prevent a user from tying up valuable system resources in scenarios such as logging onto the system and at some point walking away from the browser without logging off the system; then at some later time coming back to the browser and attempting to pick up working where they left off. The two timer values are maintained by the Financial Institutions and are retrieved by the CWS GM during server start up via the gold message Get IFS Profile Inquiry (GETIFSPR). The variable names are `,reauth_tm_val-` and `,inact_tm_val-` and their values are in seconds, **with the value of the `,reauth_tm_val` being smaller than the value of the `,inact_tm_val`.**

Each time a Gold Message is executed by a servlet page it is forwarded to the CWS GM for processing. The CWS GM determines the elapsed time between the previous Gold Message that was processed and the current time. It then compares this elapsed time value with the `,reauth_tm_val`. If the elapsed time value is larger than the `,reauth_tm_val` but smaller than the `,inact_tm_val`, the CWS GM returns a completion code value back to the servlet page indicating that a reauthentication condition has occurred. If the elapsed time value is larger than both the `,reauth_tm_val` and the `,inact_tm_val` the CWS GM returns a completion code value back to the servlet page indicating that a relogin condition has occurred. Detection of the reauthentication and relogin completion codes in a servlet page is performed by the `handleErrors()` method of the `IfsHttpServlet` class. (See ,Error Handling- on page 57.)

The `handleErrors` method in the `IfsHttpServlet` object processes completion code values returned in gold transactions. This method provides special processing for returned values representing a reauthentication or relogin condition. Both the reauthentication and relogin completion code values are specified in the `IfsSystem.properties` file located in `WEB-INF/classes` directory via the `ifs.system.reauth.brc` and `ifs.system.relogin.brc` properties. If the `handleErrors` method is overridden by a subclass of `IfsHttpServlet`, the special reroute processing to a reauthentication or relogin page will not work correctly. Additionally, for the reauthentication condition, if the `service` method of `IfsHttpServlet` is overridden by a subclass, the special reroute processing to a reauthentication page will not work correctly.



In a relolon condition, the `handleErrors` method reroutes the browser to a servlet page specified by the `ifs.system.pages.reauth.page` property in the `IfsSystem.properties` file located in `WEB-INF/classes` directory.

In a reauthentication condition, the `handleErrors` method saves the current URL, all HTTP request parameters and values sent to the current page, and the servlet request method type in the HTTP Session cache before rerouting the HTTP browser to a reauthentication form page. The reauthentication form page is specified by the `ReauthenticationPage` parameter in the `IfsSystem` object. An example Reauthentication Form Page servlet is supplied with the build and CWS toolkit. The file is `Reauth.java`.

Within the Reauthentication Form Page servlet, an HTTP form is created with two entry fields, one for the user's pin and one for the user's password and a number of hidden fields. The hidden fields written to the form consist of a new field called `reauth` and all of the HTTP request parameters and values that were sent to the servlet page that was interrupted by the reauthentication condition. The HTTP form statement is declared using the following lines of code:

```
PrintWriter pw = response.getWriter();
String page = IfsHttpServlet.getPreviousPage(request);
if (IfsHttpServlet.isPreviousPageMethodPost(request))
    pw.println("<FORM METHOD='post' NAME='refer_page' action='" + page + "'>");
else
    pw.println("<FORM METHOD='get' NAME='refer_page' action='" + page + "'>");
IfsHttpServlet.addPreviousPageRequestParameters(request,pw);
pw.println("<INPUT TYPE='hidden' NAME='reauth' VALUE='__GtReauth'>");
```

The static method `getPreviousPage` in the `IfsHttpServlet` object returns the servlet page name that was stored in the HTTP session cache during the execution of the `handleErrors` method. This enables the form to return control back to the servlet page that was interrupted by the reauthentication condition. The HTTP form method type can be determined with the static method `isPreviousPageMethodPost` in the `IfsHttpServlet` object. If this method returns a boolean value of `true`, the form method of the servlet page that was interrupted by the reauthentication condition was a **post**. Otherwise, if this method returns a boolean value of `false`, the form method of the servlet page that was interrupted by the reauthentication condition is assumed to be a **get**. The static method `addPreviousPageRequestParameters` in the `IfsHttpServlet` object adds all of the HTTP request parameters to the form as hidden parameters. These parameters were the ones that were saved in the HTTP Session cache during execution of the `handleErrors` method.

When the Reauthentication Form Page is displayed in the browser, the user enters the pin and password and then submits the form by clicking the Enter button. This will transfer control back to the servlet page that was interrupted by the reauthentication condition. The service method in the `IfsHttpServlet` object for the page looks for a specific HTTP Request Parameter to determine if a `GtReauthentication` transactions needs to be processed. The request parameter is the hidden field `reauth` that specifies an object id to be used for the `GtReauthentication` transaction. An example of the object id is shown as `__GtReauth`. If this special HTTP Request parameter is valid, the pin and password values from the form are input to an instance of the `GtReauthentication` transaction, and the transaction is executed. If the execution of the transaction was successful, reauthentication is complete and control is passed on either to the `doPost` or `doGet` method of the servlet page. If the transaction execution failed, the browser is rerouted to the Error Page where the appropriate error message text is displayed.

---

## Using Resource Bundles for Gold Message Return Codes to Text Translation

To be National Language Support (NLS) enabled, the IFS CWS has placed a large amount of the ,Code To Text- (CTT) functionality in the Java construct ,Resource Bundles-. Currently supplied with the IFS CWS is a set of four ,default- property files (see ,National Language Enablement- for more detail ) in the ,com.ibm.ifs.resource- package that contain English text to be used for CTT for Gold Message ,return codes-. They are installed in the `WEB-INF/classes/com/ibm/ifs/resource` directory.

1. `Ifs3tkErrorTextBundle`
2. `IfsCwsErrorTextBundle`
3. `IfsPendingTransactionStatus`

#### 4. IfsSystemErrorResourceBundle

Any financial institution (FI) should be able to:

- use these classes ,as is-
- modify or augment them as required
- mimic the English version by creating similar properties files using text in whatever language(s) they require (naming them appropriately - see ,National Language Enablement-).

The other resource files that are related to return codes are generated at server startup (see ,Error Handling- for more information). For instance, the ,IfsBrcErrorTextBundle.properties- file which contains resources related to the ,Business Response Code- is auto-generated.

The term ,return codes- applies to the data received in the response from the invocation of a Gold Message by a Financial Institution (FI). The response data received is parsed into appropriate fields for later use by servlets or Java Server Pages (JSP). The data which comprises the return codes, is the ,error code-, ,completion code- and ,Business Response Code- (BRC), and reflects the success or failure of the Gold Message execution. There could be additional data reflecting more details of errors and other information, in the message but this is not discussed here.

The error code is the most important field and is the main item of data used to determine the transaction s success or failure. There are three possible error codes resulting from the execution of Gold Messages. They are ,0000-, ,3TK- and ,CWS- error codes.

- If the error code is ,0000-, the transaction is normally successful, but there could still be a BRC reflecting additional information the servlet should act upon.
- If the error code is a ,Three Token- (3TK) error, there could still be a BRC returned, and if so should be considered the true result of the transaction.
- If the error code is a ,CWS- error, the web server itself has experienced an error situation.

For 3TK and CWS errors there will be a ,completion code- reflecting additional information on the success or failure of the transaction. There will also be a reason code and sensedata for 3TK errors.

Each type of completion code and BRC will have an associated hashtable and resource bundle. The hashtables contain data that reflect the severity of the completion code or BRC and will be discussed later. If multiple languages are supported, each language must have its own set of the three bundles available with text messages translated to that language. The three error code types and their associated resource bundles are:

- ,Successful-, which could also be accompanied by a Business Response Code (BRC) and normally would come from the FI (resource bundle IfsBrcErrorTextBundle).
- 3TK error code, which comes from the FI or the Core Controller, which could also be accompanied by a BRC. The resource bundle related to 3TK errors is the Ifs3tkErrorTextBundle. If a BRC also exists, the IfsBrcErrorTextBundle is used to extract the test associated with the code.
- ,Customizable Web Server- error code, which comes from the web server itself (resource bundle IfsCwsErrorTextBundle).

Normally only one of these would happen at a time, but there are occurrences where a 3TK error code could be returned as well as a BRC for a given transaction. In this case, the BRC is assumed to be valid and is what should be acted upon by the servlet rather than the 3TK error code.

For each of the above types of completion codes and BRC s, there exists in the ,IfsSystem- class a hashtable that reflects the ,severity- of the error. Current possibilities include:

- ,SUCCESS-
- ,INFO- (for information to be conveyed to the consumer)
- ,REDO- (normally reflects that the consumer tried to do something incorrect or not allowed by the FI)
- ,FATAL- (reflects that some error has happened in the CWS, Core Controller, or FI that could warrant termination of the consumer s session or some other drastic action).
- ,SPECIAL- (require special handling)
- ,LOGON- (error occurred during logon)
- ,LOGOFF- (error occurred logging off the customer)

The IFS Java CWS currently has data in the three hashtables (brc s, 3tk and cws types) defined in the IfsSytem class. The tables can be loaded in a servlet by calling the *initIfs()*.

**Note:** IfsSystem is a class extending HttpServlet.

## Normal Transaction Behavior

After a server is initially started, and upon the first consumer logging on, the above-mentioned hashtables are populated and a constant is set which reflects the current ,Locale- of the server. The Locale is information stored in the machine s operating system that reflects the country and language that machine is configured for.

Upon invocation of a Gold Message, the returned response will have one or more return codes included. The IfsTxn class will parse the Gold Message response and assign the appropriate data within the message to the ,error code-, ,completion code- and BRC fields.

- If the data values for error code and BRC are ,0000- and ,000- respectively, this is considered a successful transaction.
- If the error code is ,0000- (successful), and a BRC other than ,000- (successful) exists, the BRC is used to determine the severity of the error condition by doing a lookup in the BRC hashtable.
- If the error code is a 3TK error, and there is no BRC, the completion code is used to determine the severity of the error condition by doing a lookup in the 3TK hashtable.
- If the error code is a 3TK error, and there is a BRC, then the BRC is used to determine the severity of the error condition by doing a lookup in the BRC hashtable.
- If the error code is a CWS error, the completion code is used to determine the severity of the error condition by doing a lookup in the CWS hashtable.

A second lookup is then performed to find the language translation in the associated resource bundle for this completion code and/or BRC, using the ,Locale- of the server. The browser locale may be used instead if the browserLocaleUsage property in IfsSystem indicates to do so. The Locale is used to determine which resource bundle is used for the text lookup. When a completion code or BRC is retrieved from the Gold Message response, the actual text for it will be retrieved from the appropriate resource bundle using the completion code or BRC and the Locale value (see ,National Language Enablement- for more details). If a completion code or BRC value from a gold message does not exist in the hashtable it will be considered a FATAL error. If it does not exist in the associated resource bundle, a message indicating this will be returned.

## Error Handling

Within the IfsHttpServlet class there is a method, ,handleErrors-, which should be called after each Gold Message execution. If an error is detected, this method uses the resulting severity of the transaction to determine the appropriate action to perform. In the IfsSystem class there are values initialized which define the default servlet to be executed upon the occurrence of the 3 types of severity. Based on the value of the severity the ,handleErrors- method will execute the appropriate servlet. These servlets are:

- InfoPage - used to display pertinent information resulting from a (normally) successful transaction execution (that is, ,The date you requested for a transfer was changed due to holidays-).
- RedoPage - used to display information resulting from a (normally) unsuccessful transaction (that is, ,Transfer to and from accounts are identical-).
- ErrorPage - used to display more drastic failure information, and are typically either programmer error or technical problems in the execution of your transaction (that is, ,An internal system level error has occurred-).

These servlets display appropriate values based on the results of the transaction. Alternatively, the developer can elect to handle return codes in a customized way. For example, within their servlet, changing the values for InfoPage, RedoPage, or ErrorPage to a servlet different than what was initialized in IfsSystem to redirect to a custom page(s), or by writing their own ,handleError- routine(s).

## Technical Details

The following is a list and description of the error fields within a transaction:

- compCode - FI condition codes
- reasonCode - additional backend condition codes
- ErrorFunctionName - CWS function that failed

- `ErrorModuleName` - CWS module that failed
- `message` - BRC translated text from resource bundles. For CWS errors, the message is retrieved from the `com.ibm.ifs.resource.IfsCwsErrorTextBundle`.
- `responseCode` - BRC translated text from GM tables
- `gmMessage` - CWS, CC translated text
- `senseData` - CC error text string
- `senseDataLength` - length of error text

The following methods are provided which return the data referred to in the text above:

Method Name	Method Function
<b>getBrc</b>	This method returns the BRC associated with the Gold Message. A value of ,000- is a successful transaction. Any other value would reflect an INFO, REDO, or FATAL condition (see <code>getStatus</code> ).
<b>getStatus</b>	This method returns the severity of the completion code or BRC which was present in the Gold Message response, and will have a value of INFO, REDO, or FATAL. Doing a lookup in the appropriate hashtable returns this value. If no value is found in the hashtable, a value of ,FATAL- is returned.
<b>getCompCode</b>	This method returns the value of the ,completion code-, which is returned by the core controller or FI. This is a value that is used for indexing into the hashtable and the appropriate resource bundle.
<b>getMessage</b>	This method is used to retrieve the BRC text string from the ,message- field which would have been populated by the lookup in the BRC resource bundle from the execution of the Gold Message.
<b>getGmMessage</b>	This method is used to retrieve the 3TK or CWS text string from the ,gmMessage- field which would have been populated by the lookup in the 3TK or CWS resource bundle from the execution of the Gold Message.

---

## National Language Enablement

### Resource Bundle

A *resource bundle* is a set of resources such as strings, numbers and images. A resource can be any subclass of *Object*. Every resource in a resource bundle has a unique case-sensitive name (a string), called the key, which is used to retrieve the resource from the resource bundle. The resource here means the textual descriptive text of the key. Since a resource bundle is a live Java object that holds the resources, each resource bundle can be implemented by a different class. However, there is a special case in which there is no need to declare a new Java class to handle resources that are stored in a properties file (see Java's Properties class for more detail). In the current releases of the IFS CWS, properties files are used rather than a Java class.

### Using Resource Bundles for Localization

The IFS CWS uses Java's resource bundle as the primary mechanism for localizing applications. In fact, an application should store all of its localizable resources in resource bundles. This makes it possible to create a version of the application for a particular locale by duplicating the set of resource bundles (with slightly different names) and then replacing the resources with localized versions. There are rules for naming the localized resource bundles that make it possible to avoid modifying the application at all. These rules are explained in the following sections.

### Resource Bundle Names

The name of a resource bundle has two parts the *base name* and the *locale identifier*. The base name is any valid Java class name such as `IfsBrcErrorTextBundle`. The locale identifier has three parts the language

code, the country code, and a variant code each separated by an underscore character. An example of locale identifier is *en\_US\_MAC*, where *en* is the language code, *US* is the country code and *MAC* is the variant code. The complete resource bundle name would be *lfsBrcErrorTextBundle\_en\_US\_MAC*. International Standards Organization documents, ISO 639 Language Codes- and, ISO 3166 Country Codes- defines the 2-character symbol for the majority of possible supported languages and countries, and can be found at the ISO web sites: <http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>, and <http://userpage.chemie.fu-berlin.de/diverse/doc/ISO3166.html>, respectively.

In general, when an application is being built, the localized resources are placed in resource bundles whose names do not include a locale identifier. These resources are called *default resource bundles*. To build a localization for a particular language, append the two-character language code onto the resource bundle's base name, separated by an underscore character. Similarly, to build a localization for a particular country, the name of the language-localized resource bundle is appended with the two-character country code separated by an underscore character. Finally, if the resources in a resource bundle have to be different for different dialects or platforms, the resource bundle can be further localized to a dialect/platform.

As previously mentioned, the IFS CWS has a set of four, default- (English) resource bundles which can be used as a model for additional languages. In the previous releases, the four bundles (Java classes) provided were *lfs3tkErrorTextBundle*, *lfsCwsErrorTextBundle*, *lfsSystemErrorResourceBundle* and *lfsBrcErrorTextBundle*, while in the current release the four bundles (properties files) are *lfs3tkErrorTextBundle*, *lfsCwsErrorTextBundle*, *lfsSystemErrorResourceBundle* and *lfsPendingTransactionStatus*. Supporting an additional language would require the creation of these four classes similar to the four bundles mentioned above but appending the locale identifier to the filename (that is, *lfsBrcErrorTextBundle\_fr\_FR.properties*- for, French-) for a specific language or country.

## Auto-Generation of CTT Properties Files

In addition to the four resource bundles mentioned before, there are some more resource bundles that require code translation. These bundles are auto-generated during server startup. In previous releases the resource bundles used were of type *ListResourceBundle*-. In the current and all future releases, code to text resource bundles are changed from the *ListResourceBundle* to *PropertyResourceBundle*. The code to text mappings are maintained in the code to text tables on the Core Controller databases. During the web server startup time, the code to text tables are retrieved by issuing the CODERFSH gold message and the corresponding code to text property files are created. Such a mechanism allows updating and deploying code to text property files without recompilations of the Java code, and enables multiple web servers to retrieve the up-to-date code to text tables dynamically at the server startup time.

**Note:** When using WebSphere, the properties files will be loaded during web server startup as they are placed in the classpath. So the changes made to the properties files when the server is running, are not visible to the end-user.

The generation of these resource files is controlled by the parameters: **wsp\_ctt\_active** **wsp**, **wsp\_ctt\_prb\_path** in the web server *config file*. Setting the value of **wsp\_ctt\_active** **wsp** to *,1-* prior to server startup activates the generation of the properties files and stores them in the directory specified by **wsp\_ctt\_prb\_path**. Note the path specified by this parameter is the relative pathname of the web server and not the complete pathname. The code-to-text properties files generated use the *,lfs-* prefix, the Gold tag name excluding the *,Tg-* suffix if present, and the *,Resources-* suffix. for example, *lfsAcctTpCIResources*.

## Code-to-Text Translations

In previous releases, every transaction class (*Gt\*.java*) or data class (*Gs\*.java*) that had a field holding a nondescriptive code also had a corresponding translated text field. The name of this field had the same name as the code, but appended with *,\_text-*. Like other fields, this field was also extracted from the response buffer of the gold transaction and had corresponding setters and getters to set and get the field values respectively.

In the current release, a mechanism has been defined to handle code-to-text translations using the getters of the *,\_text-* fields that is, *get\*\_text()* methods of transaction and data classes to encapsulate the locales, resource bundles, and the encoded values inside these methods. All physical fields *\*\_text* of transaction and data classes are removed. The translated descriptive text of encoded values are now retrieved in classes using *get\*\_text()* methods based on the current session locale. For instance, the *GsAccount* class has a

method called `getType_text()` which gives the descriptive text for a type of account whose value is obtained by the method `getType()`. The `getType_text()` method uses `getType()` as the key for the lookup in the auto-generated resource bundle using the locale provided (see sample code below).

```
public class GsAccountS extends IfsObject
{
    ...

    protected String type;
    public void setType(String type)
    {
        this.type = type;
    }

    public String getType()
    {
        return type;
    }

    public String getType_text() throws IfsException
    {
        return IfsNLS.getText("IfsAcctSubTpClResources", getIfsSession().getLocale(), getType())
    }
    ...
}
```

All the `get*_text()` method use the static method of a utility class called `IfsNLS` to retrieve the locale-specific text encoded in a resource bundle using the code ( key ) and the locale. The description of this class is explained below.

**Note:** All `get*_text()` methods throw an `IfsException`. Any servlet that calls the `get*_text()` method needs to wrap the method in a try/catch block.

## Dynamic Session Locale

The IFS Web Server supports a locale object, called the session locale object, for each user session. This locale object is created and cached in the user session. This occurs during Logon, OBOLogon and Self Registration.

```
//fromGtLogon's execute method
if (IfsSystem.browserLocalUsage==IfsSystem.USE_SERVER_LOCALE)
{
    try{ sess.setLocale(IfsSystem.SERVER_LOCALE); }
    catch(IfsException e) {}
}
else
    IfsHttpServlet.checkBrowserLocale(sess.getHttpServletRequest());
```

The session locale object can be dynamically set based on each client's request during the runtime using the browser's language preference or using the programming interfaces in servlets. The contents of the web pages are created using the current session locale. This allows end users to view the same web page in different locales.

The following methods can be used in the servlets to get and set the session locale object:

```
IfsObject.getLocale()
IfsSession.getLocale()
IfsSession.setLocale()
```

The default session locale is defined by `IfsSystem.SERVER_LOCALE`. A servlet can change the session locale to any specific locale based on the `HttpRequest`, using the `IfsSession.setLocale()` method. The locale information can either be passed explicitly as a request parameter or defined in the request header through the browser's language preference feature. However, several scenarios are possible. For example, an FI may want to allow the end user to use the browser's language preference to change locale at any time. Or it may check the browser's language setting only at logon time. After logon, the FI might only allow the user to change language choices using a dropdown list or buttons in the web pages.

In order to support these possible FI customization scenarios, the following flag is available in the `IfsSystem.properties` file:

```
# If set to 1, specifies that the browser's HTTP request header,
# ACCEPT-LANGUAGE from the logon page is to be used to set this session's
# locale. If the logon page does not contain a valid ACCEPT-LANGUAGE
# attribute the server's locale is used to set the locale for this session.
# If set to 2, specifies that the browser's ACCEPT-LANGUAGE attribute
# is checked on each page. If the browser's ACCEPT-LANGUAGE
# attribute is valid and it is different from this session's locale,
# this session's locale is updated to the new locale. Whenever, the
# browser page does not contain a valid ACCEPT-LANGUAGE attribute and the
# session locale is not initialized, the server's locale is used to set the
# locale for this session.
#ifs.system.browser.locale.usage=1
```

In `IlsHttpServlet`'s service method, the Accept-Language in a `HttpRequest` header will be processed and the session locale will be properly set based on the `ifs.system.browser.locale.usage` property.

---

## Using the NLS Utility Class - `IlsNLS`

This class contains a set of static methods to be used to facilitate the Locale-based string translation. Although this class was intended to be used by the `get*_text()` methods, it can be used in servlets as well. Following is a list of methods in the `IlsNLS` class used for extracting the textual resource:

- `public static String getString(String bundleName, String key)` throws `IlsException`
- `public static String getText(String shortBundleName, String key)` throws `IlsException`
- `public static String getText(String shortBundleName, Locale locale, String key)` throws `IlsException`
- `public static String getString(String bundleName, Locale locale, String key)` throws `IlsException`

All these methods return the text associated with the *key* parameter. The resource bundle used for the lookup has the name, *bundleName* appended by the *locale* identifier (if provided). If the locale object is not provided as an argument then the web server Locale object is used. The argument *bundleName* is a fully qualified package name such as `com.ibm.ifs.resource.IlsBrcErrorTextBundle`. The *shortBundleName* refers to the name of the resource bundle. This name is not a fully qualified name. The method adds the package name to the provided bundle name (*shortBundleName*). An example of such a class name is `IlsBrcErrorTextBundle`. These methods are used mainly in the gold transaction (Gt\*) and data (Gs\*) classes.

The *key* may contain Unicode characters including space. In other words, the key in the resource bundles can be localized. The space and other invalid characters put a constraint in locating the text resource for the key. For this reason the key is encoded using the `URLEncoder` class before the lookup in the resource bundle is triggered. If the key contains a space, then the key in the resource bundle (properties file) has to be URLEncoded. Also since the textual resource (or the string returned in above methods) contains UTF-8 encoded characters, they are converted to Java Unicode characters using the `toUnicodeString` method of `IlsHttpServlet`.

**Note:** The key is UTF-8 encoded in the properties file of the auto-generated resource (properties) files. So the key passed in the argument of the above methods is first UTF-8 encoded and then URLEncoded to match the encoding of the key in the properties file.

The methods internally handle the `MissingResourceException` thrown during the search for either the resource bundle or the text resource. If the exception is thrown during the search for the resource bundle then the method throws `IlsException`, whose message contents include the resource bundle name and locale information. On the other hand, if the exception is thrown inside the method due to the missing resource (key), then an attempt is made to search for the default ,ZZZ- key in the resource bundle. If found, its resource value is returned, otherwise an `IlsException` is thrown. The message contents of this exception includes the resource bundle name and locale information.

## Locating a Resource Bundle: `IlsNLS.getBundle()` vs `ResourceBundle.getBundle()`

To locate a resource bundle, the static method `getBundle()` of either `IlsNLS` or Java's `ResourceBundle` class can be used. Since the functionality of these two methods are different, precautions should be taken to make sure they fit the purpose.

Consider how these methods differ. To retrieve a bundle using either of these methods, two pieces of information are required: the fully qualified base name of the resource bundle and a locale identifier. These two strings are concatenated together, separated by an underscore character, to form a class name. An attempt is then made to load a class with that class name (using the default system class loader). An example of such a class name is `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US`, where `com.ibm.ifs.resource.IfsBrcErrorTextBundle` is the fully qualified base name and `en_US` is the locale identifier. If a class with the class name cannot be loaded or instantiated, the class name is then successively shortened until a resource bundle class is successfully loaded and instantiated.

The shortening process is as follows. First, the complete class name base name plus the desired locale identifier is used. If this fails, the locale identifier is continually shortened by removing the rightmost underscore and subsequent characters until a resource bundle class is successfully loaded and instantiated. This is where the two methods differ. In the `getBundle` method of `IfsNLS` the search goes up to the base name and if the base resource bundle is found it is loaded and instantiated, whereas in `Java s ResourceBundle` class the base name is ignored and the process starts again using the locale identifier of the default locale.

To demonstrate the above process of class names attempted, here is an example using a base name of `com.ibm.ifs.resource.IfsBrcErrorTextBundle` and a desired locale of `Locale.JAPAN` with a `WIN_95` variant. The default locale is `Local.ENGLISH`.

In the `ResourceBundle` class, the attempts made are in the following order,

1. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN_95`
2. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN`
3. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA`
4. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp`
5. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US`
6. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en`
7. `com.ibm.ifs.resource.IfsBrcErrorTextBundle`

In the `IfsNLS` class, the attempts made are in the following order,

1. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN_95`
2. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN`
3. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA`
4. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp`
5. `com.ibm.ifs.resource.IfsBrcErrorTextBundle`
6. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US`
7. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en`

Whenever a class name fails to produce a resource bundle object, a check is made to see if a properties file with the same name exists. In particular, the class name is appended with the string `,.properties-`. If such a file exists, a `PropertyResourceBundle` object is created for that properties file. To demonstrate more concretely the sequence of files attempted, consider the following example.

In `ResourceBundle`,

1. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN_95`



2. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN_95.properties`
3. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN`
4. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN.properties`
5. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA`
6. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA.properties`
7. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp`
8. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp.properties`
9. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US`
10. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US.properties`
11. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en`
12. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en.properties`
13. `com.ibm.ifs.resource.IfsBrcErrorTextBundle`
14. `com.ibm.ifs.resource.IfsBrcErrorTextBundle.properties`

In `IfsNLS`,

1. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN_95`
2. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN_95.properties`
3. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN`
4. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA_WIN.properties`
5. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA`
6. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp_JA.properties`
7. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp`
8. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_jp.properties`
9. `com.ibm.ifs.resource.IfsBrcErrorTextBundle`
10. `com.ibm.ifs.resource.IfsBrcErrorTextBundle.properties`
11. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US`
12. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en_US.properties`
13. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en`
14. `com.ibm.ifs.resource.IfsBrcErrorTextBundle_en.properties`

## Textual Representation of Gold Primitives in Localesensitive Manner

The textual presentation of the `GsCurrency`, `GsDecimal`, `GsDate`, `GsTime` objects are formatted based on the current session locale. The exception to this is when one of these objects has been constructed with a

specific format or when the format has been specifically set. In these cases, the format that has been constructed or set, overrides the current session locale.

**GsCurrency:** The GsCurrency class is used to represent IFS-supported currencies. The class is NLS enabled. Currencies may be constructed and formatted in the Java-supported Locales. By the use of an associated class, GsCurrencyFormat, custom formats can be constructed.

In general, formatting is based on the convention that a particular locale L wants to see foreign currencies formatted in locale L's format with the exception of the use of the correct currency symbol. This requires a dynamic alteration of the currency pattern for a particular locale to use the correct currency symbol. In rare cases, when the pattern's decimal separator is contained in the foreign currency symbol, this approach will not work and the pattern for the foreign currency is used without alteration. Also, if Java 1.1.x does not contain a locale corresponding to a given ISO code, formatting will be done using the current session locale.

The formatting process described above is slightly different for the Euro. When given an ISO code of EUR, the current local is checked for support. If it does not support the EUR, an attempt is made to access a Euro variant locale. If the result of that attempt does not yield a locale that supports the EUR, the process above is used to insert the Euro symbol into the current session locale currency pattern.

Certain multiple-currency situations require the use of the correct method signatures. NLS capabilities and restrictions will be noted in the affected method's documentation.

There are two types of NLS support offered through GsCurrency:

1. If a user wants a GsCurrency object to be formatted according to the current Locale (which may be changed during a session), then constructors should be used that do not pass in a GsCurrencyFormat object. Also, direct setting of the object's format via the setFormat method must be avoided.

Example:

```
ifsSession.setLocale(Locale.FRENCH);
GsCurrency amount = new GsCurrency("0000123456789", 2, 1, "USD", ifsSession);
String result = amount.toString();
```

The result would be: 1 234 567,89 \$

2. If a user wants a GsCurrency object to be formatted a certain way regardless of the current Locale, then constructors can be used that pass in the GsCurrencyFormat object or the format can be set via the setFormat method.

Example:

```
ifsSession.setLocale(Locale.FRENCH);
GsCurrency amount = new GsCurrency("0000123456789", 2, 1, "USD", ifsSession);
GsCurrencyFormat usFormat = new GsCurrencyFormat(Locale.US);
GsCurrency.setFormat(usFormat, amount);
String result = amount.toString();
```

The result would be: \$1,234,567.89

**GsDecimal:** GsDecimal class extends GsDecimalS. It provides function for the construction, formatting and comparing of GsDecimal instances.

Just as in GsCurrency above, there are two types of NLS support offered through GsDecimal:

1. If a user wants a GsDecimal object to be formatted according to the current Locale (which may be changed during a session), then constructors should be used that do not pass in a GsDecimalFormat object. Also, direct setting of the object's format via the setFormat method must be avoided.

Example:

```
ifsSession.setLocale(Locale.FRENCH);
GsDecimal amount = new GsDecimal("0000123456789", 2, 1, ifsSession);
String result = amount.toString();
```

The result would be: 1 234 567,89

2. If a user wants a `GsDecimal` object to be formatted a certain way regardless of the current `Locale`, then constructors can be used that pass in the `GsDecimalFormat` object or the format can be set via the `setFormat` method.

Example:

```
ifsSession.setLocale(Locale.FRENCH);
GsDecimal amount = new GsDecimal("0000123456789", 2, 1, ifsSession);
GsDecimalFormat usFormat = new GsDecimalFormat(Locale.US);
GsDecimal.setFormat(usFormat, amount);
String result = amount.toString();
```

The result would be: 1,234,567.89

**GsDate:** To represent a `GsDate` textually in locale-sensitive manner, the `toString()` method should be called. To represent it in a specific format, `toString(GsDateFormat)` can be used. The `toString()` method uses default locale or system locale. Prior to the method call, the specific format can be set by invoking `setLocaleFormatter(GsDateFormat)`. This will set the `localeFormatter` field of `GsDate`. If the `localeFormatter` is never set then the call to `toString()` method will use the current session locale, in which case, if it is not set then it uses the default locale.

To illustrate, consider the following example:

```
GsDate date = new GsDate("06/14/2000", "MM/dd/yyyy", ifsSession);
// assume ifsSession is created using HttpServletRequest and HttpServletResponse
// and encapsulates locale, Locale.JAPAN date.toString();
// returns in Japanese format (short ) "00/06/14"

date.setLocaleFormatter(new GsDateFormat(Locale.US));
date.toString(); // returns in USA format (short ) "6/14/00"
//subsequent calls to toString() returns in USA format even if the session locale is changed.
```

**GsTime:** The process involved in representing `GsTime` in a locale-sensitive manner is similar to `GsDate`. The `toString()` method should be used in locale-specific format. Subsequent calls to the method will return in current session locale format if a specific format is not set. To set a format, `setLocaleFormatter(GsTimeFormat)` is used. The `toString()` method utilizes `getLocaleFormatter()` to represent local-sensitive format. If a format is set, then the call to `getLocaleFormatter()` will always return that format.

To emphasize this process, consider the following example:

```
GsTime time = new GsTime("08:40 AM", "hh:mm a", ifsSession);
// assume ifsSession is created using HttpServletRequest and HttpServletResponse
// and encapsulates locale, Locale.JAPAN
date.toString(); // returns in Japanese format (short ) "8:60"

date.setLocaleFormatter(new GsDateFormat(Locale.US));
date.toString(); // returns in USA format (short ) "8:40 AM"
//subsequent calls to toString() returns in USA format even if the session locale is changed.
```

## Charsets and Conversions

- Definition of the webserver charset:

The charset used on an IFS webserver can be configured using the `IfsSystem.cwsBrowserCharSet` variable. Its value can be set to either ,UTF-8- or ,8859\_1-.

- `IfsHttpServlet.getParameterValues()` method:

When the `IfsSystem.cwsBrowserCharSet` is set to ,UTF-8-, the HTTP request parameter value strings are presented in the UTF-8 format. Since the `getParameterValues` methods in the `javax.servlet.ServletRequest` class do not correctly convert from UTF-8 to unicode, the following method is provided in the `IfsHttpServlet` class to get the request parameter strings and convert them to unicode strings:

```
public static String[] getParameterValues(String name, String charSet, HttpServletRequest req)
```

- Set charset in the page content type:

The method `IfsHttpServlet.setDefaultHeader()` now sets the charset in the page content type using `IfsSystem.cwsBrowserCharSet`. It is important that the charset for the page be set prior to getting a `PrintWriter` so the correct writer is obtained for the charset.

---

## JDBC Considerations

IFS CWS V1R6M1 introduces new function that enables applications to input data directly into DB2 tables without sending the data through the CWS Gold Manager and the TDM. Additional software is required for connectivity between the AIX machines and the host DB2 tables. Refer to the *IFS Planning Guide* for a list of the required software programs and any special installation instructions.

Two additional DB2 tables must be created and configured before using the transactions listed below. The database schema for these new tables can be found in the *IFS Transaction Delivery Manager Guide*.

Five new classes have been defined to manipulate data in the new DB2 tables. They are:

- `IfsJDBC`,
- `IfsPendingTransactionInstance`,
- `IfsPendingTransactionInstanceDetail`,
- `IfsWebTransactionActivity`, and
- `IfsWebTransactionActivityDetail`.

These classes enable a servlet page to create, list, read, update and delete rows from the DB2 tables. Refer to the V1R6 JavaDocs for information on the methods available for these new classes.

The `IfsJDBC` class is the ancestor for both the `IfsPendingTransactionInstance` and the `IfsWebTransactionActivity` classes, providing common routines used by both.

The `IfsPendingTransactionInstanceDetail` is the storage structure holding the result set returned, for a query from `IfsPendingTransactionInstance`.

The `IfsWebTransactionActivityDetail` is the storage structure holding the result set returned, for a query from `IfsWebTransactionActivity`.

Host database connections require a user id and password. These values can be set by the page programmer using the `setUserID()` and `setPassword()` methods of the transaction classes or by using the default values from the `IfsSystem` class. The default values are maintained within the `IfsSystem` class under the attributes `IFS_WDBC_USERID`, and `IFS_WDBC_PSWRD` respectively. A database driver name is required for the database. This value can be set by the page programmer using the `setDriverName()` method or by using the `defaultDriverName` value from the `IfsSystem` class. If no driver name is set by the page programmer, the default value from `IfsSystem` is used. Finally, a URL address to the database is required for connection to the database. This value can be set by the page programmer using the `setUrl()` method or by using the default value from the `IfsSystem` class. The default value is maintained within the `IfsSystem` class under the attribute `IFS_WDBC_DBALIAS`.

A programming example is supplied in `v1r6m1_yymmdd.tar.Z` under the `cws/testdocs/jcbc` directory. To install the example, copy `pendingTransactionInstanceForm.jsp` and `pendingTransactionInstanceServlet.jsp` into the root of the `ifs.war` directory, and copy `PendingTransactionInstanceServlet.class`, `PendingTransactionInstanceForm.class`, `MethodViewBean.class` and `PendingTxnViewBean.class` into the `ifs.war/WEB-INF/classes` directory.

The example uses the Model-View-Controller concept. The JDBC example consists of these files: `PendingTransactionInstanceForm.java`, `pendingTransactionInstanceForm.jsp`, `PendingTransactionInstanceServlet.java`, `pendingTransactionInstanceServlet.jsp`, `PendingTxnViewBean.java`, and `MethodViewBean.java`. The `PendingTransactionInstanceForm` servlet and `pendingTransactionInstanceForm.jsp` are used for inputting data into the form. The `PendingTransactionInstance` servlet and `pendingTransactionInstanceServlet.jsp` process and display the data, using `PendingTxnViewBean` and `MethodViewBean`.

**To Convey Error Messages:** Since this functionality is provided in the model of an API, error messages for connection errors and validation errors are given in the form of exceptions. Therefore, when

the methods provided by these classes are used, they must be enclosed in a try-catch block (to catch the exceptions thrown by these methods).

## JDBC Configuration

JDBC access requires changes to both AIX and WebSphere configurations.

1. in user s **\$HOME/.profile**,  
add following path to CLASSPATH:

```
export
CLASSPATH=$CLASSPATH:${INSTHOME}/sql1lib/java/db2java.zip:${INSTHOME}/sql1lib/java/runtime.zip:
${INSTHOME}/sql1lib/java/sqlj.zip
```

add following path to LD\_LIBRARY\_PATH:

```
export LD_LIBRARY_PATH=$(LD_LIBRARY_PATH):${INSTHOME}/sql1lib/lib
```

**Note:** INSTHOME is the DB2 instance home directory in our case is: /home/db2inst1

2. To establish connections to DB2 through JDBC providers and Data Sources, you will need to configure JDBC providers and Data Source. To accomplish this, use the WebSphere Administrative Console. Follow the directions for configuring JDBC providers and DataSources in the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*, Chapter 16: Configuring WebSphere resources. The RedBook is available at:  
<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246176.html?Open>

## Host Database Table Changes

The database tables associated with the JDBC servlets have been modified slightly. The revised DDL is shown below and should replace the existing tables:

- PNDING\_TXN\_INST
- REL\_TXN\_ACTY

provided with the OS/390 R811 delivery without requiring changing any other R811 components.

```
--
--
--      Licensed Materials - Property of IBM
--      5648-A06
-- © Copyright IBM Corp. 1996 All Rights Reserved
--      US Government Users Restricted Rights - Use, duplication
--      or disclosure restricted by GSA ADP Schedule Contract
--      with IBM
--
--
--      Use This version of DDL if 'DB2 UDB for OS/390 V6'
--      is NOT installed. It uses a datatype of VARCHAR(31000).
--
--      Changed 01/19/01 - TDMV1R6_015 - the following columns have been
--      changed from "not null" to nulls allowed:
--
--      PNDING_TXN_INST.TXN_AMT
--      PNDING_TXN_INST.TXN_AMT_PRC
--      PNDING_TXN_INST.TXN_CURR_TP
--      REL_TXN_ACTY.TXN_AMT
--      REL_TXN_ACTY.TXN_AMT_PRC
--      REL_TXN_ACTY.TXN_CURR_TP
--      REL_TXN_ACTY.TXN_BENF
--      REL_TXN_ACTY.TXN_VAL_DT
--
SET CURRENT SQLID = '#TBLOWNR';
--
CREATE TABLESPACE #TSPREFX411
IN #DBNAME
USING STOGROUP #STOGRP PRIQTY 480 SECQTY 96
FREEPAGE 0 PCTFREE 20
SEGSIZE 64
```

```

LOCKSIZE ANY
LOCKMAX SYSTEM
BUFFERPOOL BP32K
CLOSE NO;
• Header name PNDTXNA
CREATE TABLE PNDING_TXN_INST
(
    FI_ID CHAR(8) NOT NULL ,
    PRIM_CNSMR_ID CHAR(30) NOT NULL ,
    TXN_INIT_CNSMR_ID CHAR(30) NOT NULL ,
    SRVC_RQST_NM CHAR(8) NOT NULL ,
    TXN_INIT_TMSTP TIMESTAMP NOT NULL WITH DEFAULT,
    SRVC_NM CHAR(8) ,
    TXN_DB_ACCT_CLH_ID CHAR(28) ,
    TXN_DB_ACCT_TP CHAR(3) ,
    TXN_DB_ACCT_SUB_TP SMALLINT ,
    TXN_DB_ACCT_NBR CHAR(22) ,
    TXN_DB_ACCT_NKNM CHAR(30) ,
    FOR BIT DATA ,
    TXN_AMT DECIMAL(15,0) ,
    TXN_AMT_PRC SMALLINT ,
    TXN_CURR_TP CHAR(3) ,
    TXN_BENF CHAR(40) ,
    FOR BIT DATA ,
    TXN_REF CHAR(40),
    TXN_VAL_DT DATE NOT NULL ,
    TXN_STAT_CD SMALLINT NOT NULL WITH DEFAULT,
    TXN_AUTH_CNSMR_ID CHAR(30) ,
    TXN_AUTH_TMSTP TIMESTAMP ,
    TXN_DATA VARCHAR(31000) NOT NULL
)
IN #DBNAME.#TSPREFX411;
COMMENT ON TABLE PNDING_TXN_INST
IS 'PENDING TXN INSTANCE (VARCHAR)';
CREATE TYPE 2 UNIQUE INDEX PNTIX001
ON PNDING_TXN_INST
(
    FI_ID ASC,
    PRIM_CNSMR_ID ASC,
    TXN_INIT_TMSTP ASC,
    TXN_INIT_CNSMR_ID ASC,
    SRVC_RQST_NM ASC,
    TXN_STAT_CD ASC
)
USING STOGROUP #STOGRP PRIQTY 96 SECQTY 96
FREEPAGE 0 PCTFREE 20
CLUSTER
BUFFERPOOL BPO
CLOSE NO;

COMMIT;
CREATE TABLESPACE #TSPREFX401
IN #DBNAME
USING STOGROUP #STOGRP PRIQTY 480 SECQTY 96
FREEPAGE 0 PCTFREE 20
SEGSIZE 64
LOCKSIZE ANY
LOCKMAX SYSTEM
BUFFERPOOL BP32K
CLOSE NO;

• Header name RELTXNA
CREATE TABLE REL_TXN_ACTY
(
    FI_ID CHAR(8) NOT NULL ,
    PRIM_CNSMR_ID CHAR(30) NOT NULL ,
    TXN_INIT_CNSMR_ID CHAR(30) NOT NULL ,
    SRVC_RQST_NM CHAR(8) NOT NULL ,
    TXN_LOGGED_TMSTP TIMESTAMP NOT NULL WITH DEFAULT,
    TXN_AMT DECIMAL(15,0) ,
    TXN_AMT_PRC SMALLINT ,
    TXN_CURR_TP CHAR(3) ,
    TXN_BENF CHAR(40)

```

```

        FOR BIT DATA
        TXN_REF          CHAR(40)
        TXN_VAL_DT       DATE
        TXN_LOG_REASON_CD INTEGER          NOT NULL
        TXN_AUTH_CNSMR_ID CHAR(30)
        TXN_AUTH_TMSTP   TIMESTAMP
        TXN_DATA         VARCHAR(31000)    NOT NULL
    )
IN #DBNAME.#TSPREFIX401;
COMMENT ON TABLE REL_TXN_ACTY
IS 'TRANSACTION ACTIVITY (VARCHAR)';
CREATE TYPE 2 UNIQUE INDEX RTATX001
ON REL_TXN_ACTY
(
    FI_ID          ASC,
    PRIM_CNSMR_ID  ASC,
    TXN_LOGGED_TMSTP ASC,
    TXN_INIT_CNSMR_ID ASC,
    SRVC_RQST_NM   ASC
)
USING STOGROUP #STOGRP PRIQTY 96 SECQTY 96
FREEPAGE 0 PCTFREE 20
BUFFERPOOL BPO
CLOSE NO;

```

## Procedures to Establish Connection to DB2 Database using JDBC

There are 2 ways to establish connection to desired databases.

1. Use the defaults in `IfsSystem.java`. By specifying defaults in `IfsSystem.java` we can establish connection to database without using setters and getters. There are 5 constants defined in `IfsSystem.java`, which needs to be specified.

```

/**
 * Name of default JDBC driver. Used if a driver name is not set within
 * the IfsJDBC class.
 */
public static String defaultDriverName = "COM.ibm.db2.jdbc.app.DB2Driver";

/**
 * Name of default database url used to establish a JDBC connection.
 * Used if a url is not explicitly set within
 * the IfsJDBC class.
 */
public static String IFS_WDBC_DBALIAS = "jdbc:db2:DBTIFAV";

/**
 * Name of the default region name used in one's database schema.
 * This has to be set in IfsSystem.java.
 */
public static String IFS_WDBC_SCHEMA = "DBTIF91C";
This constant is accessed in IfsPendingTransactionInstance.java and
IfsWebTransactionActivity.java.

/**
 * Name of default database user id used to establish a JDBC connection .
 * Used if a user id is not explicitly set within
 * the IfsJDBC class.
 */
public static String IFS_WDBC_USERID = "";

/**
 * Name of default database password used to establish a JDBC connection.
 * Used if a password is not explicitly set within
 * the IfsJDBC class.
 */
public static String IFS_WDBC_PSWRD = "";

```

- **defaultDriverName:** Remains the same for DB2 databases.
- **IFS\_WDBC\_DBALIAS:** In this, constant `jdbc:db2:` remains the same but `DBTIFAV` will change according to the desired database name or region.

- IFS\_WDBC\_SCHEMA: This is the qualifier needed to identify the correct table being accessed. This has to be set in IfsSystem.java.
  - IFS\_WDBC\_USERID: Should be the Id of the person who created the database specified in IFS\_WDBC\_DBALIAS.
  - IFS\_WDBC\_PSWRD: Should be the password of user id specified in IFS\_WDBC\_USERID.
2. Using setters and getters of IfsJDBC. Developers can use following methods to connect to desired database.

#### Set Driver name:

```
/*This method is used to set the Driver Name.
*It must be set before establishConnection is called
*@param o - driver name.
*/
public void setDriverName(String o )
```

Example:

```
xClass.setDriverName("COM.ibm.db2.jdbc.app.DB2Driver")
```

For DB2 databases driver name ,COM.ibm.db2.jdbc.app.DB2Driver- remains the same.

#### To set Database URL:

```
/**
 * This method is used to set the url property.
 * It must be set before establishConnection is
 * called.
 * @param o-The url of the database
 */
public void setUrl(String o )
```

Example:

```
xyzClass.setUrl("jdbc:db2:DBTIFAV");
```

Where jdbc:db2 remains the same for DB2 database and DBTIFAV should be changed according to desired database name or region.

#### To set User Id:

```
/**
 * Used to set a user's database id
 * <P>
 * @param id. The user id.
 */
public void setUserID(String id)
```

Example:

```
xyzClass.setUserID("abc");
```

Where abc , should be the Id of the person who created the database

#### To set Password:

```
/**
 * Used to set a user's database password
 * <P>
 * @param password. The user's password.
 */
public void setPassword(String o)
```

Example:

```
xyzClass.setPassword("def");
```

Where def , should be the password of user id specified in setUserID meth

**Other Dependencies on IfsSystem.java and locale:** The following are default values set in IfsSystem.java:



```

public static boolean IS_VALIDATE_ACTION_REQUIRED = true;
public static boolean IS_REQUEST_READY_ACTION_REQUIRED = false;
public static boolean IS_RESPONSE_READY_ACTION_REQUIRED = false;

```

The Currency Type for the Transaction Amount is picked up from the locale information.

---

## Considerations in Using Multiple User Ids

### Financial Impacts

User Interface (UI) servers such as the Web Server, CSR server, etc., may be customized to authenticate a consumer's transaction authority. This is an FI implementation choice. Internal IFS service providers, such as transfer, will verify authorities but the FI may negotiate with external service providers as to whether the verification is done in the UI customizable server or at the external service provider on a service by service basis.

The new secondary consumer messages are routed directly to the FI. It is the FI's responsibility to respond. This includes gathering all data in response to the Secondary Consumer Inquiry and provide for more processing when those results exceed the capacity of the message size. No order is implied in the Secondary Consumer Inquiry results.

The FI may need/wish to develop servlets to implement specific business logic. The following sections outline a possible suggested set of business logic that can be implemented in an FI-developed servlet.

### GtAddSecondaryConsumer

The servlet cannot invoke the GtAddSecondaryConsumer class unless the user has their Secondary Consumer Maintenance authority flag in their IFS Consumer Profile set to ,Y-. The servlet will need to provide the client with a list of available services/accounts to choose from based on the services/accounts available to the Primary Consumer. The user may select all or a subset of the presented services/accounts. The services/accounts are based on the service/accounts (including transactions limits in the accounts base currency code).

The primary consumer should be able to specify the initial PIN for a secondary consumer so the servlet must prompt for the PIN and prompt for the primary consumer to re-enter the PIN. The servlet should compare the initial PIN to the re-entered PIN and either reject the add or complete the transaction depending on whether the PINs match. Before actually transmitting the transaction the servlet should prompt the primary consumer to confirm the addition. Whether the servlet does this will be controlled by the Service Request Confirmation Flag (CFRM\_RQRD column) in the IFS Service Request Profile (SRVC\_REQ table) for the Add Secondary Consumer message. When prompting for confirmation, the servlet should redisplay the contents of the Add Secondary Consumer transaction (in human readable form) to the primary consumer. The servlet should not issue the transaction unless the primary consumer confirms.

A transaction limit value of \$0.00 is valid.

The default service/account authorization if the primary consumer selects a service and account but does not select the authority will be , Inquiry Only-.

When submitting an Add Secondary Consumer transaction, the servlet should **not** fill in the following fields in IFS\_CNSMR\_PRFL\_DTL as they will be completed by the FI or completed by the TDM during a later logon:

```

CnsmrEffDtTg,
CnsmrEndDtTg,
NewMsgFTg,
LastCnsmrSsnChrgTg,
CnsmrIdTg in IFS_CNSMR_ID in IFS_CNSMR_PRFL_DTL.

```

Also, the new profile fields (Date of Last PIN Change, Date of Last Successful Logon and FI Consumer Status) should not be filled in by the servlet since they will be completed by the FI prior to the first logon.

The following fields in IFS\_CNSMR\_PRFL\_DTL should be filled in by customizable code since the FI should control what values are placed in these fields for a secondary consumer. Typically, the FI should want the values to come from the primary consumer s profile:

CnsmrSignupSrcTg,  
FiEmpTpIndTg,  
CnsmrAddrDtlTg,  
SrvChrgPlnTg,  
CnsmrTpCITg,  
CnsmrMktSegTg,  
CnsmrDivisionTg,  
CnsmrLglNmTg,  
ChrgBasisTpCITg,  
DailyLmtAmtTg

The primary consumer should be prompted to complete the rest of the profile fields.

PrimCnsmrldTg in IFS\_CNSMR\_ID in IFS\_CNSMR\_PRFL\_DTL should be set to the primary consumer s permanent consumer ID.

## GtChangeSecondaryConsumer

In order to invoke the servlet for the GtChangeSecondaryConsumer class, a prior invocation for the GtSecondaryConsumerInquiry class must have been done (for full profile data) and cached the results (which included the consumer profile for each secondary consumer a primary consumer has created). The cached data about a secondary consumer should be displayed in order for the primary consumer to change what they wish.

The servlet cannot be invoked for the GtChangeSecondaryConsumer class unless the user has their Secondary Consumer Maintenance authority flag in their IFS Consumer Profile set to ,Y-. When submitting a Change Secondary Consumer, the servlet should **show the current values but not allow change** for the following fields in IFS\_CNSMR\_PRFL\_DTL as they should never be changed by the primary consumer:

CnsmrEffDtTg  
CnsmrEndDtTg  
NewMsgFTg  
LastCnsmrSsnChrgTg  
ChCnsmrldTg, CnsmrldTg (as well as PrimCnsmrldTg) in IFS\_CNSMR\_ID,  
CnsmrSignupSrcTg  
FiEmpTpIndTg  
CnsmrAddrDtlTg  
SrvChrgPlnTg  
CnsmrTpCITg  
CnsmrMktSegTg  
CnsmrDivisionTg  
CnsmrLglNmTg  
ChrgBasisTpCITg  
DailyLmtAmtTg

Also, the new profile fields (Date of Last PIN Change, Date of Last Successful Logon and FI Consumer Status) should only echo back the current values since they are never changed by the primary consumer. The primary consumer should be prompted to change (if desired) any of the rest of the profile fields with similar rules as listed for GtAddSecondaryConsumer.

If a field is not changed, the Change Secondary Consumer transaction should contain the original value for that field since this a full replacement of the secondary consumer s profile. The primary consumer must be able to add/change/remove services or accounts for a secondary consumer. They must also be able to change transaction limits and change the Secondary Consumer Maintenance authority setting. Once the primary consumer has changed everything they wish to, the servlet should validate that no invalid selections have been made then a confirmation process similar to the one specified in the previous section for GtAddSecondaryConsumer should be followed. No transaction will be issued unless the primary consumer has confirmed the change.

## GtCloseSecondaryConsumer

The servlet cannot be invoked for the GtCloseSecondaryConsumer class unless the user has their Secondary Consumer Maintenance authority flag in their IFS Consumer Profile set to ,Y-.

In order to use the servlet for this class, a prior servlet invocation for the GtSecondaryConsumerInquiry class must have been done (for full profile data) and cached the results (which included the consumer profile for each secondary consumer a primary consumer has created). The cached data about a secondary consumer should be displayed in order for the primary consumer to select secondary consumers to close. Confirmation of a close secondary consumer will be implemented similarly to confirmation of add secondary consumer. The servlet will invoke the IFS certificate agent to send an encrypted message to the Certificate Authority requesting that the certificate be added to the revocation list maintained by the Certificate Authority's directory server. Currently, the IFS certificate agent is on the IFS registration server and will have to be ported to the CWS in order to be locally available to the servlet.

## GtResetSecondaryConsumer

The servlet cannot be invoked for the GtResetSecondaryConsumer class unless the user has their Secondary Consumer Maintenance authority flag in their IFS Consumer Profile set to ,Y-.

Confirmation of a reset secondary consumer will be implemented similarly to confirmation of add secondary consumer.

## GtSecondaryConsumerInquiry

The servlet cannot be invoked for the GtSecondaryConsumerInquiry class unless the user has their Secondary Consumer Maintenance authority flag in their IFS Consumer Profile set to ,Y-.

The servlet will issue the Secondary Consumer Inquiry request message defined in this DCR. The invoker will need to be able to specify whether they wish just a list of secondary consumers or the list of secondary consumers with their profile data.

---

## Potential Issues and Problems

The following are a few issues or problems to be aware of when using the IFS CWS Toolkit.

## Instance Variables of a Servlet

In an HttpServlet, an instance variable is shared by all concurrent executions of doGet, doPost methods in responding to concurrent HTTP requests. Therefore, instance variables can not store data specific to a particular session. The following servlet code caused one customer to see details of another customer's account:

```
public synchronized class GtAccountDetailJava_Test extends HttpServlet
{
    ...
    private PrintWriter pageWriter = null;
    ...
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        ...
        // Set the socket to print the page
        try
        {
            pageWriter = res.getWriter();
        }
        catch(Exception ex)
        {
            ...
        }
    }
}
```

```
}
```

Here, the instance variable `pageWriter` caused the problem. An instance variable in a servlet is shared by all concurrent HTTP requests. When request 1 came in, the `pageWriter` was set to the response 1 associated with request 1. During the execution of the transaction 1 for request 1, request 2 came in, and the `pageWriter` was then reset to the response 2. Therefore, when transaction 1 completed and the information was written to the `pageWriter`, the output went to response 2 instead of response 1. Therefore, the browser that issued request 2 saw the account info for both customers.

Variables that belong to the processing of an individual request should not be defined as instance variables. They should be defined inside `doGet`, `doPost`, etc.

## Duplicate Transaction Problem

Page developers and FIs should be aware that consumers can cause unwanted duplicate transactions to be sent to a web server. The following are some possible scenarios.

1. **Multiple Clicks:** If a consumer clicks multiple times on a ,submit- button, the browser may submit multiple identical transactions to the web server. This behavior differs based upon which browser and version is used, the HTML type of submit button, and how many times the button was clicked.
2. **Enter-Back-Enter:** The consumer submits a transaction by clicking on a submit button. The consumer sees the response. The consumer uses the browser s ,Back- button to go to the previously submitted transaction and clicks on submit again.
3. **Enter-Reload:** The consumer submits a transaction by clicking on a submit button. The consumer sees the response and then clicks on the browser s ,Reload- button. Clicking the reload button causes the transaction to be sent to the web server again.
4. **Resizing:** Sometimes, resizing the browser window may cause the browser to behave as if the Reload button had been clicked.

Currently, the use of client-side JavaScript can help prevent the first scenario, the multiple click. For the `<FORM>` page elements, an `onSubmit` JavaScript even handler routine can be specified as follows:

```
<FORM method = ...onSubmit = "return checkIfSubmitted()">
```

For the `HREF` page elements, an `onClick` JavaScript routine can be specified as follows:

```
<A HREF = ...onClick = "return checkIfSubmitted()">
```

The following is a sample `checkIfSubmitted()` JavaScript event handler routine:

```
<SCRIPT>
var isFormSubmitted = "no";

function checkIfSubmitted()
{
    if (isFormSubmitted == "no")
    {
        isFormSubmitted = "yes"
        return (true);
    }
    else
    {
        return (false);
    }
}
</SCRIPT>
```

The use of `onSubmit` and `onClick` event handlers prevent duplicate transactions from being submitted from multiple clicks of submit buttons. However, this technique introduces one drawback dealing with a normal usage scenario. If a consumer enters one invalid field on the form and submits the transaction, the execution of the transaction will return an error message indicating the invalid input field. Then, if the consumer uses the ,Back- button on the browser to get the cached form page, makes the correction on the input field, and clicks the submit button to re-submit the transaction, the `checkIfSubmitted()` will prevent such re-submit. To allow for this, the form page should be coded such that a reload will retain the input data previously entered.

In addition, this client-side technique can only prevent the ,multiple click- scenario. It can not prevent the three other causes of duplicate transactions mentioned above.

## Using Object Cache

In `HttpServlet`, an object cache mechanism is provided with APIs: `putObject(...)`, `getObject(...)`, and `removeObject(...)` to allow multiple pages to share objects. Due to the dynamic nature of the page navigation, it is hard to find a proper spot to remove cached objects. Thus, when system-generated OIDs are used, frequent reload of pages that create a new transaction object and put it in the session cache will cause many unused transaction objects to remain in the session cache without being able to be garbage collected. With hundreds and thousands of consumers online at the same time, such usage scenarios may exhaust the server memory space.

Therefore, the servlet pages that create objects and put them in cache need to be coded carefully to minimize the memory usage. The following are a few approaches to deal with this problem assuming that `PageA` is a form for inputting data; and `PageB` handles the POST and creates the transaction object and puts it into the cache.

1. When using a system-generated OID, `PageA` can use `getNewOid()` to get a system generated OID in advance even before the actual object is created. `PageA` can pass the OID on the HTTP request as follows:

```
out.println( "<A HREF = \"PageB?txn=\" + getNewOid(session) + \"\"></A>");
```

When `PageB` receives this request, it tries to get the transaction object from the cache using the OID in the request and can not find the transaction object. Then, `PageB` creates a new transaction object and caches it under the OID. Since the OID is defined in the request, when `PageB` is reloaded, the transaction will be retrieved from the cache to be reused. Therefore, the transaction object will not be created or cached multiple times.

2. If no multiple pages of the same servlet are required on the client's screen at the same time, a customized OID can be used to minimize the usage of the cache.

On `PageB` when creating the transaction object and putting it in the cache, Instead of using:

```
GtAddVariablePayment payment = new GtAddVariablePayment(ifsSession);
putObject(ifsSession, payment);
```

use:

```
GtAddVariablePayment payment = new GtAddVariablePayment(ifsSession);
putObject(ifsSession, "__Payment", payment);
```

In this way, every time the page is reloaded, a new transaction object is created; the previous transaction object cached under the OID `__Payment`- is removed from the cache; and the new transaction object is cached under the OID. Thus, there will be no unused multiple transaction objects in the cache due to the reload.

Better yet, the transaction object does not have to be created every time. The following code illustrates how to reuse the transaction object:

```
GtAddVariablePayment payment = (GtAddVariablePayment )getObject(ifsSession, "__Payment");
if ( payment == null)
{
    payment = new GtAddVariablePayment(ifsSession);
    putObject(ifsSession, "__Payment", payment);
}
```

If multiple pages of the same servlet on the client's screen at the same time are required, a system generated OID is the best way to handle the situation.

---

## Chapter 5. Troubleshooting Java in the CWS

Within CWS, JSPs are used to generate HTML to be shipped to the browser. To execute Java code within the CWS, a Java interpreter, called the Java Virtual Machine (JVM), is now included within the CWS. This capability is provided through the inclusion of the IBM Websphere product within the CWS.

Servlets are written as a combination of FI-provided capability, along with CWS-provided interfaces for common functionality, Gold interface, etc. This latter set of functionality is called the Java Configurable Web Server (CWS) Application Program Interface (API), or JCWSAPI.

---

### FI-Developed Servlets

Any time CWS servlets are created and compiled that had previously been referenced and executed within a CWS, that CWS must be brought down (with the Administrative Console or the `tools/stopweb` command) and restarted (with the Administrative console or the `tools/startweb` command). An exception to this requirement for FI-Developed Servlets is contained within the path specified by the Reloadable Servlet Classpath within WebSphere (see the `servlets.properties` file).

For full information on how to replace or deploy Java classes under WebSphere, see ,Appendix H. Building And Deploying CWS Web Applications In WebSphere 4-.

---

### Java CWS API Classes

The relevant Java CWS API classes for troubleshooting include the ,system- classes, the ,security- classes, the ,Gold- classes and the `ifs.system.debug` system property in `IfsSystem.properties` file in the `WEB-INF/classes` directory. If values in any of these files (the files with the `java` extension) are changed, they must be recompiled with the `javac` command. If any Java files are changed or the debug property is changed, the CWS must be brought down and restarted.

The **system classes** contain system-wide definitions for common data types and system inheritance structure from the Java `http servlet` class. Some of these data types include Accounts (`GsAccount`), Date (`GsDate`), Decimal Data (`GsDecimal`), Decimal Format (`GsDecimalFormat`), Time (`GsTime`), Currency (`GsCurrency`), Currency Format (`GsCurrencyFormat`), Self-Defining Fields (`GsGmSelfDefiningField{Data,Def}`). The class `IfsSystem`, contains system-wide definitions for an instance of the CWS that includes country codes, language codes, page colors, layout parameters, FI identification, location of images as well as default values for various CWS parameters. These files should not be changed. These files are located in the directory `cws/docs/jcwsapi/com/ibm/ifs/gold`.

The , **security-** classes implement certain Gold transactions that require encryption of data and session management rather than just transmission of that data from the browser to the FI. These include classes for Logon (`GtLogon`), Change Pin / Password (`GtChangePinPassword`), Self Registration (`GtSelfRegistration`), Reauthentication (`GtReauthentication`), Logoff (`GtLogoff`) and Small Business Wire Transfer (`GtSmallBusWireTransfer{Init,Cancel,Inquiry}`). These files should also not generally be changed. These files are also located in the directory `cws/docs/jcwsapi/com/ibm/ifs/gold`.

The , **Gold-** classes implement the remaining Gold transactions not covered above. These files are automatically generated and should definitely not be changed. These files are located in the directory `cws/docs/jcwsapi/com/ibm/ifs/gold/auto`.

The , **Resource-** classes are Java resource bundles that contain mappings for CWS error codes and their respective error messages, as well as the Code To Text data. The error code resource bundles (`Ifs3tkErrorTextBundle`, `IfsCwsErrorTextBundle`, `IfsBrcErrorTextBundle`, and `IfsSystemErrorResourceBundle`) are provided as part of the Java CWS API installation. The Code To Text resource bundles are generated from the information in the TDM Code To Text database table that is downloaded during CWS startup or when a Code To Text table refresh is requested (CODERFSH transaction).

**Note:** The TDM Code To Text table can contain thousands of rows of data and use of this feature may increase the CWS startup time significantly.

---

## Traditional CWS Debugging

### CWS Trace Files

If tracing has been enabled (setting `,wsp_trace_flag=1-` in the `cws/config` file, specifying `,wsp_trace_file=XXXXX-` [for example, `/tmp/xyz`], and setting the value of `,wsp_max_trace_size=YYYYYY-` [for example, `1000000`], setting the file to be truncated at startup `,wsp_erase_log_at_startup=1-` and stopping and restarting the CWS), an internal trace of CWS will be performed. It provides timing information on transaction processing through various parts of the CWS and can help to pinpoint locations where timeouts may be affecting the operation of the CWS. The resulting trace file (a binary file) can be displayed using the tool `ifs.war/tools/tracefmt`. To run `tracefmt`, the path to `ifs.war/bin` must be in the `LIBPATH`.

### Gold Trace Files

A trace of the Gold transactions issued from the CWS to the FI is managed by the value of the CWS config file variable `,wsp_gold_trace_active-` (setting to `,1-` enables this capability) and the variable `,wsp_gold_trace-`, which specifies the file to which the Gold messages will be dumped.

**Note:** Due to system and AIX file system flushing, this file may not always contain a complete message transaction for the last transaction issued.

### CWS Message Log

The messages delivered from the CWS to the `,console-` can also be maintained in a log file. This file is specified by the value of the variable `,wsp_log_file-`, which can be truncated at server startup by setting the value of the variable `,wsp_erase_log_at_startup-` to `,1-`.

### Core Files

Sometimes the program will perform an operation that causes an exception to occur. The AIX Operating System will generate a core file that contains the values of the data structures in use at the time an exception occurs (most commonly a signal `SIGSEGV - 0a`). Sometimes CWS will itself determine that it has an internal inconsistency, report an error and call the necessary routines to create the core file as well. This core file should be saved, along with all available trace files and other logs for development to investigate. It is not uncommon for this file to be as much as 400 - 500 megabytes long. Occasionally this file will be truncated because of disk space limitations or because the shell under which the command `tools/startweb` was started had limited the size of generated core files (issue the command `,limit-` to see the current settings in `csh`).

---

## Java Specific Debugging Capabilities

### ifsSystem.properties and ifs.system.debug

Enabling debugging tracing within the CWS via the `ifs.system.debug` system property in `ifsSystem.properties` file in the `WEB-INF/classes` directory:

1. Change the value of `ifs.system.debug` from `false` to `true`.
2. Determine the output destination and medium of the debug statements by changing the `ifs.system.debug.to.std.out`, `ifs.system.debug.to.html`, and `ifs.system.debug.to.file` properties.

3. If using `debugInfoToFile`, changing the value of `ifs.system.debug.log.file.name` from null to the name of the file for the debugging output (for example, `./tmp/USERID.javadebuglog-`);
4. Stop and restart the CWS.

**Note:** Enabling this level of debugging functionality will cause SEVERE performance degradation within the CWS and generate a significant amount of additional output, which has the potential for filling up filesystem space if not monitored.

## javacore.txt file

The file `javacore.txt` has some similarities and some differences from the traditional AIX core file. The `javacore.txt` file is generated by the Java Virtual Machine (JVM) upon instances where it determines an internal inconsistency has occurred. The `javacore.txt` file consists mainly of a backtrace of the execution of the JVM at the time of the inconsistency. Unlike the traditional AIX core file, there is no data, state-information maintained about the execution of the CWS at the time that the file is generated, so it is not possible to connect a debugger to the `javacore.txt` file and look around at the values of the data at the time the exception occurred. The `javacore.txt` file is particularly useful in determining whether callouts have been made to the Java Native Interface (JNI) functionality along the path leading to the exception occurrence.

To turn off the `javacore.txt` file generation in order to obtain an AIX core file with more state information, set the JVM system property `DISABLEJAVADUMP` to true. This can be done on the `JVM Settings` page of your Application Server in the Administrative Console.

---

## Specific Error Messages

### 78 - Timeout Has Occurred

Receipt of this error during webserver startup can indicate that the TDM or SPA may have gone down. Receipt of this error during the operation of CWS may indicate that either the MQ timeout for this Gold message has expired or that either the TDM or SPA may have gone down.

### 404 - File Not Found

This is a classic http protocol error. It occurs both when the http server is unable to find a page as specified by the Universal Resource Locator (URL) provided by the browser, or when the WebSphere process is unable to find a servlet. In the first case, review the logs/errors and logs/access files under the directory in which the CWS was started. Unless the paths have been changed, the root path for pages to be delivered to the browser is the `cws/docs` directory. In cases where WebSphere is unable to find a servlet, check the value of the `CLASSPATH` variable in the environment before the CWS was started. It may need to be updated when new installations of CWS are performed in new locations.

### 801 - Unarchitected Error Code

This is a result of attempting to run a transaction through the CWS before the back end (the websrvr process) has completed its initialization. The delay could be due to a number of problems - timeout with the Financial Institution (FI) for the various messages at server startup time (Code to Text data, E-Biller Logos and Clearinghouse data) or misconfiguration with the Core Controller for routing of the messages are the most likely candidates.



---

## Chapter 6. CWS Developer's ToolKit

---

### Introduction

This chapter will outline the specific steps for installing and running the Customizable Web Server (CWS) Toolkit for developers. The CWS Toolkit is a collection of programs, procedures and documentation used to assist in the development and testing of the bank's Customizable Web Server pages. The CWS Toolkit is also referred to as the Toolkit.

The Toolkit provides a CWS-like environment for bank web page developers to write and test bank web pages. It accepts CWSAPI requests made from a web page, and returns test data appropriate to the request. The Tool provides the CWS environment in a standalone manner. It does not require network connections to a Transaction Delivery Manager (TDM) or other Interactive Financial Services (IFS) Interface Modules.

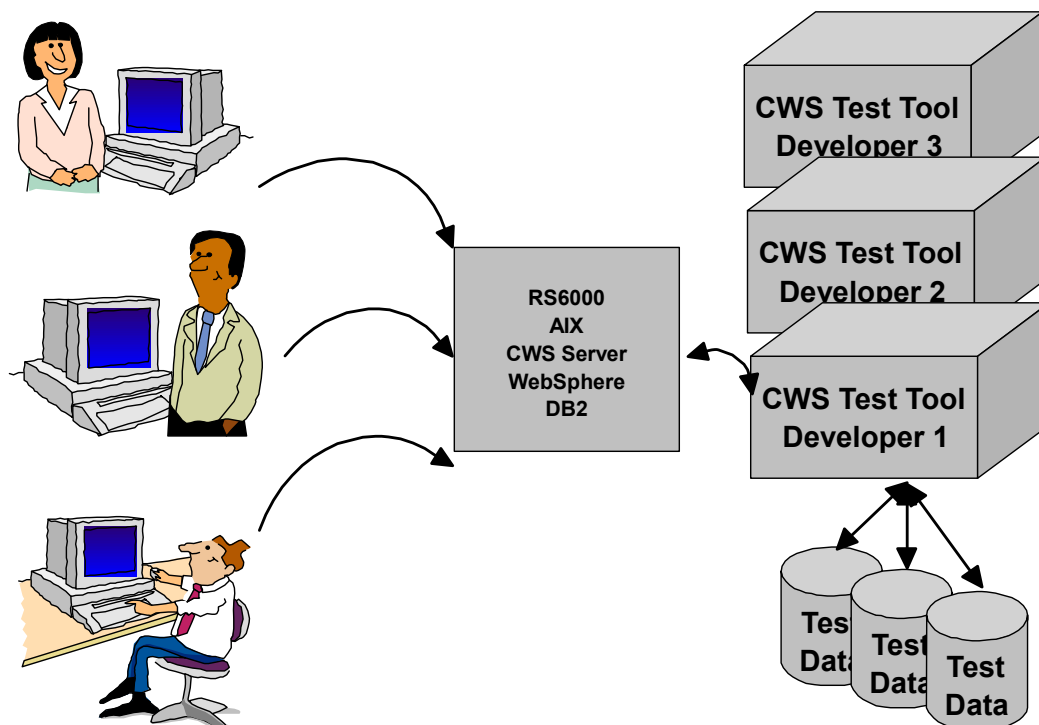


Figure 3. CWS Toolkit Environment

The Tool simplifies the task of the web page developer. The developer can focus on writing and testing web pages, rather than on the setup and debug of the underlying network. The Toolkit allows the developer to perform the following tasks:

- Visual test of the page
- Verify the linkage and flow from one page to another
- Verify the correct syntax is used when making CWSAPI requests
- Verify the returned data correctly populates the corresponding Java properties.

An optional enhanced simulation capability enables the developer to verify additional content in gold requests from the customized pages and enables the Toolkit user to modify the returned data. This enhanced simulation capability requires DB2 to be installed.

Information is presented for configuring the WebSphere server and the optional DB2 setup for use with the Toolkit. A section is also included on tips and techniques for developing and testing web pages in the CWS environment.

The Toolkit runs in a self-contained environment, and can be installed multiple times by multiple users in a system. Each developer configures and uses his/her own copy of the tool as depicted in Figure 5.

## Test Data Flow

In the bank's production environment, Gold transactions are sent from the web server to the FI, and the FI returns Gold data. In the Toolkit environment, Gold transactions are sent to the Toolkit, and the Toolkit returns Gold data.

Without DB2 installed, the Toolkit contains one Gold transaction response message for every Gold request message. The Tool returns the same response data for each request type, regardless of the request message parameters. The tool neither examines nor verifies the correctness of the request message; it simply returns a static response for each request.

With DB2 installed, the Toolkit examines the Gold request message to determine whether the transaction is supported by the enhanced simulation capability. Table 2 lists the supported transactions. If the transaction is not supported by the enhanced simulation capability, then the tool returns the same response as it would without DB2 installed.

When the request is supported by the enhanced simulation capability, the Toolkit examines the request to determine whether the request is valid based on the contents of the database. If the request is valid, the Toolkit returns a Gold transaction response based on data in the database. If the request is not valid based on the database contents, the Toolkit returns a Gold transaction to indicate the error. For example, the logon request will provide the account summary data if the user ID, password, and PIN match the database or it will return a logon failure message.

Figure 4 shows the data flow between the Toolkit and the web page.

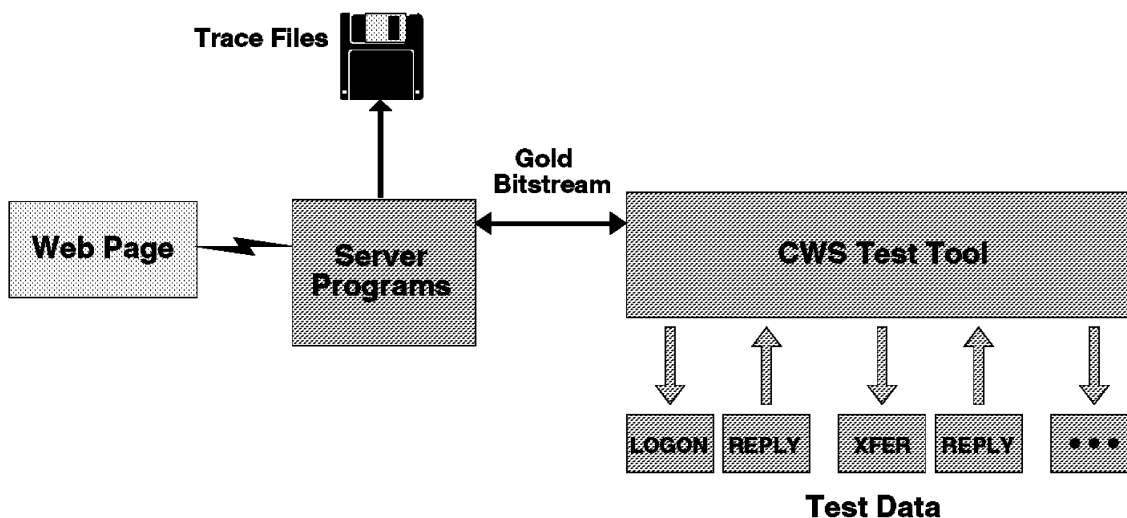


Figure 4. CWS Toolkit Data Flow

When a CWSAPI request is made from a page, the request is routed to the Toolkit which returns data based on an internal table for the transaction or from the database.

---

## Using the Basic Tool

It is suggested you use the tool initially without the DB2 option, so you get acquainted with the test programs and procedures. It is also suggested that initially you verify the tool installation for the first user before installing the tool for additional users.

As your expertise and familiarity increases, you can install DB2 to use the enhanced simulation capabilities, and include additional users for concurrent use of the tool.

The first step in getting started is to review the hardware and software environment needed to install the Toolkit.

## Prerequisites

### Hardware

The CWS Developer s Toolkit is installed on an IBM e-server pSeries or RS/6000. For a typical, three-user configuration, the following hardware environment is suggested:

- Entry level IBM e-server pSeries or RS/6000 system
- 256MB+ RAM
- 4GB+ disk space
- Graphics attachment for local user access
- LAN attachment for remote user access

### Software

See the *IFS Planning Guide* for a listing of all of the IFS Software Prerequisites.

## WebSphere Installation

The WebSphere requirements for using CWS Toolkit are the same as any CWS instance. After installing WebSphere and IBM HTTP Server (IHS), you must configure the following items:

1. CWS instance,
2. application server,
3. virtual host and
4. port number.

These steps are covered in Chapter 3. Installing CWS Components in the subheading ,Creating a CWS Instance-. If you have not done so, please refer to that section and create them.

## Installing the CWS Toolkit

Deploying the CWS Toolkit is similar to deploying any CWS instance. The primary difference is in the configuration options specified in `etc/config.toolkit`.

The CWS Toolkit may be deployed using the WebSphere Administrative Console method discussed in Chapter 3. Installing CWS Components, "Update and Deploy the ifs.war File- or manually by unpacking the compressed CWS Toolkit file on top of a previously deployed CWS instance. Regardless of deployment method, after deployment, configuring and running the CWS toolkit is the same.

The file name for the Toolkit will be something like `v1r6m1_020325tk.tar.Z`. The first part is the MTI level, followed by ,m1\_-, the date, and .tar.Z. If the file is downloaded to a system other than AIX, the browser may modify the file name. If this happens, rename the file to include the ,.tar.Z- ending after it is transferred to the AIX system.

## WebSphere Deployment

[Editor s Note: At the time this section was written, the Austin build was not corrected to create the CWS toolkit ifs.war file. These deployment steps were not tested so this may change slightly.]

The first step is to unpack the file containing the CWS toolkit files.

1. Create a temporary directory and place the `*tk.tar.Z` file in the directory.
2. Unpack the file with `zcat v1r6m1_020325k.tar.Z | tar -xvf`

3. Deploy the `ifs.war` file contained in the `cws/bin` directory. Do this with the WebSphere Administrative Console ,Install Enterprise Application- Installation Wizard.

The deployment of the `ifs.war` file is the same process discussed in Chapter 3. Installing CWS Components subheading ,Update and Deploy the `ifs.war` File-.

This ,Install Enterprise Application- Installation Wizard creates the CWS toolkit directory structure under the WebSphere Application Server instance directory. If you followed the instructions with default values this directory would be `/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war`.

## Manual Deployment

Assuming you already have an installed CWS instance, another method to deploy the CWS toolkit is to overlay the CWS toolkit files on top of the installed CWS instance. To do this, follow these steps:

1. Create a temporary directory and place the `*tk.tar.Z` file in the directory.
2. Unpack the file with `zcat v1r6m1_020325k.tar.Z | tar -xvf`
3. Copy or move the resultant tree to the WebSphere Application server instance. For example, go to the directory just above the unpacked CWS toolkit `cws` directory and recursively copy the all `cws` directory files including toolkit `bin`, `docs`, `tools`, and `fis` directories to the WebSphere Application server. If you followed the instructions with default values this `cws` directory would be under `/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war`.

## CWS Toolkit Package Contents

The CWS Toolkit is composed of the following elements:

- Toolkit programs
- The CWS programs
- Java servlet examples
- Test data files for use with DB2

After unpacking, the `cws` directory contains the `docs` directory which is the document root for the web server. The `cws` directory also contains the `bin` directory for the CWS executables and the `tools` directory for executable scripts. The `docs` directory contains the subdirectories `bld`, `jcwsapi`, and `servlets`. The `servlets` directory contains example Java servlets.

The `fis` directory is parallel to the `cws` directory. The `fis` directory contains the `data` directory for the DB2 based simulation data, the `bin` directory for the Toolkit executables, and the `testdata` directory for the response data file templates.

The Toolkit is organized using the directory structure shown in Figure 5.

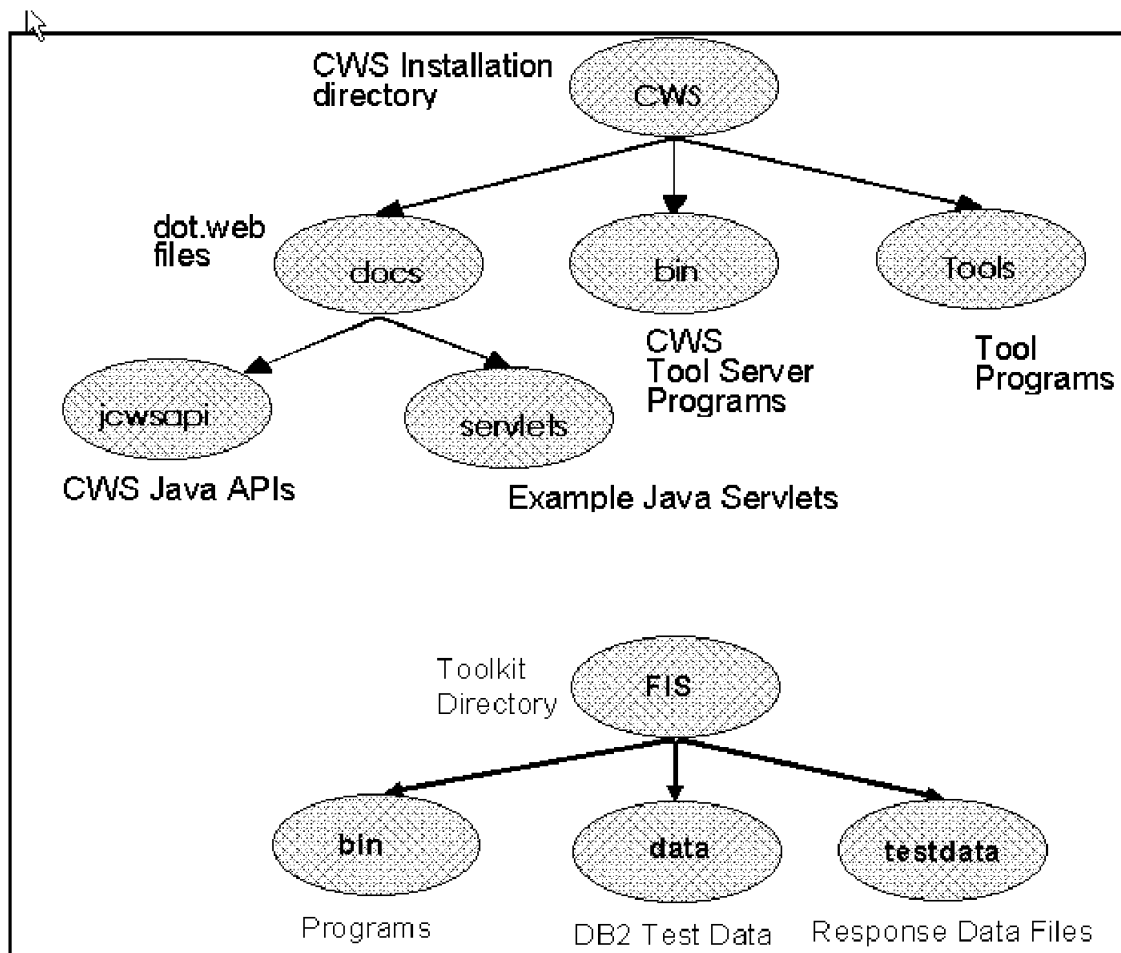


Figure 5. CWS Toolkit Directory Structure

## CWS Configuration File

The name of the sample CWS configuration file distributed with the CWS Toolkit is `config.toolkit` and is located in the `cws` directory. WebSphere installations must make the application server instance point to `config.toolkit` and then modify the new file for their web server name. You should always use the `config.toolkit` file distributed with the CWS as a base for your own configuration file.

To change the WebSphere Application server instance to point to the CWS toolkit `config.toolkit` file:

1. Start the WebSphere Administrative Console.
2. Select and highlight the Application Server instance.
3. Select and highlight the ,JVM Settings- tab on the configuration notebook.
4. Add or modify a system property named ,ifs.config.file-. The value of this property should be a relative path to your CWS toolkit config file. For example, enter `etc/config.toolkit`.

Please note that when making changes to configuration and sample files, you need to ensure that no extraneous characters are inserted. For example, if you use a DOS based editor, you need to remove the CR-LF and EOF characters from the file before using it with AIX.

## Configure the CWS Toolkit Server Programs

To get started, you only need to do the following:

- Substitute your web server name for WEBNAME.

If you did not install the server at the recommended path, then the paths in the config file will have to be modified accordingly.

Other setting modifications such as running code to text or ebiller logos follow the same procedure as described in the normal CWS server.

As you begin to develop and debug web pages, you may want to change additional parameters. The descriptions of these parameters are included in comments in the `config.toolkit` file. For example, the `sim_exe_name` is initially set to `fisimtkm` which provides a pre-stored response for each transaction type. Change this value to `fisimtkd` (last letter ,d- instead of ,m-) to run the DB2 based simulation after you have installed DB2.

## Running the Toolkit

To start the Toolkit manually use the following command from  
`/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war`

`tools/startweb`

You may verify the toolkit started by using any of the normal methods. First, check the WebSphere Administrative Console event queue to see the message ,WSVR0023I: Server <userid> open for e-business-. The next step is to read the `stdout.txt` and `stderr.txt` files produced when starting up the toolkit. Normally, these files are under the `websphere4/working` directories. Use the messages in these files to see if the configuration file was read properly and no error messages are produced.

If the CWS server (websrvr) did not start properly as indicated by the above messages, you will need to read the WebSphere Administrative Console Event Message log and determine the problem. The most common problems are typos in either the config file or in the classpath or libpath files used in running the `ifs.war` file.

## Using the Java Servlet Examples

The Java example servlets provided with the toolkit require that WebSphere be installed and correctly configured for the user instance. The example servlets can be run with either the ,`fisimtkm`- (canned data) or ,`fisimtkd`- (DB2 data) configurations.

The example servlets reside in the directory:

`/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/cws/docs/servlets`

To access the servlets, first setup the config file with the desired bank simulator (that is, ,`fisimtkm`-, ,`fisimtkd`-) and then start the CWS. On the browser enter the URL to call the Logon servlet. The URL format is ,`http(s)://<IPName>:<port number>/servlet/Logon`-. Dependant on the bank simulator being used, enter valid DB2 values for User ID, pin and password (for ,`fisimtkd`-) or any value for user id, pin and password (for ,`fisimtkm`-). The AccountSummary servlet will be called and the example page displayed. By following the instructions in each page display, most of the servlets, with the exception of QIF download functions, can be exercised. Exercise the QIF download servlets by entering ,`QIFDownloadForm`- after logging in.

The CSR OBO servlets require that the CWS be configured for a CSR user. After the cws is started, the `lfsLogon` servlet initiates the access to the OBO logon.

**Note:** Additional servlets may be provided as requested by the banks. All servlets for V1R6M1 are located in the servlets directory (i.e., `http-<instance id>/cws/docs/servlets`). All users should review the contents of this directory whenever a new Toolkit delivery is available.

## Enabling Server Encryption

When testing web pages with the Toolkit, you should be running without certificates or security. However, if you want to test using SSL, you need to obtain a server certificate from a certificate authority and install it. For the time being, recall that the objective is for you to test the pages, and that security (certificate and encryption) is beyond the scope of the current Toolkit.

---

## Using the Enhanced Simulation Tool

The enhanced simulation tool requires DB2 version 7.1 with fix pack 5 or later be installed, setup and a change to the CWS config file.

### DB2 Configuration

DB2 installation for use of the Toolkit may be done with either the panels included with the DB2 Version 7.1 product or with SMIT panels and some command line entries. The DB2 *Quick Beginnings* (S105-8148) book contains detailed instructions for this installation. This book can be used online starting with the web site <http://www.software.ibm.com/data/db2/library/> and following the links to ,DB2 Publications- and then ,Quick Beginnings for Unix-. Additional information can be found by downloading the FAQs, hints and tips, or Redbooks (extensive product use guides) at this web site:

<http://www-3.ibm.com/software/data/db2/udb/support.html>.

The next two sections summarize these instructions.

### Using DB2 Installation Panels

DB2 installation for use of the Toolkit may be done with the following steps:

1. Logon as root.
2. Create a Journal File System with mount point /home/db2inst1 and size 122000 blocks of 512 bytes or equivalent.
3. Mount the CD ROM containing the DB2 product using SMIT panels.
4. Run the “**db2setup**” program on the CD and install the following:
  - DB2 Client Application Enabler
  - DB2 UDB Workgroup Edition
  - DB2 Product Library - customize for en\_US
5. From these same panels, create a DB2 Instance and optionally create a DB2 Fence user and an Administration server. Go to ,Completing DB2 Setup-.

### Using SMIT Panels

Instead of the DB2 panel, installation for use of the Toolkit may be done with the following steps:

1. Logon as root.
2. Create a group ,db2-.
3. Create a user ID for ,db2inst1- with ,db2- as primary group.
4. Add the ,db2- group to all users of the tool.
5. Create a Journal File System with mount point /home/db2inst1 and size 122000.
6. Use the SMIT panels to install the following:

- DB2 Client Application Enabler Software Bundle

- DB2 Server(s) Software Bundle

DB2 Client Application Enabler

Code Page Conversion Tables - Uni Code Support

DB2 Connect

DB2 Communication Support for TCP/IP

DB2 Engine

DB2 Run-time Environment

DB2 Product Document (HTML) - English

Java Database Connectivity (JDBC) Support

Open Database Connectivity (ODBC) Support

License Support for DB2 UDB Workgroup Edition

7. Go to the /usr/lpp/db2\_05\_00/instance directory and create an instance with

```
./db2icrt -u db2inst1 db2inst1
```

8. As root, go to the /usr/lpp/db2\_05\_00/cfg directory and create links with

```
./db2ln
```

9. Go to the /usr/lpp/db2\_05\_00/cfg directory and update the product licenses

```
./db2licinst /usr/lpp/db2_05_00/install/db2work.lic
```

If you want to use a Lotus Approach or Excel spreadsheet on a Windows 95 system to view and change database tables, then do these steps on the AIX system and see below.

1. Type these commands:

```
db2set DB2COMM=TCPIP  
db2  
update database manager configuration using  
SVCENAME db2cdb2inst1  
quit
```

2. As root, edit the /etc/services file to add these lines:

```
db2cdb2inst1 50000/tcp # Communications port for DB2  
db2idb2inst1 50001/tcp # Interrupt port for DB2
```

3. As a user, stop and then restart the database manager with these commands:

```
db2stop  
db2start
```

## Completing DB2 Setup

1. Logon as a toolkit user and add this line to your .profile

```
./home/db2inst1/sql1lib/db2profile
```

2. Type db2 and the following commands:

```
db2start  
create database <userid>  
list database directory  
quit
```

3. Change to the /usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/fis/data directory and type "**setupfis -t**" to load the initial data in the database. A large number of messages from the database



manager will be displayed. Some of these messages will indicate a failure to delete a table the first time you run setupfis. These ,delete- commands will be needed when you rerun setupfis to restore the initial data. When this step completes successfully, the printout will include a statement near the end that says:

*Binding was ended with "0" errors and "0" warnings.*

Each time you reboot the machine, you must start the database manager again. This can be automated if root would add this line to /etc/inittab

```
rcdb2:2:wait:/etc/rc.db2 >/dev/console 2>&1 # Start db2
```

and create a file in the /etc directory containing the following:

```
#!/bin/ksh
#-----
# Start db2 database manager
#
# Note: We do this by becoming the db2inst1 user,
#       and running the script:
#       /home/db2inst1/sqllib/adm/db2start
#
#-----
su - db2inst1 '-c \
. /home/db2inst1/sqllib/db2profile; \
/home/db2inst1/sqllib/adm/db2start '
```

## CWS Configuration File for Enhanced Tool

For the enhanced simulation, change the CWS config file as follows:

- Change ,sim\_exe\_name- by changing the last character from ,m- to ,d- to read -fis/bin/fisimtkd-
- Set ,sim\_db2\_name- equal to <userid> (your database name) if this was not already done.
- Set the wsp\_code\_to\_text\_active and wsp\_ebiller\_logos\_active to the recommended values for the enhanced tool.

## Running the Enhanced Toolkit

As with the regular Toolkit, go to the directory

/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war, and type:

```
tools/startweb
```

You may verify the toolkit started by using any of the normal methods. First, check the WebSphere Administrative Console event queue to see the message ,WSVR0023I: Server <userid> open for e-business-. The next step is to read the stdout.txt and stderr.txt files produced when starting up the toolkit. Normally, these files are under the websphere4/working directories. Use the messages in these files to see if the configuration file was read properly and no error messages are produced.

## Basic Data Management

The simplest way to view the contents of the DB2 database is to issue direct queries. Try the following:

```
db2
db2start (if needed)
connect to <database name> (database name should be your user ID)
select user_id from ifs_user
select * from ifs_user where user_id='ARF1001'
list tables
quit
```

Note that DB2 is not case sensitive except inside the quotes, which must be single quotes. As you can see, ,ifs\_user- is a table name and ,user\_id- is a column name. Using the ,select- statement, you can view all data in the tables provided with the enhanced simulation.

Using the ,update- and ,insert- statements, you could modify and augment the data in the tables provided although these updates will have to satisfy the rules of the database. See the *DB2 Command Reference* for syntax of these commands.

A recommended approach for modifying data is to create a DB2 script containing only the required changes. This script can then be rerun to restore the values to a known state after installing a new release and after experimenting with temporary changes. Please note, however, that future updates of the Toolkit will have to add columns as Gold message definitions change. Also, design changes in the Toolkit could significantly alter table layouts. This means that your data updates may have to be reworked to accommodate table definition changes.

## Data Management using Approach or Excel

A Lotus Approach or Microsoft Excel spread sheet on a Windows Operating System system can be used to view and update data in the DB2 tables on the AIX system with some additional setup.

On the Windows system, do the following:

1. Install Lotus Approach or Microsoft Excel.
2. Install IBM Universal Database Version 7 - DB2 Connect Personal Edition.
3. Using the DB2 Client Configuration Assistant, add the database on the AIX system using manual configuration, TCP/IP, LAN based, your host name and port 50000, your database name, and register for ODBC.
4. Using the spread sheet program, open with file type of ODBC and find your database name.
5. When viewing a table, overwrite an entry and press enter to change the value in the database on the AIX system. Use control-r to refresh the view from AIX.

Please note that future updates of the Toolkit will add columns as Gold message definitions change. Also, design changes in the Toolkit could significantly alter table layouts. This means that your data updates through the spread sheet will be lost when you move to a new version of the Toolkit because you will have to reload new tables from the new version.

---

## Development and Test Environment

Once installed, you have a development and test environment that allows you to create and modify web pages. You should be using your own server port so that other users can run their own copies of the server and the Toolkit without interfering with each other.

The development and test environment contains the complete CWS programs as released with the product with one exception. For the Toolkit, the CWS message server has been modified to simplify operations. In the Toolkit, the CWS message server communicates directly with the Financial Institute (FI) simulator instead of using MQ Series queues to communicate with the FI. Also, pin and password encryption is not used in the Toolkit.

---

## The Test Environment

As you develop and customize your CWS applications, you will probably like to quickly inspect the results to ensure the look-and-feel of the pages is correct.

At this point, it is likely you are not too concerned with performance or security, since you simply want to make sure that the user interface is efficient and easy to use.

Note that for the basic Toolkit, all the returned data is static in nature and can not be changed by your application. With the DB2 enhanced Toolkit, you can modify the tables to test additional aspects of your customized pages.

If you have problems restarting the web server, delete your trace and log files (rm /tmp/\*.your-queue-name, rm /tmp/\*.port-number), prior to starting the server. To start the server use the `ifs.war/tools/startweb` script.

**Note:** If you start your server as root, you might have problems later accessing the trace files. You should always run the server under your userid and not as root. You can verify your userid and the processes you have started by entering “**ps x**” under your name.

## CWS Trace and Log Facilities

The CWS contains several logging facilities which can be used to gather diagnostic information. These facilities are:

1. A message log
2. An internal execution trace
3. A log file for Gold messages

### CWS Message Log

The CWS message log file contains operational and error messages generated by the CWS. Messages such as startup messages and failure messages are written to this file.

The CWS message log file is identified in the CWS configuration file by the `wsp_log_file=` parameter. The AIX userid under which the CWS executes must have AIX file permissions to write to the message log file. Following is a sample parameter:

```
wsp_log_file=/tmp/WEBNAME.msglog
```

The CWS always writes messages to the message log. Logging of messages can not be turned off. CWS operations personnel should periodically review the message log and remove messages from it. The CWS always writes new messages to the end of the existing log. The CWS never deletes messages from the log.

### CWS Internal Execution Trace

The CWS contains a facility for tracing internal web server execution activity. This facility is useful for debugging internal web server problems. The internal execution trace contains trace records identifying entry and exit to key CWS functions.

CWS internal execution tracing is optional. The CWS configuration file contains parameters containing the name of the trace file, the maximum size of the trace file and whether or not internal tracing is to be active when the CWS is started. Following are sample parameters:

```
wsp_trace_flag=1
wsp_trace_file=/tmp/WEBNAME.trace
wsp_max_trace_size=100000
```

The above parameters indicate that trace should be active, the name of the trace file, and the maximum trace file size is 100000 bytes.

The CWS clears the contents of the trace file at CWS start time. The trace file wraps when the maximum trace file size is encountered.

Internal trace execution can be turned on and off during the execution of the web server. The `CMDSRV` utility can be used to turn it on and off. Following are the `CMDSRV` commands to turn tracing on and off:

```
cmdsrv trace system on  
cmdsrv trace system off
```

The `cmdsrv` utility program is installed in directory `ifs.war/tools/`. To run `cmdsrv`, the path to `ifs.war/bin` must be in the `LIBPATH`. You will also need to export the environment variable `WSP_SHAREDMEM_FILE` to point to the shared memory file. For example:

```
export LIBPATH=$LIBPATH:  
/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/bin  
export WSP_SHAREDMEM_FILE=/home/<userid>/CWS/logs/<userid>.sharedmem
```

It is recommended that production web servers do not run with internal execution tracing active for performance reasons. However, it is recommended that production web servers specify an internal trace file in the CWS config file to allow it to be turned on as needed.

## Viewing the Internal Execution Trace

The internal execution trace file is not in a human readable format. The **tracefmt** utility, contained in the `ifs.war/tools` directory, is provided to format the trace file into a readable format. To run `tracefmt`, the path to `ifs.war/bin` must be in the `LIBPATH`. It is invoked from an AIX console, and takes a single parameter, the name of the internal trace file to format. `Tracefmt` outputs formatted trace records. These trace records can be redirected to an output file for viewing with a text editor. Following is a sample invocation of `tracefmt` which pipes the output to a file called `mytrace`:

```
tracefmt /tmp/WEBNAME.trace > ~/mytrace
```

This assumes that the `tracefmt` program is contained in a directory in your AIX `PATH` variable.

## Gold Messages Trace

The CWS contains a facility for logging Gold messages which are generated or received by the web server. This facility is also referred to as the Gold trace facility.

Gold trace is very useful for debugging problems related to the content of gold messages. `Goldtrace` allows you to see the content of messages sent by the web server to the FI. It also allows you to see the content of messages returned to the web server from the FI.

It is often necessary to see the actual gold messages which are sent and received in order to diagnose why a web page may not contain the expected information. There may be problems in the web page because the web page does not pass the correct information when the request gold message is built. Or, there may be problems because the FI does not put the correct data in the response message. The gold trace facility allows you to see the content of these messages.

The tracing of gold messages is optional. The CWS configuration file contains parameters containing the name of the `goldtrace` file, an indicator whether or not gold tracing is to be active when the CWS is started, and a `userid` for whom gold tracing should be done. Following are sample parameters:

```
wsp_gold_trace_active=1  
wsp_gold_trace=/tmp/WEBNAME.goldtrace  
wsp_trace_userid=WEBNAME
```

The above parameters indicate that gold trace should be active when the web server starts, the name of the gold trace file, and the `userid` to gold trace.

The CWS clears the contents of the gold trace file at CWS start time. The CWS appends new gold trace records at the end of the file. The gold trace file size always grows when the CWS is tracing gold messages. The CWS config file variable `wsp_gold_trace_active` determines the state of the gold trace facility at CWS startup. The following values are supported:

```
0 -> gold trace not active  
1 -> gold trace active for all userids  
2 -> gold trace active only for userid specified in wsp_trace_userid
```

Gold tracing can be turned on and off during the execution of the web server. The **CMDSRV** utility can be used to turn it on and off. Following are the CMDSRV commands to turn tracing on and off:

```
cmdsrv trace gold on
cmdsrv trace gold off
```

It is recommended that production web servers do not run with gold tracing active for performance reasons. However, it is recommended that production web servers specify a gold trace file in the CWS config file to allow gold trace to be turned on as needed.

## Viewing the Gold Trace Messages

The gold trace file is in standard ASCII text format, and can be viewed with a text editor. The gold trace includes each request and response. It will start with a ,SRVSTART- pair of messages, a ,GETIFSPR- pair of messages, and numerous ,CODERFSH- pairs of messages prior to the first ,LOGON- pair of messages. The short names in Table 2 can be used to relate these gold messages to the enhanced simulation transaction names.

The following is an excerpt from a gold trace for an ,ACCTHIST- (Account History Transaction) Both the request and response include a message header and body. The structure of these messages and the codes used for values are defined in the file ifscstrs.h which is part of the Gold Message Translation Interface (MTI).

```
===== REQUEST MESSAGE =====
• Message Translation Interface: MTI/GOLD 99 R01 M02 * Message Header:
Message Info:
HeaderType: [1]
OriginatorType: [2001]
Message Mode: [5001]
Code Page: [850]
Request Type: [ACCTHIST] Interval Number: [0]
TimeStamp: [1998-10-23-14.57.03.000000]
GMTOffsetMinutes: [0]
Server Context: []
remoteAddress = [9.53.131.169]
Session Info:
UserId: [ARF1001]
ServerId: [donjames]
FI Id: [IBANKA]
Session Num: [2]
IFS Trace ID:
Server Instance Number: [1]
Service Request Reference: [5]
Return Code Info:
Comp Code: [0]
Reason Code: [0]
senseData = []

Extra Phase 2 Header Fields:
2      MsgHdrTg      (Id: [2019])
2.1    TrnUIdTg      (Id: [20069], Len: [1], Val: [])

Message Body:
1      IfsCnsmrIdTg ((IFS_CNSMR_ID) Id: [14])
1.1    cnsmr_id: [ARF1001]
1.2    ch_cnsmr_id: [ARF1001]
1.3    perm_user_id: []
2      IfsTxnRqstDtlTg ((IFS_TXN_RQST_DTL) Id: [59])
2.1    acct: (IFS_ACCT_ID)
2.1.1  acct_tp_cl: [CCA]
2.1.2  acct_nbr_id: [ARF1001CCA1]
2.1.3  acct_nck_nm: [ARF1001 CREDIT CARD 1]
2.2    txn_dt_rng: (IFS_DT_RNG)
2.2.1  strt_dt: (GOLD_DATE)
2.2.1.1 dt_val: []
2.2.1.2 dt_fmt_code_cl: [ISO]
2.2.2  end_dt: (GOLD_DATE)
2.2.2.1 dt_val: []
2.2.2.2 dt_fmt_code_cl: [ISO]
2.3    txn_amt: (IFS_AMT_RNG)
```

```

2.3.1          low_amt: [USD 0.0]
2.3.2          high_amt: [USD 0.0]
2.4            chk_nbr_rng: (IFS_CHK_RNG)
2.4.1          low_srl_nbr_id: [0]
2.4.2          high_srl_nbr_id: [0]
2.5            txn_prd_tp_cl: []
2.6            txn_tp_ind: [1]
2.7            txn_srce_val: []
2.8            fi_refr_nbr_id: []
2.9            txn_rec_cntl: (IFS_REPEAT_CNTL)
2.9.1          max_nbr_rec_val: [15]
2.9.2          1st_nbr_rec_val: [0]
===== RESPONSE MESSAGE =====
• Message Translation Interface: MTI/GOLD 99 R01 M02 * Message Header:
Message Info:
HeaderType: [1]
OriginatorType: [2001]
Message Mode: [5002]
Code Page: [850]
Request Type: [ACCTHIST] Interval Number: [0]
TimeStamp: [1998-10-23-14.57.03.000000]
GMTOffsetMinutes: [0]
Server Context: []
remoteAddress = [9.53.131.169]
Session Info:
UserId: [ARF1001]
ServerId: [donjames]
FI Id: [IBANKA]
Session Num: [2]
IFS Trace ID:
Server Instance Number: [1]
Service Request Reference: [5]
Return Code Info:
Comp Code: [0]
Reason Code: [0]
senseData = []

Extra Phase 2 Header Fields:
2            MsgHdrTg          (Id: [2019])
2.1          TrnUIdTg          (Id: [20069], Len: [1], Val: [])

Message Body:
1            IfsRespCodeTg ((IFS_RESP_CODE) Id: [44])
1.1          resp_code_cl: [000]
1.2          fi_proc_ind: []
2            IfsTxnRespSumTg ((IFS_TXN_RESP_SUM) Id: [58])
2.1          info_dt: (GOLD_DATE)
2.1.1        dt_val: [1998-10-23]
2.1.2        dt_fmt_code_cl: [ISO]
2.2          acct_id_dtl: (IFS_ACCT_ID)
2.2.1        acct_tp_cl: [CCA]
2.2.2        acct_nbr_id: [ARF1001CCA1]
2.2.3        acct_nck_nm: []
2.3          txn_info: (IFS_REPEAT_INFO)
2.3.1        more_f: []
2.3.2        tot_txn_mtch_val: [1]
2.3.3        tot_txn_snd_val: [0]
2.4          rpt_txn_resp_dtl: (IFS_TXN_RESP_DTL, Occurs:[1])
2.4.1        rpt_txn_resp_dtl: ((IFS_TXN_RESP_DTL), Occur: [1/1])
2.4.1.1      txn_orig_dt: (GOLD_DATE)
2.4.1.1.1    dt_val: [1997-10-02]
2.4.1.1.2    dt_fmt_code_cl: [ISO]
2.4.1.2      ldgr_bal_amt: [USD 0.00DR]
2.4.1.3      txn_tp_cl: [2]
2.4.1.5      txn_code_cl: []
2.4.1.5      txn_srce_val: []
2.4.1.6      txn_tp_desc: [CREDIT CARD TRANSACTIONS]
2.4.1.7      txn_pst_dtl: (IFS_TXN_PST_DTL)
2.4.1.7.1    txn_pst_dt: (GOLD_DATE)
2.4.1.7.1.1  dt_val: [1997-10-03]
2.4.1.7.1.2  dt_fmt_code_cl: [ISO]
2.4.1.7.2    fi_refr_nbr_id: [19971005]
2.4.1.7.3    srvc_provd_refr_id: [19971005]

```

```

2.4.1.7.4      txn_amt: [USD 125.14DR]
2.4.1.7.5      frgn_curr_amt: [ 0.0]
2.4.1.7.6      exch_rate: [0.0]
2.4.1.8        txn_chk_id: (IFS_TXN_CHK, Occurs:[0])
2.4.1.9        atm_txn_id: (IFS_ATM_TXN_ID, Occurs:[0])
2.4.1.10       txn_pymt_to_amt: (IFS_TXN_PYMT_TO_AMT, Occurs:[0])
2.4.1.11       ach_id: (IFS_ACH, Occurs:[0])
2.4.1.12       mortgage_dtl: (IFS_TXN_MORTGAGE, Occurs:[0])
2.4.1.13       dep_term_mat: (IFS_DEP_TERM_MAT, Occurs:[0])
2.4.1.14       dep_term_dep: (IFS_DEP_TERM_DEP, Occurs:[0])
2.4.1.15       trnst_rout_nbr_id: [0]
2.4.1.16       to_pncpl_amt: [ 0.0]
2.4.1.17       to_int_amt: [ 0.0]
2.4.1.18       to_escrw_amt: [ 0.0]
2.4.1.19       to_od_amt: [ 0.0]
2.4.1.20       comm_amt: [ 0.0]

```

## Selective Userid Gold Tracing

The gold trace facility provides the ability to trace all gold messages sent or received by the web server. It also provides the ability to only trace messages sent to or received by a selected userid. The selective gold tracing ability allows traces to be gathered for a selected userid that may be having problems without having to write all gold trace messages to the gold trace file.

Selective userid gold trace can be specified in the CWS config file via the following parameters. The example uses a userid of WEBNAME:

```

wsp_gold_trace_active=2
wsp_trace_userid=WEBNAME

```

Selective userid gold trace can also be enabled via the **cmdsrv** utility command:

```
cmdsrv trace gold userid WEBNAME
```

All gold tracing can be stopped with the following cmdsrv utility command:

```
cmdsrv trace gold off
```

Tracing gold messages for all users can be started with the following cmdsrv utility command:

```
cmdsrv trace gold on
```

## Reducing the Size of a Gold Trace

Another utility, **parefile** works with huge Gold traces. It takes 2 arguments:

- The name of the input file
- The search string to look for to start writing records out

The utility outputs *<input file>.out* which contains the gold trace starting at the first record where the search string was found, and an *<input file>.strip* which contains 1 line for each gold request and response item in the file. The type of transaction (response or request), userid, time, transaction name, completion code, reason code, and session number are formatted for each request or response block.

Another utility, **golduser** reduces the size of a gold trace. This utility will search a goldtrace and create a goldtrace output file with only the transactions for the specified user. The tool takes 2 input parameters:

- The name of the gold trace file
- The userid to search for

The output file is the input goldtrace file name with the userid to search for appended to the end.

---

## Code to Text Problems

Code To Text tables lookup problems can be diagnosed with a program called **qcodes**. It uses input data files obtained via the cmdsrv interface to verify or display text from user entered tag/code data pairs.

**cmdsrv codes dumpmti** - generates a binary image file of the currently loaded table as filename codetbl.mti

**cmdsrv codes dump** - generates a text file of the currently loaded table as filename codetbl.out

**qcodes codetbl.mti** - invokes interactive selection of Tag/Code and displays the text found

**qcodes codetbl.mti codetbl.out** - invokes verify mode, which checks that all tag/code identified in codetbl.out can be found in codetbl.mti

---

## Logging for FISim

There are currently two forms of logging for FISim: FISim Event Messages and Gold Request/Reply Dumps. The Gold Request/Reply Dumps are hard-coded to be in /tmp/<login-id>.fisgold. There is no flexibility with these log files. FISim always dumps the request and all replies. The FISim Event Messages are in /tmp/<login-id>.ifsdlog. FISim startup messages are always logged until a GOLD Request is processed. The /tmp/<login-id>.<cnsmr\_id> is also available for review. The login-id is the environment variable for LOGIN and the cnsmr\_id is retrieved from the Gold Request.

When a Gold Request is processed, in the enhanced toolkit, flags are retrieved from the DB2 table, IFS\_SYSTEM. These flags help to define the Logging Level. Logging can be turned off if all flags in the IFS\_SYSTEM table are set to zero. That is logging is turned off once the first Gold Request has been processed. The initial startup events will be reported in /tmp/<loginpid>.ifsdlog. Gold Request/Reply messages are always logged.

---

## Debug Process

This may be the most time consuming phase after you have created new pages.

## Debugging

Before you start debugging, you should ensure that all the necessary processes are running.

Type “ps x”

Something similar to the following will be displayed:

```
PID      TTY STAT  TIME COMMAND
39790    -  A      0:26 /usr/WebSphere4/AppServer/java/jre/bin/java -classpath
41128    -  A      0:03 websrvr /usr/WebSphere4/AppServer/installedApps/userid
56904 pts/0 A      0:00 ps x
```

...

**Potential Problems:** If any of the processes above are missing from your list it is an indication that you have a configuration problem. Possible problems are:

- The configuration of the CWS under the WebSphere Administrative Console has errors. Check the Event Message log for possible errors.
- The Interactive Financial Systems Web Module under Enterprise Applications has not been deployed properly. Reinstall the Enterprise Application (perhaps using the Install Enterprise Application Wizard) and check the Event Message log for errors.
- The Interactive Financial Systems Web Module under the Application Server has not been installed properly. Check the Application Server for proper settings in the properties tabbed notebook.
- The trace files (\*.<userid>), or server files (\*.your-port-number) are owned by a different user, and your server can not access them.
- The server has no access to the Web application. Try changing the directory permissions to 755.
- Hardware problem (insufficient memory, problems with disk segments, etc.).



## Debug Functions

There are several functions available in the CWSAPI to assist the developer during the test phase of the project. By using functions around code that you may be having trouble with, you can get additional hints in isolating the problem.

The Debug functions display information during development to help you troubleshoot problematic areas in your code. The Debug functions are included in the CWSAPI library.

These debug functions write output to the same browser windows to which normal output is directed, so they should be removed or disabled prior to deploying production code. Also, you should retest your programs to ensure that you have not accidentally removed necessary code. In order to use these functions, you should become familiar with the CWSAPI document.

## Debugging Tips

**Errors Returned by the CWSAPI:** If you are getting an error from a CWSAPI call, you should inspect the error-log and trace files. Errors in the CWSAPI are probably related to:

- Calling a function with the incorrect number of parameters
- Calling the function with an incorrect data type

A common mistake is to send an integer data type instead of an character string value. For example, specifying value=1 instead of value=-1-. If you get a numeric error without a meaningful string associated with it, locate the error number in the CWS include files to find its description.

**Gold Definitions in the CWS:** The include file ,cws/bld/lgmdef.h- contains Gold values used by the CWS. Inspection of this file can be helpful in identifying internal values that may be reported in error and diagnostic messages.

The **lgmdef.h** include file is segmented in the following areas:

- **Definitions and Value Assignments for Gold Fields.** For example:

```
#define Atm_IfsRespDt1Tg 1 /* StructNode */
#define Atm_IfsRespDt1Tg_RespCodeC1Tg 2 /* LeafNode */
#define Atm_IfsRespDt1Tg_FiRfrNbrIdTg 3 /* LeafNode */
#define Atm_IfsRespDt1Tg_SrvCPrvdRfrIdTg 4 /* LeafNode */
#define Atm_GMSeIfDefiningInfoTg 5 /* StructNode */
#define Atm_GMSeIfDefiningInfoTg_GMSDFieldDefTg_repeatCounter 6 /*LeafNode /
#define Atm_GMSeIfDefiningInfoTg_GMSDFieldDefTg 7 /* LeafNode */
...
```

- **Gold Top-Level Structure Definitions.** For example:

```
#define LogonTxnTg 1
#define LogoffTxnTg 2
#define IfsPingTxnTg 3
#define AccountHistoryTxnTg 4
#define UserProfileTxnTg 5
#define BpProblemTxnTg 6
#define AcctNicknameTxnTg 7
...
```

- **Gold Transaction Names.** For example:

```
#define GMREQ_GENUNDLV 1
#define GMREQ_QSELFDEF 2
#define GMREQ_ADDACCT 3
#define GMREQ_BALINQ 4
#define GMREQ_ACCTDTL 5
#define GMREQ_ACCTHIST 6
#define GMREQ_NICKNAME 7
...
```

If you get a diagnostic message stating that certain ,transaction x- with ,atom y- is in error, or that can not be accessed, it is likely that you have an error on a particular CWSAPI request. In that case, look through the **lgmdef.h** file to locate the name of the transaction and the corresponding atoms.

**Error in Constructor:** If you are not getting an error in the CWSAPI but not all the attributes are set on the object created, you are probably having a problem in the CWS Programs (wsp.so and websrvr).

Run-time error in the CWS programs are probably related to:

- Your data files being from a different version of the Gold Data Definition.
- Your library path might be pointing to an old version of cws.so.

**Miscellaneous Debugging Tips:** Debugging requires certain knowledge about how the Enterprise Server manages your application. After you make changes and compile your code, your application must be restarted. Normally you do not need to shutdown the server, but simply use the Application Manager to restart your web application.

Before you make changes to the source code, you should study the flow of the web page. Use your client browser to run the original, unmodified web applications and inspect the page dependencies. Pay particular attention at the URL entry field where the HTML file name is going to be displayed, and the links that will be taken from this page, including possible help text where needed. Once you understand the flow and dependencies of the page you can start making changes. As a good programming practice always keep copies of the original work for reference.

As you modify your pages, you may want to identify your changes for easy reference and future maintenance. Instead of deleting unneeded sections of code, you should comment out those sections.

Also, the consistency of style when making changes will pay back many times over, especially when new versions of the CWS are installed.

## Support Pages

Some of the CWS Customization APIs require HTML pages for interaction with a client. These pages will typically not be seen by end-users under normal operation.

**error.html:** This page is used by the Raise Exception function for message display at the client.

---

## Gold Transactions Serviced By Enhanced Simulation

Table 4. Transactions Serviced by DB2 Based Simulation

Request Message	Short Name	Meaning
ACCOUNT_HISTORY_TXN	ACCTHIST	Account History Inquiry
ACCT_DTL_TXN	ACCTDTL	Account Detail Inquiry
ACCT_NICKNAME_TXN	NICKNAME	Change Account Nickname
ACCT_PINPSWD_TXN	PINPSWD	PIN / Password Change
ACCT_TAX_DTL_TXN	TAXDTL	Account Tax Detail
ADD_ACCT_TXN	ADDACCT	Add Account to Service Profile
ADD_BILLER_TXN	ADDBILR	Add Biller
ADD_BUSINESS_INVOICE_TXN	ADDBSINV	Add Business Invoice Transaction
ADD_FIX_PYMT_TXN	AFIXPYMT	Add Fixed Amount Recurring Payment Transaction
ADD_FIX_TFR_TXN	AFIXTFR	Add Fixed Amount Recurring Transfer
ADD_PAYEE_TXN	ADDPAYEE	Add Payee
ADD_SRVC_TXN	ADDSRVC	Add Service
ADD_VAR_PYMT_TXN	AVARPYMT	Add Variable Amount Single Payment
ADD_VAR_TFR_TXN	AVARTFR	Add variable Amount Single Transfer
AUTH_ACCT_TXN	AUTHACCT	Authorize Online Registration Accounts
BAL_INQ_TXN	BALINQ	Balance Inquiry
BILLER_DTL_TXN	BLRDTL	Biller Detail
BILLER_INQ_TXN	BLRINQ	Biller Inquiry
BP_PROBLEM_TXN	BILLPROB	Report Bill Pay Problem

Request Message	Short Name	Meaning
BP_VERIF_TXN	BPVERF	Request Bill Pay Verification
BUSINESS_INVOICE_INQR_TXN	BSINVINQ	Business Invoice Inquiry Transaction
CC_DSPUT_TXN	CCDSPUT	Report Credit Card Dispute
CHECK_COPY_TXN	CHKCOPY	Check Copy Order
CHK_BOOK_TXN	CHKBOOK	Check Book Reorder
CNCL_STOP_CHK_PYMT_TXN	CNCSTPMT	Cancel Stop Check Payment
CHNG_BILLER_TXN	CHNGBILR	Change Biller
CHNG_CC_LMT_TXN	CHGCCLMT	Request Change Credit Card Limit
CHNG_FIX_PYMT_TXN	CFIXPYMT	Change Fixed Amount Recurring Payment
CHNG_FIX_TFR_TXN	CFIXTFR	Change Fixed Amount Recurring Transfer
CHNG_PAYEE_TXN	CHGPAYEE	Change Payee
CHNG_PYMT_INSTN_TXN	CPYMTINS	Change Payment Instance
CHNG_VAR_PYMT_TXN	CVARPYMT	Change Variable Amount Single Payment
CHNG_VAR_TFR_TXN	CVARTFR	Change Variable Amount Single Transfer
CNSMR_BP_AUDT_INQR_TXN	CBPAUDIN	Consumer Bill Pay Audit Inquiry
CNSMR_MSG_CFRM_TXN	CNSMCFRM	Consumer Message Confirmation
CNSMR_SSN_START_TXN	CONSSEVT	IFS Consumer Session Start Transaction
CODE_REFRESH_TXN	CODERFSH	Code Refresh
CR_APPL_TXN	CRAPPL	Small Business Credit Application
CSR_CNSMR_STRT_TXN	STRCNSMR	CSR Consumer End
CSR_CNSMR_END_TXN	ENDCNSMR	CSR Consumer Start
CSR_CUST_PRFL_TXN	CSGNSMR	CSR - Consumer Profile Request
CSR_FORCE_USR_TXN	FORCEUSR	IFS CSR Force User
CSR_SHRT_CNSMR_PRFL_TXN	CSSCNSMR	CSR Short Consumer Profile
CSR_UPDT_CUST_PRFL_TXN	CSUCNSMR	CSR - Update Customer Profile
CUST_FI_EMAIL	CSTEMAIL	Customer Reply or Communications to FI
DEL_BILLER_TXN	DELBILR	Delete Biller
DEL_BUSINESS_INVOICE_TXN	DELBSINV	Delete Business Invoice Transaction
DEL_FI_MSG	DELMSG	Delete a FI Initiated Message
DEL_FIX_PYMT_TXN	DFIXPYMT	Delete Fixed Amount Recurring Payment
DEL_FIX_TFR_TXN	DFIXTFR	Delete Fixed Amount Recurring Transfer
DEL_PAYEE_TXN	DELPAYEE	Delete Payee
DEL_PYMT_INSTN_TXN	DPYMTINS	Delete Payment Instance
DEL_VAR_PYMT_TXN	DVARPYMT	Delete Variable Amount Single Payment
DEL_VAR_TFR_TXN	DVARTFR	Delete Variable Amount Single Transfer
DISCLOSURE_ACPT_TXN	ACPTDSCL	Agreement Online Registration Disclosure
DISCLOSURE_INQ_TXN	DISCLINQ	Disclosure Inquiry
EBILL_ACT_DLVRY_TXN	EBACTBLR	Activate Electronic Biller Delivery
EBILL_BILL_INQ_TXN	EBBILINQ	Electronic Bill Inquiry
EBILL_BILLER_INQ_TXN	EBBLRINQ	Electronic Biller Inquiry
EBILL_CHG_STATE_TXN	EBCHGST	Change Electronic Bill State
EBILL_DEACT_DLVRY_TXN	EBDEABLR	Deactivate Electronic Biller Delivery
EBILL_BILLER_DLVRY_INQR_TXN	EBDLINQ	Electronic Biller Delivery Inquiry
EXCH_RATE_INQR_TXN	EXCHRATE	Exchange Rate Inquiry Transaction
FI_PROFILE_TXN	FIPROF	FI Profile

<b>Request Message</b>	<b>Short Name</b>	<b>Meaning</b>
GET_CHG_CNSMR_ID_TXN	GCHCNSMR	Get Changeable Consumer ID
GET_CLHS_TP_TXN	GCLHSTYP	Get Clearinghouse Type Table
GET_CNSMR_ID_TXN	GCNSMRID	Get Consumer ID
GET_DEST_CLHS_TXN	GDESTCLS	Get Destination FI Clearinghouse Type Table
GET_DEST_FI_TXN	GDESTFI	Get Destination FI table
GET_DEST_CLHS_PRFL_TXN	GDSTCLSP	Get Destination FI Clearinghouse Profile
GET_EUR_EXCH_RT_TXN	GETEURRT	Get Euro Exchange Rates Inquiry Transaction
GET_EXCH_CAL_TXN	GEXCHCAL	Get Exchange Calendar
GET_FI_MSG	READMSG	Read FI Initiated Message
GET_IFS_PROFILE_TXN	GETIFSPR	Get IFS Profile Inquiry
GET_INV_SVC_DTL_TXN	GINVDTL	Get Investment Service Details
GET_NO_PAY_ADV_C_TXN	GNPAYADV	Get No-Pay Advice
GET_SEC_DESC_TXN	GSECDESC	Get Security Descriptions Transaction
GET_TFR_DT_ADJ_ADV_C_TXN	GTFRDATV	Get Transfer Date Adjustment Advice
GET_UNDLVD_MSG_LIST_TXN	CCUMSGLS	Get Undelivered List
GET_UNDLVD_MSG_TXN	CCUMSG	Get Undelivered Message
IFS_PING_TXN	IFSPING	IFS Ping
IFS_REAUTH_TXN	REAUTH	Reauthentication
IFS_UNDELIVERED_TXN	UNDELIV	Undelivered Transaction
INQ_SRVC_TXN	INQSRVC	Service Inquiry
INV_ACCT_INFO_TXN	INVACCT	Investment Account Info
INV_ACCTDTL_TXN	IACCTDTL	Investment Account Detail
INV_ACCTHST_TXN	IACCTHST	Investment Account History
INV_ORD_STOCK_TXN	INVORDS	Investment Order - Stock Transaction
INV_ORD_CANC_TXN	IORDCANC	Investment Order Cancel Transaction
INV_ORD_CAN_CHG_TXN	IORDCHG	Investment Order Cancel/Change Transaction
INV_ORD_STAT_INQ_TXN	IORDINQ	Investment Order Status Inquiry
LIST_FI_MSG	LISTMSG	List FI Initiated Messages
LOAN_ADV_TXN	LOANADV	Loan Advance
LOAN_REPYMT_TXN	LOANRPYM	Loan Repayment
LOGOFF_TXN	LOGOFF	Logoff
LOGON_TXN	LOGON	Logon
MASS_LOGOFF_TXN	MASSLGOF	Mass Logoff
OBO_LOGON_TXN	OBOLOGON	On Behalf Of (OBO) Logon
PAY_DCSN_RQST_TXN	RPAYDCSN	Request Pay No Pay Decision
PAYEE_DTL_TXN	PAYEEDTL	Payee Detail
PAYEE_INQ_TXN	PAYEEINQ	Payee Inquiry
PAYMENT_INQ_TXN	PAYINQ	Payment Instance Inquiry
PYMT_DEFN_INQR_TXN	PYMTDEFI	Payment Definition Inquiry
REM_ACCT_TXN	REMACCT	Remove Account From Service File
REM_SRVC_TXN	REMSRVC	Remove Service
REV_BP_RQST_TXN	RVBPRQST	Reverse Bill Pay Request
RSET_CNSMR_SCRTY_TXN	RSCNSCRT	Reset Consumer Security
SALES_CPY_TXN	SALESSLP	Sales Slip Copy Order
SELF_REGIS_TXN	SELFREGST	Self Registration

Request Message	Short Name	Meaning
SERVER_START_TXN	SRVSHUTD	Server Startup
SERVER_STOP_TXN	SRVSTART	Server Shutdown
SETTL_2_ACCT_TXN	SETTLACT	Settle to Account
SRCH_CNSMR_PRFL_TXN	SRCHCNSM	Exchange Rate Inquiry Transaction
STOP_CHK_PYMT_TXN	STPCKPMT	Stop Check Payment Transaction
STOP_CHK_PYMT_INQR_TXN	STPMTINQ	Stop Check Payment Inquiry
STOP_RCUR_DRFT_TXN	STPRCPMT	Stop Recurring Draft
STMT_COPY_TXN	STMTCOPY	Statement Copy Order
STMT_PERIODS_TXN	STPDSINQ	Request Statement Period Information
TERM_DEP_INQR_TXN	TDEPINQR	Term Deposit Inquiry
TERM_DEP_RQST_TXN	TDEPRQST	Term Deposit Account Request
TERM_DEP_OPN_TXN	TDEPOPEN	Term Deposit Account Open
TFR_PROBLEM_TXN	TFRPROB	Report Transfer Problem Transaction
TFR_INQ_TXN	TFRINQ	Transfer Inquiry
UPDT_CH_CNSMR_ID_TXN	UPCHCNSM	Update Changeable Consumer ID
UPDT_CLHS_TP_TXN	UCLHSTYP	Update Clearinghouse Type Table
UPDT_CNSMR_DTL_TXN	UCNSMRDL	Update Consumer Details
UPDT_EXCH_CAL_TXN	UEXCHCAL	Update Exchange Calendar
UPDT_DEST_CLHS_TXN	UDESTCLS	Update Destination FI Clearinghouse Type Table
UPDT_DEST_FI_TXN	UDESTFI	Update Destination FI Table
GET_INV_SVC_DTL_TXN	GINVDTL	Get Investment Service Details
UPD_INV_SVC_DTL	UINVDTL	Update Investment Service Details
UPDT_SRVC_TXN	UPDTSRVC	Update Service
USER_PROFILE_TXN	USERPROF	FI Consumer Profile Inquiry
WIRE_TFR_CNCL_TXN	WTFRCNCL	Cancel Repetitive / Standing Wire Transfer Model
WIRE_TFR_INIT_TXN	WTFRINIT	Initiate Repetitive Wire Transfer
WIRE_TFR_INQR_TXN	WTFRINQR	Repetitive/Standing Wire Transfer Model Inquiry

## Predefined Responses

Transaction responses are generated in one of the following ways:

1. If there is a *predefined response file* indicated for the particular transaction in the *user configuration file*, then the contents of the specified response file will be returned as the Gold Response. The response file can be either a binary file containing Gold bitstream data, or a text file containing a description of a Gold Response in the required format.
2. If using *fisimtkd* and no predefined response file was indicated, the internal business logic is used to generate a Gold response based on the contents of the database.
3. If using *fisimtkm* or if no logic has been implemented for a particular transaction, the MQWeb library will be used to return a ,canned- Gold response.

This section deals with responses of the first type, and shows how to configure predefined responses for the CWS Developer s Toolkit.

In order to use predefined responses, the user submitting the request must have an associated *user configuration file* that maps response files to transaction types. These files are described in more detail below.

## Important Points To Consider

- Response files can contain either Gold bitstream data or a **goldmine** input. If the response file is a text file, the file will be converted to a Gold bitstream format by passing it to an internal **goldmine** converter. This can delay the delivery of the response message. The default timeout value in the **goldbank** configuration file can be increased accordingly to avoid timeout errors. For more information on **goldmine** and **goldbank**, please reference the Appendix C, ,Gold Message Interface Tool (GMIT)-.
- The response file contains the whole response, including the message header. Any fields left unspecified are not returned in the response.
- If any errors are encountered during the processing or delivery of a response file, a 247 BRC will be returned in the IFS\_RESP\_CODE structure instead, indicating Technical Difficulties

## Environment variables

The FISIM\_PATH environment variable can be set using a directory name or a colon-separated list of directory names to indicate the directories to be searched when the system looks for user configuration files. Path names should be fully qualified to avoid ambiguity.

For example, if FISIM\_PATH=/home/fisim/test:/home/fisim/old/test, then /home/fisim/test will be searched first, followed by /home/fisim/old/test.

The header file, *ifscstrs.h* is also required when generating responses via data files. This header file is usually located in */usr/fis/bin*. The FISIM\_PATH environment variable is also used to locate this file. The directory or directories specified in FISIM\_PATH will be searched first. If *ifscstrs.h* cannot be found, the Toolkit will crash.

## User Configuration Files

There must be one user configuration file for the user submitting the transaction. This is a text file that maps the user's response files to their respective transactions, and names the directory containing all of the response files. A sample user configuration is included in */usr/fis/bin* called *fis.ARF1001.config.SAMPLE*.

A response file will be used as the transaction response if both of the following conditions are met:

1. A configuration file for the user is located in one of the directories specified in FISIM\_PATH, as described above.
2. The configuration file contains a *transaction line* for the specified Gold transaction (described below).

If either of these conditions is not met, then the transaction response must be constructed from the backend business logic or the MQWeb library.

Configuration files are named *fis.USERID.config*, where *USERID* must match the user ID specified in the SessionKey.userID in the header of the incoming Gold Request Message. The file name *fis.config* is reserved for use with transactions (e.g., SRVSTART or CODERFSH) where the user ID is not specified.

User configuration files may be edited while the Toolkit is running. A configuration file will automatically be reloaded when the timestamp of the file indicates that it has been modified. (Files accessed over network-mounted file systems may have unreliable timestamps.)

The contents of a user configuration file follow the following rules:

Blank lines and lines starting with a *#*- character (used for comments) are ignored.

- The first non-blank, non-comment line in the file must be the name of a directory. This indicates the location of all of the user's response files. This line must begin with a whitespace character, and the directory name should be fully qualified.
- A line that does not begin with a whitespace character is a *transaction line*. It consists of the short name of a transaction in all capital letters. This line is followed by one or more *file specification* lines, which indicate response files for this type of transaction.

- A line that begins with a whitespace character is a *file specification line*. It contains the name of a file (which must be present in the previously-defined directory).
- There may be multiple file specification lines for a single transaction type. When the Toolkit needs to generate a response for a transaction of this type, it will rotate through the different responses in the order that they appear in the file.

A sample user configuration file appears below:

```
# Sample UserConfig file (fis.ARF1001.config)

# Comment line

# Search Directory for response files

/home/fisim/test1001

# First Transaction (LOGON) - Alternate responses

LOGON

ARF1001.LOGON.resp.dat

ARF1001.LOGON_FAIL.resp.dat

# (LOGOFF) - Always respond with same response

LOGOFF
ARF1001.LOGOFF.resp.dat
# (BALINQ) Cycle responses through CHECKING/SAVINGS/IRA
BALINQ
ARF1001.BALINQ_CHECKING.resp.dat
ARF1001.BALINQ_SAVINGS.resp.dat
ARF1001.BALINQ_IRA.resp.dat
```

## Response Data Files

All response files for a particular user are stored in a single directory. As seen above, the user configuration file indicates the location of the response files and maps the response files to their respective transactions.

The *fis/testdata* directory contains response file templates for every transaction supported by the current MTI level. The names of all of the files in this directory end with the suffix, *.resp.dat*. The simplest approach to creating new response data files is to copy a response file of the appropriate type from this directory, edit its contents as necessary, and place the file in the user's response directory, taking care to ensure that the file name matches the entry in the user configuration file.

These response data files contain two sections; a Header and a Message Body. The fields in the Header are the Gold header fields defined by IFS. *<Header>* indicates the beginning of the Header section, and *</Header>* indicates the end of the Header section. The following are examples of fields that may be edited in the Header section:

Table 5. Response Header Editable Fields

<b>&lt;Header&gt; Section</b>	<b>Function</b>
RequestType	Defines the request/response type for the Gold Message being built. For example: <i>RequestType LOGON</i>
SessionKey.fi	Defines the FI bank name used for testing. For example: <i>SessionKey.fi IBANKA</i>
SessionKey.userID	Defines the name of a consumer's userid. For example: <i>SessionKey.userID ARF1001</i>

The Message Body contains the structures and fields as defined in the Gold Message Responses. *<Message\_Body>* indicates the beginning of the Message Body section, and *</Message\_Body>* indicates the end of the Message Body section. The structures in the Message Body are composed of tag-value pairs representing the various fields in the format:

<Tag Name> <Tag Data> # <Tag Description> (<Data Type> <Length>)

Comments are always preceded with #. Do not place comments within structure level tags unless they follow a tag-value pair.

## Repeat Items

Structures that can have multiple occurrences are defined in the data files with the comment value ,... (Repeat Item)-. Each occurrence is referred to as a single structure instance. By default the data files are generated with only a single occurrence of Repeat Item structures. Data files that contain Repeat Items structures can be modified to include multiple structure instances of a Repeat Item structure. To include multiple structure instances of a Repeat Item structure the structure instance(s) must be enclosed by a <Repeat\_Structure> and </Repeat\_Structure> statements with the number of structure instances specified on the <Repeat\_Structure> line.

Using the ADDBSINV (Add Business Invoice Transaction) data file as an example the structure tag InvLineDtlTg is defined as a Repeat Item, as shown:

```
<Structure> InvLineDtlTg          # ... (Repeat Item)
  InvLineNbrIdTg 1                # Invoice Line Number (char [12])
  <Structure> InvLineAmtTg        # ...
    <Structure> AmtTg              # Amount
      DcmldataValTg 6835          # Decimal Data (char [15])
      Dcm1PrecValTg 2             # Decimal Precision (long)
      DrCrIndTg 1                 # Debit/Credit Indicator (long)
</Structure>
IsoCurrCodeTg USD                 # ISO Currency Code (char[3])
</Structure>
InvLineMemoNmTg memo txt          # Line Memo Text (char[80])
</Structure>
```

To create multiple structure instances of the Repeat Item structure edit the data file as shown:

```
<Repeat_Structure> 2              # repeat count of 1 or greater must be specified
  <Structure> InvLineDtlTg          # ... (Repeat Item)Structure Instance 1
    InvLineNbrIdTg 1                # Invoice Line Number (char [12])
    <Structure> InvLineAmtTg        # ...
      <Structure> AmtTg              # Amount
        DcmldataValTg 6835          # Decimal Data (char [15])
        Dcm1PrecValTg 2             # Decimal Precision (long)
        DrCrIndTg                  # Debit/Credit Indicator (long)
  </Structure>
  IsoCurrCodeC1Tg USD              # ISO Currency Code (char[3])
  </Structure>
  InvLineMemoNmTg memo txt          # Line Memo Text (char[80])
  </Structure>
  <Structure> InvLineDtlTg          # ... (Repeat Item)Structure Instance 2
    InvLineNbrIdTg 2                # Invoice Line Number (char[12])
    <Structure> InvLineAmtTg        # ...
      <Structure> AmtTg              # Amount
        DcmldataValTg 2503          # Decimal Data (char [15])
        Dcm1PrecValTg 2             # Decimal Precision (long)
        DrCrIndTg 1                 # Debit/Credit Indicator (long)
  </Structure>
  IsoCurrCodeC1Tg USD              # ISO Currency Code (char[3])
  </Structure>
  InvLineMemoNmTg memo txt          # Line Memo Text (char[80])
  </Structure>
</Repeat_Structure>
```

The Repeat Count specified on the <Repeat\_Structure> line must equal the number of Structure Instances you add, otherwise an error will occur informing you that there are too many or too few structure instances defined. The structure instances must be defined inside the <Repeat\_Structure> and </Repeat\_Structure> statements.

## Goldmine

A response file can be converted into Gold binary format by running it through **goldmine** first, and using the resulting .gold file as the response file. In any case, it is a good idea to use **goldmine** to verify your source



file syntax prior to running any tests. For more information on **goldmine**, please reference Appendix C, ,Gold Message Interface Tool (GMIT)-.

---

## Appendix A. Web Server Program (WSP) Error Messages and Codes

---

### WSP Component Return Codes

The following return codes can be set by the WSP Component:

Code	Error Message
1	Security failure - Unable to Logon
2	Security failure - Authentication Failure
3	Security failure - Auto-registration failure
4	Security failure    UserID revoked

---

### WSP PFM Return Codes

Code	Error Message
10	PFM Initialization failed
20	MTI Add-header failed
30	MTI Add-CStruct failed
40	MTI Failure writing message to trace log
50	MTI Put-Message-to-Bitstream failure
60	MTI Memory allocation failure
70	MTI Terminate-message failure
80	Name to Index failure
1000	MQ Send/Receive failure

### Reason Codes Used in the Gold Header for the PFM Server

Code	Error Message
12001	Bad originator type
12002	Invalid FI name
12003	Invalid message mode
12004	Not a logon request
12005	One of the following failures: MTIGetMessageElement failed to get the IFS_CNSMR_DATA_ID structure MTIGetCStructCopy failed to get the IFS_CNSMR_DATA_ID structure MTIAddHeader failed to add updated header to request MTIPutMessageToBitstream failed for request MTIInitializeMessage failed to for gold response from CC MTIGetMessage from bitstream failed on gold response MTIGetHeader failed on gold response MTIGetMessageElement failed to get the consumer profile detail structure MTIGetCStructCopy failed to get the consumer profile detail structure MTIGetMessageElement failed on the IFS_CNSMR_ID structure MTIGetCStrucCopy failed to get the IFS_CNSMR_ID structure
12006	Userid in header does not match consumer id in body (logon) The changeable consumer ids didn t match (request)
12007	name2index failed on request type
12008	MQ timed out waiting for a response
12009	MQ returned a Gold response message length of 0
12010	Response from CC is not for the request we sent
12011	Invalid ip address

**Code    Error Message**

---

12013	The call to encryptRSA encryption of the password failed
12014	User is already logged on when a logon transaction is encountered
12012	The unchangeable consumer ids didn t match

---

**Web Server Return Codes****Code    Error Message**

---

100	User previously logged on <b>Note:</b> Error return codes 101 - 110 are used by the certificate validation
101	Certificate Revoked
102	No Certificate
103	Invalid Signature on Certificate
104	Certificate is Expired
105	CRL is Expired
106	Certificate Invalid Extensions
107	Certificate No Extensions
108	No CRL Distribution Point
109	Certificate Has Expired
110	CRL Invalid Signature
200	Maximum number of users logged on
300	Verify failed
400	Security failure
500	Reauthentication failure
600	reauthcpw_required
699	Session already in progress
700	Missing certificate
701	Encryption failure
702	Bad Post data
703	Missing cookie
704	Password PIN mismatch
705	User rejected cookie
706	Memory allocation error
707	Certificate encode failure
708	CA certificate cannot be retrieved from directory server
709	Record cannot be retrieved from directory server
710	Invalid certificate
711	Certificate suspended
712	Missing CRL
800	The LOGON request cannot be completed. The service is unavailable.
801	The request cannot be completed. No service is available.
802	The request cannot be completed. AIC IPC services are not available to the web server (lock failed).
803	The request cannot be completed. AIC IPC services are not available to the web server (mutex failed).
804	The request cannot be completed. AIC IPC services are not available to the web server (msgsend failed).
805	The request cannot be completed. AIC IPC services are not available to the web server (msgrcv failed).
806	No response to the request.

---

## CWSAPI Request Codes

Code	Error Message
------	---------------

1000	Bad Message Index Internal error codes
2000	Invalid start flag
2001	Invalid shutdown flag

---

## Gold Manager Codes

Code	Error Message
------	---------------

1001	Gold Request - Bad input buffer
1002	Gold Request - Missing parameter
1003	Gold Request - Parameter type incorrect
1004	Gold Request - Invalid request Flag
1005	Gold Request - Invalid Request
1006	Gold Request - Incorrect Number of parameters
1007	Gold Request - Invalid Transaction number
1008	Gold Request - Repeat type is invalid
1009	Gold Request - Invalid MTI structure
1010	Gold Request - Incorrect type for Repeat Count
1011	Gold Request - Cache Transaction not allowed
1012	Gold Request - FI Transaction not allowed
1013	Gold Request - MTI error (Wrong element class)
1014	Gold Request - Internal error ((bad user session)
1015	Gold Request - Parameter references a negative index
1016	Gold Request - Invalid On-behalf-of response
1017	Gold Request - MQ Send/Receive message is zero length
1018	Gold Request - Gold Dictionary mismatch
1019	Gold Request - Too many Repeat-structures
1020	Gold Request - Invalid Repeat Argument
1021	Gold Request - Invalid Integer type
1022	Gold Request - Invalid Repeat String Length
1023	Gold Request - Memory allocation failure
1024	Gold Request - MTI error processing request
1025	Gold Request - Internal error (GM_GMREQ_PARENT_CHAIN_NOT_FOUND)
1026	Gold Request - Null transaction name
1027	Gold Request - Cache transaction name required
1028	Gold Request - Number too short
1029	Gold Request - Malformed repeat

---

## Codes Processing Java Request

Code	Error Message
------	---------------

2001	Internal Error (unable to allocate memory) - Deprecated
2002	Too much data passed
2003	Java request - Unsupported data type
2004	parameter1_wrong_type
2005	parameter2_wrong_type
2006	parameter3_wrong_type
2007	bad_gmjo_acronym - Deprecated
2008	bad_gmjo_error_code - Deprecated
2009	unexpected_data_type

<b>Code</b>	<b>Error Message</b>
2010	bad_logon_cookie
2011	arg_count_error
2012	MaxField length error

---

## Internal Errors in Gold Manager - Locating Atoms

<b>Code</b>	<b>Error Message</b>
3001	Locate index is too short
3002	Locate index is negative
3003	Locate index is too large
3004	Unable to find toplevel Structure
3005	Locate contains a bad atomid
3006	Invalid toplevel structure
3007	Invalid repeat index
3008	copy unlike atoms
3009	copy unknown type
3010	Put atom of unknown type
3011	Put atom of the wrong type
3012	Memory allocation failure
3013	Get numeric atom was not type long
3014	No space

---

## Gold Manager Error Codes While Creating the Output Response Buffer

<b>Code</b>	<b>Error Message</b>
4001	Locate error
4002	Invalid atom type
4003	Invalid atom need long
4004	Index array overflow
4005	Index array underflow
4006	Buffer overflow
4007	Single multi error
4008	Multi underflow
4009	Code lookup error
4010	Filter format error

---

## Internal Conversion Errors (GMNIBBLE)

<b>Code</b>	<b>Error Message</b>
5001	Output area too small
5002	Input string not even
5003	Invalid nibble char

---

## Gold Manager Initialization Codes

<b>Code</b>	<b>Error Message</b>
6001	Initialization failure
6002	FI name too long
6003	web server name too long
6004	Logon userid too long
6005	Logon Memory Allocation failure

---

Code	Error Message
------	---------------

---

6006	No data returned on Logon
6010	Logon response too short
6011	Logon response bad eyecatcher
6012	Logon response bad errflag
6012	Logon response cws error
6014	Logon response 0000 no BRC
6015	Logon response 0000 bad BRC
6016	Logon response CC too short
6017	Logon response CC no BRC
6018	Logon response CC bad BRC

---

## Errors While Logging Off

---

Code	Error Message
------	---------------

---

6020	No data returned
6021	Invalid session
6021	RSTCACHE Invalid Session
6022	Invalid request

---

## Extra Error Codes

---

Code	Error Message
------	---------------

---

6050	Missing parameter
6051	Wrong data type
6052	Malloc failure
6053	Unsupported request
6054	Incorrect length for transaction id
6055	Bad session in transaction id
6060	Required number mismatch
6061	Transaction name not found

---

## Appendix B. IFS Web Page Performance/Response Time Tuning Recommendations

---

### Introduction

This appendix is an introduction and primer for IFS Web page performance. It is intended to assist those designing or implementing IFS pages primarily. Factors that influence the other components of response time are identified but not the focus of this appendix. The information contained herein is based upon IFS production experience.

---

### Environment

#### Service Provider

This service provider environment is defined by the online real time processing of the banking transaction by the bank's in-house systems or the systems of a third party. In a given implementation a financial institution may have both connections to the in-house systems for banking and connection to a third party for services such as bill presentment. A request for a transaction is received from the web client browser via the IFS web server. It is then converted to a Gold message and passed to the core controller instance. The core controller performs its necessary processing (such as logging or routing) then passes the message to the service provider for processing. A reply is generated by the service provider, sent through the core controller to the web server, formatted into a web page, and sent along to the client browser.

#### Hosted Applications

This hosted application environment is similar in most respects to the service provider with the difference being in the processing site of the Gold message. There are three current applications hosted at IFS:

- The strip file application to respond to core banking.
- The bill payment engine for IFS Bill Payment.
- The messaging server for secure bank mail.

The core controller sends the Gold message to the hosted application and receives its reply from it. In a given implementation a financial institution may have both connections to the service provider for selected transactions and connections to hosted applications for services such as core banking or messaging.

---

### IFS Response Times

#### Response Time Considerations

The IFS ,cloud- is defined for purposes of response time as starting at the web server and running through the core controller until the message exits onto an MQ channel (either to a hosted application or a service provider). Production experience has shown that well designed and implemented pages can achieve an elapsed time, round trip, of only a few seconds on average in the IFS cloud. Therefore, the response time as seen by the consumer is dominated by other parts of the end-to-end transaction.

---

## Other Response Time Factors

### Networks

#### The Web

The World Wide Web is by definition an internetwork, that is a network of networks. The elapsed time that it takes to send data between two points is known as latency. Latency on the web will vary based on several factors, most of which are outside the control of IFS or the bank:

- Local POTS (Plain Old Telephone Service) - Most consumers access the web via a dial in connection to an ISP. The quality of this connection will affect the overall response time and consumer experience. To date, with several localized exceptions, this has not been a problem. Most consumers seem able to obtain a reasonably reliable connection at or above 28.8Kb/sec which has proven sufficient for the production banking pages. Note that when the quality of the connection is a problem, it can be difficult and time consuming to track the problem through the web and demonstrate its source.
- ISP - The number of lines, modems, routers and router speeds can also be significant. Again, with only localized exceptions, ISP has not been a problem to date. As with POTS, when these problems do arise they are difficult and time consuming to track down.
- ISP Connections - Many ISPs purchase connections to the Internet backbone from the long distance providers and then connect the consumer via a local modem pool. If this connection is not sufficient the consumer can experience widely varying response times based upon the number of other users connected at the ISP and their level of traffic. Typically, this happens during peak usage hours.
- Backbone Bandwidth and Routing - The Internet backbones themselves are owned by different companies and maintained to differing levels of performance. Routes through the Internet can dramatically affect the response times for consumers. The IBM Global Network maintains custom links and connections points with several other high speed backbone providers in order to ensure consistent bandwidth and predictable routes.

### What Does All This Mean?

The Internet will likely never be able to provide entirely consistent and speedy response times in the style (private corporate networks) to which many of us have become accustomed. However, good page design, careful system construction, and tuning can help to create a fast, reliable online banking application.

### Backend Latency

In the case of online banks (that is, service providers for core banking) there is additional latency introduced by the connection from the core controller to the service provider and by the elapsed time to process the transaction within the service provider. In the experiences of production banks to date, this back end processing/network time (called backend latency) is a large component of consumer response time. Thus it can be seen that back end network performance and data center throughput/response time will contribute to the consumer experience. Banks planning for production should focus on this area if they expect to provide speedy response times for their consumers.

---

## Page Design

As detailed in an earlier section, the IFS infrastructure can deliver dynamic banking pages with consistently short response times. There are several areas of page design, however, that influence the consumer perception of response time.

### Graphics

Graphics can enhance the overall consumer experience. They can reinforce the bank's brand, assist with page navigation, and advertise new services. However, graphics add to the overall complexity and latency of the pages and will degrade response time when compared to a text only implementation. So choosing



graphics for your pages is a matter of trading off speed for the other benefits. You will have to balance these considerations in order to provide a high-performing and usable consumer experience.

Some factors to consider:

**Caching** - Many ISPs provide a system cache for frequently referenced URLs. In a region with a lot of consumers referencing popular sites, perceived response times can be quite fast since the consumer is really hitting a local cache and not loading off the actual Internet server. Additionally, the browser will cache these URLs to disk.

However, SSL encrypted sessions cannot be cached (outside of the browser) because the data is encrypted and the cache has no way of knowing what page/URL is being transferred. So, if your browser cache does not contain the required item, in an SSL session, it must always load from the Internet banking server.

Furthermore, Netscape Communicator (4.x) only caches encrypted pages in the memory cache. Thus every time you start Communicator it will need to reload the GIFs and client side JavaScript from the server. It will then cache them in memory for the duration of your session.

Internet Explorer (4.x and 5) will cache encrypted pages to both the memory and disk cache. Thus it will not have to reload the GIFs and client side JavaScript each session. This is the default behavior and can be changed on the advanced configuration screen. Note, that from a security perspective, Netscape's behavior is superior.

- **Start-up Delay** - Most informational web pages are static. That is, the page is defined once as a file and does not change very often. The HTML is downloaded, the GIFs are listed inline, and the browser can start loading them right away.

IFS banking pages, in contrast, are dynamic. When you request a transaction, the Gold message must complete the transaction before the dynamic page can be built. Thus the browser will not see the GIF URLs, and cannot start loading them, until after the transaction is complete and the dynamic HTML has been delivered to the browser. Thus dynamic pages with lots of GIFs will load slower than the same page implemented as static. The other benefits of dynamic pages outweigh this limitation, but it is a good idea to know about it and to keep the number of GIFs to a minimum.

The page designer can improve response time by implementing ,early emit-. If you place the informational and navigational items at the top of the page and the dynamic data (obtained from a Gold transaction) below them; you can flush the server page buffer just before you begin the Gold transaction. To do this you build and output all the page HTML up to the dynamic portions, then use the Java flush() operative to begin sending the page to the browser.

The browser can therefore ,see- the GIFs and will begin loading them and rendering the page to the screen in parallel with the cloud and legacy processing of the Gold message. This overlapping of the page load/render and transaction processing will reduce total page elapsed time.

Additionally, the consumer perception of system responsiveness is dramatically improved because the page will ,flip- and begin to render almost instantly compared to the otherwise several second delays (with the hourglass on screen) before the consumer perceives that something is happening.

Experience has shown that consumers dislike the hourglass wait (they seem to become uncertain and worry that something has gone wrong) but once the page begins to render, are more forgiving of the time taken (they can see incremental progress). Note that if you use this technique you **CANNOT** issue a redirect once the flush() is done since the HTTP header has been sent.

- **Size** - It may seem obvious to mention this, but keep the size of GIFs to a minimum to reduce network latency during transfer.
- **Color palette selection** can dramatically affect GIF size and therefore latency. A 20x100 pixel GIF with 4-bit color (16-color palette) takes about .35 seconds over a 28.8 modem connection to send the data. Use a color from the 16-bit color palette (64K colors) that same GIF is going to take 1.4 seconds to send. If you just have to have that ,perfect- color from the 24-bit palette (16 Million colors) its going to cost you 2 seconds for the GIF to cross the modem.

Consider that only graphic designers have large color corrected monitors. The rest of us mostly make do with 14 inch PC monitors with limited color accuracy so we are not going to see the color the way you do anyway.

- **JPEG s** - Can you beat the color palette size problem by using a JPEG? Possibly, if you use a mostly solid color. But JPEG includes poor compression, which means you can lose information/resolution when the image is uncompressed. This is a real concern when perfect image restore is needed (for example, medical imaging), but it is not a problem with banking web pages.

So, in theory, the transform can allow a larger image to be compressed and sent as a smaller package. In practice the efficacy of the transform depends on the image. A solid color image or one without a lot of color variation compresses nicely. However, mottled color or an image with significant bleed or color blending compresses poorly.

- **Quantity** - Quantity might not be as obvious as size but is very important. Browser levels prior to Netscape 4.x and Internet Explorer 4.x used HTTP Version 1.0 (or, 0.9). Many users (AOL, for example) are still on these versions. These browsers will open one TCP socket for EVERY URL needed on a page (each GIF is a new URL).

The overhead of opening an SSL TCP socket is not trivial and often exceeds the overhead and latency of transmitting the GIF data itself. Furthermore, browsers are asked to behave well in the standards documents. Good behavior consists of never opening more than four sockets to a single server concurrently. Most browsers comply with the good behavior guidelines (intended to limit network storms and avoid choking the Internet). This means that if you have lots of GIFs on the page, they will be loaded sequentially in batches of four. This is compounded by the short duration of the socket connections, preventing TCP pacing from taking effect. So pages with a large number of (even small) GIFs will load more slowly than those with only a few (even if somewhat larger) GIFs, and will contribute to clogging the network in the process.

Later browser levels (4.x and above) implement HTTP V1.1 or better. They still limit themselves to only four concurrent connections, but can make multiple requests over each connection. Thus much of the penalty of establishing multiple TCP connections is reduced. However, GIFs will still load sequentially in batches of four.

- **Rendering** - Graphics (and other advanced page elements like frames, etc.) take the browser a long time to render and display as compared to plain text.
- **Recommendation: Reuse GIFs.** Use GIFs on more than one page (they only get loaded once per session) so that subsequent pages will load and render much more quickly. NEVER use a different directory path for the same GIF on different pages; the browser cache will NOT recognize them as the same and will load a second copy.
- **A ,text only-** setting is a good idea for advanced users and those who prefer speed. It will reduce their time at the browser and may lead to overall improved customer perceptions.
- **Always use ALT attributes on images** (other than purely decorative) so users can understand the page if they have graphics turned off.
- **Always pre-scale graphics** and explicitly provide correct values for height and width attributes.

## Multiple Transactions on a Page

Gold messages issued by the web server are processed synchronously. Some web pages have been coded to issue more than one Gold message in order to construct pages with complex combinations of consumer financial data. As these messages are processed one after the other, the total elapsed time is cumulative. Thus four messages issued from a single web page (with an average IFS cloud time of 2 seconds and a backend latency of 5 seconds) will take 4x the average Gold time (or 28 seconds in this example) before the page completes. While it may not be possible to entirely avoid this type of page, they should be reduced to an absolute minimum. Frankly, the consumer may prefer to navigate a small set of rapidly responding pages rather than wait a minute for that page with all information consolidated. Performance is as much perception as reality.

This is not to say that one should not code such pages when it makes sense. The Logon sequence is often used to gather some general information from multiple sources (such as pending messages and payments). Another example is a bill payment screen where you may allow the consumer to enter data for several payments and then process them all with one click. Just know and test the effects on response time when making such tradeoffs.

## Large Messages

Several of the Gold messages can contain large amounts of data, and they can also contain replicated structures (account and payment history are examples). These messages add powerful capabilities to the IFS system. However, the overhead of processing extremely large messages can overwhelm the benefits of cramming all this data into one transaction. It is recommended that banks limit the number of history items that are viewed on one screen and provide a ,more- or ,next- function to scroll through additional data (with a subsequent transaction). This will provide faster perceived response time to the consumer and help the web server to maintain consistent throughput. In practice we have found 15 history items to be a good tradeoff. Enough items are supplied to the consumer so that they do not have to reissue too many requests, and the system can handle this number in an efficient manner.

## Web Caching

The web server contains functions for caching consumers data during the duration of their session. Page designers should take advantage of these functions and avoid purging the cache. Production experience has shown that inefficient use of the cache or excessive purging of the cache can severely degrade throughput and response time as well as drive up the Gold message rate.

## Stay on Budget

Game designers must strictly keep to a polygon budget in order to stay within response time constraints. If you want snappy banking pages you must set a budget per page (total bytes) and stick to it.

The slowest link in the response time chain is the consumer modem. If we assume that we want our pages to perform well for a consumer on a 28.8Kbps connection, then we use the following formula to calculate the effects of page size:

$$ML = (B \times 10) / 28800 \text{ (or replace with your typical speed)}$$

ML is modem latency

B is total page size in bytes

10 is used as bits per byte because it helps account for TCP overhead, packet headers, etc.

28800 is the modem speed in bits/second

So if we have a page with 10K bytes (all data, HTML, GIFs, etc. combined) it will take approximately:

$$(10000 \times 10) / 28800 = 3.5 \text{ seconds to cross the modem.}$$

But change that page to 35K and it takes 12 seconds.

Now let's see what effect this has on the consumer. Let's assume that the IFS cloud is processing your page/transaction in approximately 2 seconds and your backend latency is 4 seconds. Then the consumer response time (not including browser render time, Internet latency, etc.) is:

$$ML + Backend + Cloud \text{ or } 3.5 + 4 + 2 = 9.5 \text{ seconds for a 10K byte page}$$

But,  $12 + 4 + 2 = 18$  seconds for that 35K byte page.

## Client Side JavaScript

Client side JavaScript can be a wonder in helping users navigate the pages and in validating their input rapidly. It can also destroy the performance of the system if overused. It must be used in moderation.

- Because the consumer can disable client side JavaScript, do not use it to implement essential functions.
- Client side validation can be performed to ensure that all required fields are filled in. This validation is a good use of client side JavaScript.
- Field types, that eliminate the need for validation (like pull downs or radio buttons), are preferred to those, that require validation (text fields or other input fields that must be checked to insure that the consumer has entered the correct type of data).

- Remember that size counts. Track the size of the JavaScript you write, and stay within your page budget. All those bytes have to be sent as data to the browser. If you have to use client side JavaScript follow the rules:

Always put it into a file. Never use embedded JavaScript (unless it is VERY SMALL). Embedded JavaScript is sent with the page each time it is referenced. So if the consumer hits the back button or has to enter transactions many times on the same page (pay bills, etc.) you get to pay the latency freight every time the page is referenced. If you put it in a separate file, then it is only downloaded once per session. Subsequent references to the logic by the page will fetch from the browser cache.

Reuse routines from page to page. Once the JavaScript file is loaded into the cache it can be fetched directly.

Split up the routines. Several smaller files with JavaScript routines are preferred to one big file. It is easier to put it all in one big basket, but this will tend to take a long time to load and probably will be needed on the very first page. Thus the initial consumer interaction with the system is poisoned.

Use the right URLs. The browser will recognize the file as cached and not reload it if the fully resolved (qualified) URL is the same. Do not play tricks with alternate or symbolic links on the server file system if they are going to show up as different URLs to the browser and preclude it from hitting the cache.

Scrub your files. JavaScript is an interpreted language. As such the entire source file is sent to the browser. This includes any formatting or white space you included to make your code readable. The interpreter doesn't need this. So strip out all white space and readability formatting, etc. to minimize the bytes being sent over the network. Think this is trivial? Well we have found in production that scrubbing reduces the typical file from 40-60%. Not bad when you consider many of these JavaScript files are 10-40K bytes in length to start with.

## Testing Pages

Do not test the pages only on your workstation which is connected via a high-speed backbone connection. If you do so, you are testing the look but not the feel of the system. Instead, test the pages using a connection at the typical speed you expect your consumers to attain. This can vary from a 28.8 dial up modem to a high-speed cable (or other fast) connection. You then experience what the consumer will experience. Run the pages normally (with graphics on), then shut down and restart the browser (flush the cache also if Internet Explorer) then turn graphics and JavaScript off in the browser to see how the pages run without all the pretty frills. This test provides a good indicator of how page design is impacting system performance.

---

## Page Implementation

### Gold Manager Caching

The CWS Gold Manager contains caching routines to improve the efficiency of Gold transaction processing and the resulting response times. Typically, a consumer session will need various items of information more than once. Examples would include account details (the accounts, rules, and so forth that apply to the consumer) and history data (the consumer might select various filter parameters to control display of his transaction history). It would be quite inefficient, expensive, and time consuming to perform a Gold transaction to the legacy system each time an item of data was needed.

The Gold Manager Cache is, to a great extent, under control of the page implementers who must ensure proper and efficient cache management. Proper cache management is achieved by using the Gold manager option flags passed when a transaction is requested. These are:

- FI - Get results from the FI legacy system.
- CACHE - Get results from cache.
- CACHEFI - If results exist in cache, use them; otherwise, send transaction to legacy system.
- CACHEFIDATA - Similar to CACHEFI but used by HTML Native Formatter to allow multiple hits with differing sort and select filters.
- FIDATA - Get results from FI but save for future use by CACHEFIDATA.

**Important Note:** This discussion is general and not intended to be exhaustive. There are several different Gold Manager Cache types and variations on these option flags. Please see the IFS CWS documentation for detailed information on proper cache usage and options.

## Web Server Caching

The previous section described the use of the Gold Manager to cache messages for optimum efficiency and response time. This section discusses caching in the actual web server. There are times when caching is necessary to ensure good response times, efficient Java processing, and maximum system capacity. Typically caching is needed when you want to maintain information for a consumer session and make it available to some or all pages.

In Java, caching is implemented using the `lfsHttpSession` object, which allows you to save information during a session. Caching is a fast and reliable way to insure good page performance.

There is a second condition when the Web Server Cache should be used. Some Gold data might be needed quite often. The Gold Manager Cache is excellent for reducing unnecessary calls to the legacy systems. But even when a Gold transaction is fetched from the Gold Manager Cache the data must be processed through Java object instantiation via a byte array. For trivial messages, which are not referenced too often, this is not a problem; however, if the transaction contains many fields to be processed and if it is referenced frequently, then server capacity and page response time can be hurt.

A perfect example is the consumer profile. Returned at Logon, it typically contains a wealth of information. As part of page generation and input validation, some page designers have seen the need to refer to some of the profile fields on nearly every page. As services are added, options fielded, and account types and quantities increase, the consumer profile can grow to many fields. Thus, the overhead of creating attributes/objects for each field every time it is fetched from the Gold Manager can dominate page response time.

Caching the user profile resolves this difficulty.

### Bill Presentment Logos

The bill presentment function provides the ability to display the electronic biller's logo on the consumer page. To minimize the overhead involved in handling these binary images, the CWS has implemented a cache strategy for these logos. At CWS boot time, the CWS will fetch the set of supported logos via an EBILLERINQ message. It will then save the individual logos to the file system, using the path specified in the CWS configuration and the logo names specified in the EBILLERINQ response message.

When coding a consumer bill presentment page, you can include the electronic biller logo by specifying its name (returned in the BILLERINQ message) with the correct path from the CWS configuration, on an HREF tag.

---

## Appendix C. Gold Message Interface Tool (GMIT)

---

### Introduction

This appendix assumes the reader is familiar with the Gold Message Standard. The Gold Message Interface Tool (GMIT) provides programs to generate Gold messages and then send these messages to a Financial Institution (FI) server. The two programs included in GMIT; **goldmine** and **goldbank** are summarized below.

Table 6. *goldmine* and *goldbank* Functions

Program	Function	Input Files	Output Files
Goldmine	Generates Gold message dump and Gold bit stream files.	One <request>.dat	<request>.out (An MTI dump of the generated Gold message.) <request>.gold (Gold bit stream file)
Goldbank	Send the Gold message via MQSeries, in the form of a bit-stream, to the FI server.	One <request>.gold (Gold bit stream file generated by goldmine)	<request>.gresp (Gold bit stream response file) <request>.fgresp (MTI dump file which is a readable dump of the Gold response from the FI server.)

GMIT provides a set of sample request data files. Each data file represents a single request. For example, the following table shows you a subset of the request types with their descriptions, followed by the associated request data file name:

Table 7. *Subset of Request Data Files*

Request type	Description	Request data file name
ABLRVLD	Add biller account	ablrld.dat
ACCTDTL	Account details	acctdtl.dat
ACCTHIST	Account history	accthist.dat
ACCTINFO	Account information	accinfo.dat
ADDACCT	Add account	addacct.dat

### The Test Data Flow

The test data flow for GMIT is shown below.

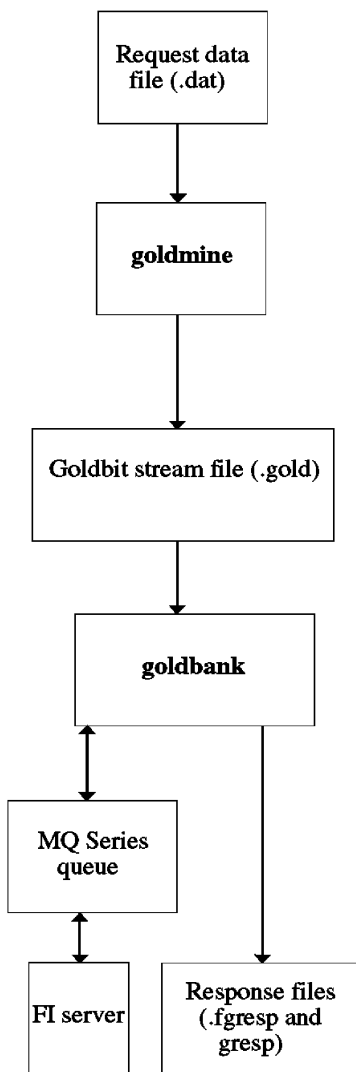


Figure 6. GMIT Program Flow

## Glossary

Acronym	Explanation
BE	Back-End
CWS	Customizable Web Server
CWSAPI	CWS Application Interface
FI	Financial Institution
IFS	Interactive Financial Services
GMIT	Gold Message Interface Tool
MTI	Gold Message Translation Interface
SMIT	System Management Interface Tool
TDM	Transaction Delivery Manager

---

## Getting Started

### Prerequisites

To use GMIT, you need the following hardware and software:

## Hardware

- Entry level IBM e-server pSeries or RS/6000 system
- LAN attachment for remote user access

## Software

- AIX Operating System (Version 4.3.3 or newer, which includes TCP/IP)
- MQSeries Version 5.2 (configured using the System Management Interface Tool (SMIT). If you do not have SMIT installed, you can install it from:  
<http://www.software.ibm.com/ts/mqseries/txppacs/txpm1.html>
- Financial Institution Simulator (FISim), if back-end is not available.

## Installing GMIT

The GMIT filename is delivered in the format of gmitv1r6m1\_yymmdd.tar.Z. Where, yy=year, mm=month and dd=day. To install GMIT:

- Create a new directory.
- Copy gmitv1r6m1\_yymmdd.tar.Z into that directory.
- Type `zcat gmitv1r6m1_yymmdd.tar.Z | (cd new directory; tar xvf - )`

## MQSeries and GMIT

The following instructions assume GMIT and the FI server reside on different machines. In these instructions, the machine on which GMIT resides is called the **source machine** since it is the source of Gold message requests. The FI server machine is called the **destination machine**. Before using GMIT, install MQSeries and System Management Interface Tool (SMIT) on both the **source and destination** machines. To install MQSeries, see [Beta TBD].

On each machine, you define:

- A queue manager
- A transmission queue for sending messages, which is a local queue
- A queue for receiving messages, which is a local queue
- A remote queue (for the queue manager on the other machine)
- A sender channel
- A receiver channel
- A trigger process for sending messages, which is a local queue

## Adding a Queue Manager

Once you install MQSeries, add a **queue manager**. This queue manager will provide queuing services to applications and manage the queues that belong to it. The queue manager also ensures that messages are put on the correct queue. Each queue manager must have a unique name. Multiple users may share a queue manager. Only the first user on each machine needs to add the queue manager.

In the following steps, *qmgrname* (all lowercase) is the name you assign to the queue manager. *userid* (all lowercase) is your userid. These must be different on the source and the destination machines. Use SMIT to add a queue manager as follows:

1. Type *smi t*. Press the ENTER key.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queue Managers*.
5. On the *Work with MQSeries Queue Managers* screen, select *Add a queue manager*.
6. On the *Add a queue manager* screen, in the *Queue Manager Name* entry field, type *qmgrname*.



**Note:** Names in MQSeries are case sensitive.

In the *Description* entry field, type *qmgrname* queue manager for GMIT.

In the *Dead Letter Queue* entry field, type SYSTEM.DEAD.LETTER.QUEUE. Press Enter.

SMIT displays the *Command Status* screen. At first, the screen will read **Command: Running**. Then it will read **Command: OK** and MQSeries queue manager created and objects created or **Command: failed** and indicate why it failed. Do not complete the steps for adding a default object if the Command: failed was displayed.

**Note on dead letter queues:** A dead letter queue is a local queue where messages are routed if they cannot be routed to their correct destination. If you do not specify a dead letter queue, application program errors may cause channels to be closed. For example, if an application tries to put a message on a queue on another queue manager but a failure occurs, the channel is stopped and the message remains on the transmission queue. Then, other applications are blocked from using that channel for their messages. If you have a dead letter queue, the undelivered message is simply put on the dead letter queue at the receiving end, leaving the channel and its transmission queue available.

## Starting a Queue Manager

Once you add the queue manager, start it through SMIT as follows:

1. Type *smi t*. Press the ENTER key.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queue Managers*.
5. On the *Work with MQSeries Queue Managers* screen, select *Start a queue manager*.
6. On the *Queue Manager Name* screen, move the cursor to select the queue manager you want to start: *qmgrname*.

SMIT displays the *Command Status* screen. At first, the screen will read **Command: Running**. Then it will read **Command: OK** and MQSeries queue manager started or **Command: failed** and indicate why it failed.

## Adding Default Objects

Once you start the queue manager, add default objects to it through SMIT as follows:

1. Type *smi t*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queue Managers*.
5. On the *Work with MQSeries Queue Managers* screen, select *Add default objects to a queue manager*.
6. On the *Queue Manager Name* screen, move the cursor to select the queue manager you want: *qmgrname*.

SMIT displays the *Command Status* screen. At first, the screen will read **Command: Running**. Then it will read **Command: OK** and display several messages about MQSeries queues, channels, and processes created or **Command: failed** and indicate why it failed.

## Defining Queues

After you start the queue manager and add the default objects, add the queues you need to be able to use GMIT programs. In MQSeries, there are three general types of queues:

- **Local queues**
- **Remote queues**
- **Alias queue objects**

A **local queue** belongs to the queue manager to which the application is connected. GMIT requires that you add two local queues: one for transmitting messages and one for receiving messages.

A remote queue belongs to another queue manager. This queue is defined as a local queue to that queue manager. The information that you specify when you define a remote queue object allows the local queue manager to find the remote queue manager so that any messages that are destined for the remote queue manager get to the correct queue manager. Before applications can send messages to a remote queue, you must define a **transmission queue** and channels between the queue managers.

An alias queue object allows applications to access a queue by referring to it indirectly in MQI calls. An alias queue object is not a queue, but an object that you can use to access another queue. When an alias queue object is used in an MQI call, the name is resolved to the name of either a local or a remote queue at run time. This way, you can change the queues that your application uses without changing the application, you just have to change the alias queue object definition to reflect the name of the new queue. GMIT does not require alias queues, but you may need alias queues to map goldbank queue names to the queue names used by your FI server.

## Adding Local Queues

To use GMIT, add two local queues as follows:

- A queue for transmitting messages
  - A queue for receiving messages
1. Type smit. Press ENTER.
  2. On the *System Management* screen, select *Communications Applications and Services*.
  3. On the *Communications Applications and Services* screen, select *MQSeries*.
  4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
  5. On the *Work with MQSeries Queues* screen, select *Local Queues*.
  6. On the *Local Queues* screen, select *Add a local queue*.
  7. On the *Queue Manager Name* screen, click the cursor to select the local queue manager to be added: *qmgrname*.
  8. On the *Add a local queue screen*, in the *Queue Name* entry field, type the transmission queue name *userid .xmit*. This is the queue that will send your application messages to the remote queue manager for processing.
  9. In the *Description* entry field, type GMIT transmission queue.
  10. Toggle the *MQGET* operations and the *MQPUT* operations fields to *ENABLED*.
  11. In the *Usage* field, press TAB to toggle to *XMITQ* (transmission queue).
  12. Toggle the *Trigger Enabled* field to *yes*.
  13. In the *Initiation Queue* field, type *SYSTEM.CHANNEL.INITQ*.

14. In the *Trigger Process* field, type `userid_CHL_PROCESS`.

15. Press ENTER.

SMIT displays the *Command Status* screen. At first, the screen will read **Command: Running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('<USERID>.xmit') +
: DESCR ('GMIT Transmission queue') +
: USAGE (XMITQ)
```

Use the same SMIT screens to add the queue for receiving messages. Unlike the transmission queue, this queue will use the defaults for all fields except the queue name.

16. Type a **Queue Name** of `userid.receive`. Press Enter. The queue is added.

## Adding the Remote Queue

Add the definition of the remote queue, which is used for the transmission to the remote queue manager.

1. Type `smit`. Press ENTER
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Remote Queues*.
6. On the *Remote Queues* screen, select *Add A Remote Queue*.
7. On the *Queue Manager Name* screen, click the cursor on the desired queue manager name: *qmgrname*.
8. On the *Add a remote queue* screen, in the *Queue Name* entry field, type the name of the queue manager for the other machine.

In the *Name of remote Queue Manager Entry* field, type the name of the queue manager for the other machine.

In the *Transmission queue entry* field, type `userid.xmit`. Press Enter.

SMIT displays the *Command Status* screen. First, the screen reads **Command: Running**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: QREMOTE ('<USERID>')
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

## Adding Alias Queues

1. Type `smit`. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.

5. On the *Work with MQSeries Queues* screen, select *Alias Queues*.
6. On the *Alias Queues* screen, select *Add an alias queue*.
7. On the *Queue Manager Name* screen, click the cursor to select the alias queue manager to be added: *qmgrname*.
8. On the *Add an alias queue* screen, in the *Queue Name* entry field, type the transmission queue name *userid.xmit*.
9. In the *Like existing queue* field, select from the list of queues.
10. In the *Description* entry field, type GMIT transmission queue.
11. Toggle the *MQGET* operations and the *MQPUT* operations fields to **ENABLED**

SMIT displays the *Command Status* screen. At first, the screen will read **Command: Running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
  : QALIAS ('<alias_name>.xmit') +
  : LIKE ('<USERID>.xmit') +
  : DESCR ('GMIT')
```

## Adding Channel Definitions

1. Type *smit*. Press ENTER
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Channels*.
5. On the *Work with MQSeries Channels* screen, select *Add/Change/Show/Delete channel definitions*.
6. On the *Add/Change/Show/Delete channel definitions* screen, select *Sender channel definitions*.
7. On the *Sender channel definitions* screen, select *Add a Sender channel*.
8. On the *Queue Manager Name* screen click the cursor on the name of the queue manager, *qmgrname*.
9. SMIT displays the *Add a sender channel* screen. In the *Channel Name* field, type a name composed of the local *qmgrname*, a slash, and the remote *qmgrname*.  
  
In the *Connection Name* field, type the IP address and MQSeries port number for the remote (other) machine in the format *nnn.nnn.nnn.nnn(pppp)*. The default port number for MQSeries is 1414.  
  
In the *Name of transmission queue* field, type *userid.xmit*. Press Enter.
10. Back on the *Add/Change/Show/Delete channel definitions* screen, select *Receiver channel definitions*.
11. On the *Receiver channel definitions* screen, select *Add a Receiver channel*.
12. On the *Queue Manager Name* screen click the cursor on the name of the queue manager, *qmgrname*.
13. SMIT displays the *Add a receiver channel* screen. In the *Channel Name* field, type a name composed of the remote *qmgrname*, a slash, and the local *qmgrname*.
14. Press OK.

## Adding Process Definitions

1. On the *MQSeries* screen, select *Work with MQSeries Process Definitions*.
2. On the *Work with MQSeries Process Definitions* screen, select *Add a process definition*.
3. On the *Queue Manager Name* screen, click the cursor on the queue manager name, *qmgrname*.
4. On the *Add a process definition screen*, in the *Process Name* field, type `userid_CHL_PROCESS`.

In the *Application Type* field, type `UNIX`.

In the *User Data* field, type the sender channel name defined above. Press Enter.

## Starting a Channel Initiator

1. On the *MQSeries* screen, select *Work with MQSeries Channels*.
2. On the *Work with MQSeries Channels* screen, select *Start/Stop channel initiators*.
3. On the *Start/Stop channel initiators* screen, select *Start channel initiator*.
4. On the *Queue Manager Name* screen, click the cursor on the queue manager name, *qmgrname*

To use *inetd* to start channels, configure two files.

1. Add a line in the `/etc/services` file:

```
MQSeries 1414/TCP
```

2. Add a line in the `/etc/inetd.conf` file:

```
MQSeries stream tcp nowait mqm /usr/lpp/mqm/bin/amqcrsta amqcrsta -m qmgrname
```

## Displaying Channel Status

1. On the *MQSeries* screen, select *Work with MQSeries Channels*.
2. On the *Work with MQSeries Channels* screen, select *Display channel status*. Display current status.
3. On the *Queue Manager Name* screen, move the cursor to queue manager name, *qmgrname*.
4. On the *Queue Name* screen, click the cursor on the queue name for which you want to display status.

If the status of the channels looks incorrect, either the sender or receiver machine is set up incorrectly, the process definitions are incorrect, or the channels are not started.

---

## goldmine

**goldmine** generates a Gold bit stream file by reading input from a request data file. The Gold bit stream file is then used as input to the **goldbank** program. The request data file, which ends in *.dat*, contains information for building a Gold bit stream file and is explained in more detail later in this section. The complete set of request data files is shown in ,Request Data Files-.

## Modifying Request Data Files

Request data files contain information needed to construct a Gold bit stream, which **goldmine** writes to a Gold bit stream file that ends with *.gold*. The request data file contains two sections:

- <Header>
- <Message\_Body>

The <Message\_Body> section contains <Structure> sections.

Table 8. Request Data File Section Contents

Section	Contains
<Header>	Gold header fields defined by IFS.
<Message_Body>	<p>Structures and fields defined by IFS for each request type.</p> <p>Structures are tag-value pairs representing the fields defined by IFS. For example:</p> <pre> &lt;Message_Body&gt; &lt;Structure&gt;  IfsCnsmrDataIdTg # IFS CONSUMER ID              CnsmrDataIdItemTg ARF1010 # Consumer Id      char  30               CnsmrDataEncryptTg 0          # Consumer Encrypt Flag &lt;/Structure&gt; &lt;/Message_Body&gt; </pre> <p>If a field is not needed, set the value to <i>NULL</i> or leave blank. Represent any nested or additional structures by &lt;Structure&gt; and &lt;/Structure&gt; tags.</p>

**<Header>:** You can modify these fields in the <Header> section:

Table 9. <Header> Record Functions

<Header> record	Function
FileName	<p>Controls the name of the output files added by <b>goldmine</b>. The file names generated by <b>goldmine</b> start with the name given on this record and end in either:</p> <ul style="list-style-type: none"> <li>• .out for the Gold message file</li> <li>• .gold for the Gold bit stream file</li> </ul> <p>For example, this FileName record creates the files logon.gold and logon.out:</p> <pre>FileName logon</pre>
RequestType	<p>Defines the request type for the Gold message being built. See Request Data Files for a complete list of request types. For example:</p> <pre>RequestType LOGON</pre>
SessionKey.fi	<p>Defines the FI bank name used for testing. For example:</p> <pre>SessionKey.fi IBANKA</pre>
SessionKey.userID	<p>Defines the name of a client userid. For example:</p> <pre>SessionKey.userID ARF1010</pre>

**<Message\_Body>:** You can modify the <Message\_Body> records within the <Structure> tags in the request data file. For example, from logon for MTI V1R6M1.

Table 10. <Message\_Body> Record Definitions

<Message_Body> Record	Defines
CnsmrDataIdItemTg	<p>Consumer identifier. For example:</p> <pre>CnsmrDataIdItemTg ARF1010</pre>
CnsmrDataEncryptTg	<p>Password encryption flag. For example:</p> <pre>CnsmrDataEncryptTg 0</pre>
PswdIdTg	<p>Password. For example:</p> <pre>PswdIdTg 1010</pre>
ScrtyIdTg	<p>Security identifier. For example:</p> <pre>ScrtyIdTg 1010</pre>
PswdEncryptTpCITg	<p>Password encryption flag. For example:</p> <pre>PswdEncryptTpCITg 0</pre>
ScrtyEncryptTpCITg	<p>Security Encryption Flag. For example:</p> <pre>ScrtyEncryptTpCITg 0</pre>

**Comments:** In the request data file, precede comments with #. Place comments after each tag-values pair. Do not place comments within structure level tags unless they follow a tag-value pair.

A conversion tool creates the comments for tags and structures in GMTI supplied request data files. The record format of these comments is:

```
<Tag Name> <Tag Data> # <Tag Description> <Data Type> <Length>
```

Examples:

```
PswdIdTg 1010 # Password binary 256
ScrtyIdTg 1010 # Security ID binary 256
```

**Note:** **goldmine** does not encrypt data.

**Repeat Items:** Structures that can have multiple occurrences are defined in the data files with the comment value ,...(Repeat Item)-. Each occurrence is referred to as a single structure instance. By default the data files are generated with only a single occurrence of Repeat Item structures. Data files that contain Repeat Items structures can be modified to include multiple structure instances of a Repeat Item structure. To include multiple structure instances of a Repeat Item structure the structure instance(s) must be enclosed by a **<Repeat\_Structure>** and **</Repeat\_Structure>** statements with the number of structure instances specified on the **<Repeat\_Structure>** line.

Using the ADDBSINV (Add Business Invoice Transaction) data file as an example the structure tag InvLineDtITg is defined as a Repeat Item, as shown;

```
<Structure> InvLineDtITg          #...(Repeat Item)
  InvLineNbrIdTg 1                # Invoice Line Number ( char [12] )
  <Structure> InvLineAmtTg        #...
    <Structure> AmtTg              # Amount
      Dcm1DataValTg 6835          # Decimal Data (char [15])
      Dcm1PrecValTg 2             # Decimal Precision (long)
      DrCrIndTg 1                 # Debit/Credit Indicator (long)
  </Structure>
  IsoCurrCodeTg USD               # ISO Currency Code (char[3])
</Structure>
InvLineMemoNmTg memo txt         # Line Memo Text ( char[80])
</Structure>
```

To create multiple structure instances of the Repeat Item structure edit the data file as shown;

```
<Repeat_Structure> 2              # repeat count of 1 or greater must be specified
  <Structure> InvLineDtITg        #...(Repeat Item) Structure Instance 1
    InvLineNbrIdTg 1              # Invoice Line Number ( char [12] )
    <Structure> InvLineAmtTg      #...
      <Structure> AmtTg            # Amount
        Dcm1DataValTg 6835        # Decimal Data (char [15])
        Dcm1PrecValTg 2           # Decimal Precision (long)
        DrCrIndTg 1               # Debit/Credit Indicator (long)
  </Structure>
  IsoCurrCodeCITg USD            # ISO Currency Code (char[3])
</Structure>
InvLineMemoNmTg memo txt        # Line Memo Text ( char[80])
</Structure>
  <Structure> InvLineDtITg        #...(Repeat Item) Structure Instance 2
    InvLineNbrIdTg 2              # Invoice Line Number ( char [12] )
    <Structure> InvLineAmtTg      #...
      <Structure> AmtTg            # Amount
        Dcm1DataValTg 2503        # Decimal Data (char [15])
        Dcm1PrecValTg 2           # Decimal Precision (long)
        DrCrIndTg 1               # Debit/Credit Indicator (long)
  </Structure>
  IsoCurrCodeCITg USD            # ISO Currency Code (char[3])
</Structure>
InvLineMemoNmTg memo txt        # Line Memo Text ( char[80])
</Structure>
</Repeat_Structure>
```

The Repeat Count specified on the **<Repeat\_Structure>** line must equal the number of Structure Instances you add; otherwise an error will occur informing you that there are too many or too few structure instances defined. The structure instances must be defined inside the **<Repeat\_Structure>** and **</Repeat\_Structure>** statements. Process the modified data file using **goldmine** as specified.

**Sample Request Data File:** The following is an example of a **goldmine** request data file for building a LOGOFF request:

Figure 7. Logoff.dat

When goldmine runs, it uses header file *ifscstrs.h*. The ifscstrs.h file must reside in the same directory as goldmine, which is assumed to be the current directory. To run *goldmine* type:

*Filename.dat* is the name of the request data file. The request data file must be in the current directory or the path to the file must be provided. A list of available request data files is provided in ,Request Data Files-. The actual files are provided with the system in the data subdirectory. To use the request data files either reference the data directory path or copy the files to the current directory.

1. Gold bit stream file ending with .gold,
2. Gold message file ending with .out



```

OriginatorType: [2001]
Message Mode: [5001]
Code Page: [850]
Request Type: [LOGOFF ] Interval Number: [0]
TimeStamp: [1999-04-20-08.57.44.000000]
GMTOffsetMinutes: [300]
Server Context: []
remoteAddress = []
Session Info:
UserId: [ARF1001]
ServerId: []
FI Id: [IBANKA]
Session Num: [0]
IFS Trace ID:
Server Instance Number: [0]
Service Request Reference: [0]
Return Code Info:
Comp Code: [0]
Reason Code: [0]
senseData = []

Extra Phase 2 Header Fields:
2          MsgHdrTg      (Id: [2019])
2.1        TrnUIdTg      (Id: [20069], Len: [1], Val:[])

Message Body:
1          IfsCnsmrDataIdTg ((IFS_CNSMR_DATA_ID) Id: [839])
1.1        cnsmr_data_id_item: [ 41] Len: [7]
1.2        cnsmr_data_encrypt_tp_c1: [H]

```

Figure 8. Logoff.out

When you run **goldmine**, you may receive one or more error messages or return codes. See ,Messages and Codes- for information on these messages and codes.

The first request to be made by **goldmine** and subsequently **goldbank** must be logon. Any of the remaining request data files may be used after logon is complete. When the session has been complete, **goldmine** and **goldbank** must run the logoff request.

## goldbank

**Goldbank** reads the goldbank.config file, which should be modified before running **goldbank**. The name of the gold bit stream file (i.e., logoff.gold that is built by **goldmine**) must be passed into **goldbank** at runtime. This section describes both files in detail.

## Modifying the goldbank Configuration File

**Goldbank** reads the configuration file, goldbank.config, each time it runs. You can edit these values in the file:

Table 11. goldbank Configuration File Values

Value	Specifies
AIF	When AIF is set to 0, goldbank does not build an AIF header and the transaction is sent to the target queue.  When AIF is set to 1, goldbank builds an AIF header in EBCDIC and the transaction is sent to a queue named <i>FDK.servicename.FIName</i> where <i>servicename</i> is read from the RequestType field of the <Header> section of the request data file and FIName is read from goldbank.config.
DefaultTimeout	The time, in seconds, that goldbank waits for a response message to arrive in the specified reply queue. Set this to a value greater than 0 which means no response is expected.
LocalQManager	The name of the local queue manager to which goldbank will connect.
RemoteQManager	The name of the queue manager to which goldbank sends the message.



```

FI Id: [IBANKA]
Session Num: [0]
IFS Trace ID:
Server Instance Number: [0]
Service Request Reference: [0]
Return Code Info:
Comp Code: [0]
Reason Code: [0]
senseData = []

Extra Phase 2 Header Fields:
2          MsgHdrTg          (Id: [2019])
2.1        TrnUIdTg          (Id: [20069], Len: [1], Val: [])

Message Body:
1          IfsRespCodeTg      ((IFS_RESP_CODE) Id: [44])
1.1        resp_code_cl: [000]
1.2        fi_proc_ind: []

```

Figure 10. logoff.fgresp

## Request Data Files

The following table lists the available server test cases for MTI level V1R6.

Table 12. Server Test Cases for MTI Level V1R6

Request Type	Request Message	Meaning	Request Data File
ABLRVLD	ADD_BILLER_ACCT_VLD_TXN	Add biller account validation rule.	ablrvid.dat
ACCTDTL	ACCT_DTL_TXN	Account detail inquiry.	acctdtl.dat
ACCTHIST	ACCOUNT_HISTORY_TXN	Account history inquiry.	accthist.dat
ACCTINFO	ACCT_INFO_TXN	Ask about account.	acctinfo.dat
ACPTDSCL	DISCLOSURE_ACPT_TXN	Agreement online registration disclosure.	acptdscl.dat
ADDACCT	ADD_ACCT_TXN	Add account to service profile.	addacct.dat
ADDBILR	ADD_BILLER_TXN	Add biller.	addbilr.dat
ADDBSINV	ADD_BUSINESS_INVOICE_TXN	Add business invoice transaction.	addbsinv.dat
ADDCRMIT	ADD_CUST_REMIT_DATA_TXN	Add customer remittance data transaction.	addcrmit.dat
ADDMPBEN	ADD_MP_BENF_TXN	Add multiple payment beneficiary.	addmpben.dat
ADDPAYEE	ADD_PAYEE_TXN	Add payee.	addpayee.dat
ADDPBEN	ADD_PP_BENF_TXN	Add priority payment beneficiary.	addppben.dat
ADDRCAT	ADD_RGSTR_CAT_TXN	Add register category.	addrcat.dat
ADDREGEN	ADD_RGSTR_ENTR_TXN	Add register entry.	addregen.dat
ADDSECN	ADD_SEC_CNSMR	Add secondary consumer.	addsecn.dat
ADDSO	ADD_STDG_ORDR_TXN	Add standing order.	addso.dat
ADDSRVC	ADD_SRVC_TXN	Add service transaction.	addsrcv.dat
AFIXPYMT	ADD_FIX_PYMT_TXN	Add fixed amount recurring payment transaction.	afixpymt.dat
AFIXTFR	ADD_FIX_TFR_TXN	Add fixed amount recurring transfer.	afixtfr.dat
APPLOGON	APPL_LOGON_TXN	Application logon.	applogon.dat
AUTHACCT	AUTH_ACCT_TXN	Authorize online registration accounts.	authacct.dat
AVARPYMT	ADD_VAR_PYMT_TXN	Add variable amount single payment.	avarpymt.dat
AVARTFR	ADD_VAR_TFR_TXN	Add variable amount single transfer.	avartfr.dat
BALINQ	BAL_INQ_TXN	Balance inquiry.	balinq.dat

<b>Request Type</b>	<b>Request Message</b>	<b>Meaning</b>	<b>Request Data File</b>
BBPAUDIN	BLR_BP_AUDT_INQR_TXN	Biller bill pay audit inquiry.	bbpaudin.dat
BILLPROB	BP_PROBLEM_TXN	Report bill pay problem.	billprob.dat
BLRDTL	BILLER_DTL_TXN	Biller detail.	blrddl.dat
BLRINQ	BILLER_INQ_TXN	Biller inquiry.	blrinq.dat
BLRVLDI	BILLER_ACCT_VLD_INQR_TXN	Biller account validation inquiry.	blrvldi.dat
BPMTADD	BILL_PYMT_ADD_TXN	Add bill payment.	bpmtadd.dat
BPMTDEL	BILL_PYMT_DEL_TXN	Delete bill payment.	bpmtdel.dat
BPMTINQ	BILL_PYMT_INQR_TXN	Bill payment inquiry.	bpmtadd.dat
BPVERIF	BP_VERIF_TXN	Request bill pay verification.	bpverif.dat
BSINVINQ	BUSINESS_INVOICE_INQR_TXN	Business invoice inquiry transaction.	bsinvinq.dat
CBLRVLD	CHNG_BILLER_ACCT_VLD_TXN	Change biller account validation rule.	cblrvld.dat
CBPAUDIN	CNSMR_BP_AUDT_INQR_TXN	Consumer bill pay audit inquiry.	cbpaudin.dat
CCDSPUT	CC_DSPUT_TXN	Report credit card dispute.	ccdsput.dat
CCSTART	IFS_CCSTARTUP_TXN	CC startup.	ccstart.dat
CCUMSG	GET_UNDLVD_MSG_TXN	Get undelivered message.	ccumsg.dat
CCUMSGLS	GET_UNDLVD_MSG_LIST_TXN	Get undelivered list.	ccumscls.dat
CFIXPYMT	CHNG_FIX_PYMT_TXN	Change fixed amount recurring payment.	cfixpymt.dat
CFIXTFR	CHNG_FIX_TFR_TXN	Change fixed amount recurring transfer.	cfixtfr.dat
CHGBSINV	CHNG_BUSINESS_INVOICE_TXN	Change business invoice transaction.	chgbssnv.dat
CHGCCLMT	CHG_CC_LMT_TXN	Request change CC limit	chgcclmt.dat
CHGCRMIT	CHNG_CUST_REMIT_DATA_TXN	Change customer remittance data transaction.	chgcrmit.dat
CHGDDSO	CHG_DIR_DR_STDG_ORDR_TXN	Change direct debit / standing order.	chgddso.dat
CHGMPBEN	CHG_MP_BENF_TXN	Change multiple payment beneficiary.	chgmpben.dat
CHGPAYEE	CHNG_PAYEE_TXN	Change payee.	chgpayee.dat
CHGPPBEN	CHG_PP_BENF_TXN	Change priority payment beneficiary.	chgppben.dat
CHGRCAT	CHG_RGSTR_CAT_TXN	Change register category.	chgrcat.dat
CHGREGEN	CHG_RGSTR_ENTR_TXN	Change registry entry.	chgregen.dat
CHGSECN	CHG_SEC_CNSMR	Change secondary consumer.	chgsecln.dat
CHKBOOK	CHK_BOOK_TXN	Check book reorder.	chkbook.dat
CHKCOPY	CHECK_COPY_TXN	Check copy order.	chkcopy.dat
CHNGBILR	CHNG_BILLER_TXN	Change biller.	chngbilr.dat
CHPYMTHS	CHNG_PYMT_HIST_TXN	Change payment history.	chpymths.dat
CLSSECN	CLS_SEC_CNSMR	Close secondary consumer.	clssecln.dat
CNCSTPMT	CNCL_STOP_CHK_PYMT_TXN	Cancel a Cancel-Payment request.	cncstpmt.dat
CNSMCFRM	CNSMR_MSG_CFRM_TXN	Consumer message confirmation.	cnsmcfrn.dat
CNVRPYTP	CNVR_PAYEE_TP_TXN	Convert payee type.	cnvrpytp.dat
CODERFSH	CODE_REFRESH_TXN	Code refresh.	coderfsh.dat
CONSSEVT	CNSMR_SSN_START_TXN	IFS consumer session startup.	conssevt.dat
CPYMTINS	CHNG_PYMT_INSTN_TXN	Change payment instance.	cpymtins.dat
CRAPPL	CR_APPL_TXN	Request loan/line of credit.	crappl.dat
CRMITINQ	CUST_REMIT_DATA_INQR_TXN	Retrieve customer remittance data.	crmitinq.dat
CSGCNSMR	CSR_CUST_PRFL_TXN	CSR - Customer profile request.	csgcnsmr.dat
CSGERRLG	GET_ERROR_LOG_TXN	CSR - Error log request.	csgerrlg.dat

Request Type	Request Message	Meaning	Request Data File
CSGEVGLD	GET_EVENT_GOLD_TXN	CSR - Event Gold request.	csgevgld.dat
CSGEVLOG	GET_EVENT_LOG_TXN	CSR - Event log request.	csgevlog.dat
CSGFISTF	GET_FISTAFF_PRFL_TXN	CSR - Staff profile request.	csgfistf.dat
CSGSCRGL	GET_SCRTY_GOLD_TXN	CSR - Security Gold request.	csgscrgl.dat
CSGSCRLG	GET_SCRTY_LOG_TXN	CSR - Security log request.	csgscrlg.dat
CSGTXNGD	GET_GOLD_LOG_TXN	CSR - Transaction Gold request.	csgtxngd.dat
CSGTXNLG	GET_TXN_LOG_TXN	CSR - Transaction log request.	csgtxnlg.dat
CSGTXNLS	GET_TXN_LIST_TXN	CSR - Transaction list request.	csgtxnls.dat
CSGUSSSN	GET_USER_SESSION_TXN	CSR - User session query.	csgusssn.dat
CSLOGOFF	LOGOFF_CTW_TXN	CSR - Logoff CTW.	cslogoff.dat
CSLOGON	LOGON_CTW_TXN	CSR - Logon CTW.	cslogon.dat
CSSCMSMR	CSR_SHRT_CNSMR_PRFL_TXN	CSR - Short consumer profile transaction.	csscnsmr.dat
CSTEMAIL	CUST_FI_EMAIL	Customer reply or communication to FI.	cstemail.dat
CSUCNSMR	CSR_UPDT_CUST_PRFL_TXN	CSR - Update customer profile.	csucnsmr.dat
CSUFISTF	UPDT_FISTAFF_PRFL_TXN	CSR - Update staff profile.	csufistf.dat
CVARPYMT	CHNG_VAR_PYMT_TXN	Change variable amount single payment.	cvarpymt.dat
CVARTFR	CHNG_VAR_TFR_TXN	Change variable amount single transfer.	cvartfr.dat
DBLRVLD	DEL_BILLER_ACCT_VLD_TXN	Delete biller account validation rule.	dblrvld.dat
DDSODTL	DIR_DR_STDG_ORDR_DTL_TXN	Direct debit / standing order detail.	ddsodtl.dat
DDSODLIST	DIR_DR_STDG_ORDR_LIST_TXN	Direct debit / standing order list.	ddsolist.dat
DELBILR	DEL_BILLER_TXN	Delete biller.	delbblr.dat
DELBSINV	DEL_BUSINESS_INVOICE_TXN	Delete business invoice transaction.	delbsinv.dat
DELCRMIT	DEL_CUST_REMIT_DATA_TXN	Delete customer remittance data transaction.	delcrmit.dat
DELDDSO	DEL_DIR_DR_STDG_ORDR_TXN	Delete direct debit / standing order.	delddso.dat
DELMPPBEN	DEL_MP_BENF_TXN	Delete multiple payment beneficiary.	delmpben.dat
DELMSG	DEL_FI_MSG	Delete a FI initiated message.	delmsg.dat
DELPAYEE	DEL_PAYEE_TXN	Delete payee.	delpayee.dat
DELPPBEN	DEL_PP_BENF_TXN	Delete priority payment beneficiary.	delppben.dat
DELRCAT	DEL_RGSTR_CAT_TXN	Delete register category.	delrcat.dat
DELREGEN	DEL_RGSTR_ENTR_TXN	Delete registry entry.	delrcat.dat
DEPBOOK	DEP_BOOK_TXN	Request deposit book.	depbook.dat
DFIXPYMT	DEL_FIX_PYMT_TXN	Delete fixed amount recurring payment.	dfixpymt.dat
DFIXTFR	DEL_FIX_TFR_TXN	Delete fixed amount recurring transfer.	dfixtfr.dat
DISCLINQ	DISCLOSURE_INQ_TXN	Disclosure inquiry transaction.	disclinq.dat
DPYMTINS	DEL_PYMT_INSTN_TXN	Delete payment instance.	dpymtins.dat
DVARPYMT	DEL_VAR_PYMT_TXN	Delete variable amount single payment.	dvarpymt.dat
DVARTFR	DEL_VAR_TFR_TXN	Delete variable amount single transfer.	dvartfr.dat
EBACTBLR	EBILL_ACT_DLVRY_TXN	Activate electronic biller delivery transaction.	ebactblr.dat
EBBILINQ	EBILL_BILL_INQ_TXN	Electronic bill inquiry transaction.	ebbiling.dat
EBBLRINQ	EBILL_BILLER_INQ_TXN	Electronic biller inquiry transaction.	ebblrinq.dat
EBCHGST	EBILL_CHG_STATE_TXN	Change electronic bill state transaction.	ebchgst.dat

<b>Request Type</b>	<b>Request Message</b>	<b>Meaning</b>	<b>Request Data File</b>
EBDEABLR	EBILL_DEACT_DLVR_TSN	Deactivate electronic biller delivery transaction.	ebdeablr.dat
EBDLINQ	EBILL_BILLER_DLVR_INQ_TSN	Electronic biller delivery inquiry transaction.	ebdlinq.dat
ENDCNMR	CSR_CNSMR_END_TSN	CSR consumer end.	endcnsmr.dat
ERRORNOT	IFS_ERRORNOTIFY_TSN	Error notify transaction.	errornot.dat
EXCHRATE	EXCH_RATE_INQR_TSN	Exchange rate inquiry transaction.	exchrte.dat
FBPAUDIN	FI_BP_AUDT_INQR_TSN	FI bill pay audit inquiry.	fbpaudin.dat
FICUST	FI_CUST_EMAIL	FI to customer communication.	ficust.dat
FIPROF	FI_PROFILE_TSN	FI profile.	fiprofat
FORCEUSR	CSR_FORCE_USR_TSN	IFS CSR force user.	forceusr.dat
GACIADJN	GET_ACI_ADJ_NOTE_TSN	Get ACI adjustment note.	gaciadjn.dat
GACPTMPL	GET_ACP_TMPL_TSN	Get ACP template.	gacptmpl.dat
GBILELMT	GET_BILL_ELMT_TSN	Get billing elements.	gbilelmt.dat
GBILENTR	GET_BILL_ENTR_TSN	Get billing entries.	gbilenr.dat
GBPSRVC	GET_BP_SRVC_DTL_TSN	Get bill pay service details.	gbpsrv.dat
GCLHSTYP	GET_CLHS_TP_TSN	Query CLEARINGHOUSE_TYPE table.	gclhstyp.dat
GDESTFI	GET_DEST_FI_TSN	Get information from DESTINATION_FI table.	gdestfi.dat
GDESTCLS	GET_DEST_CLHS_TSN	Get clearing house ids for a destination FI ID.	gdestcls.dat
GDSTCLSP	GET_DEST_CLHS_PRFL_TSN	Queries DESTINATION_FI and DEST_FI_CLHS_TYPE tables (for CWS).	gdstclsp.dat
GETACP	GET_ACP_TSN	ACP request.	getacp.dat
GETEURRT	GET_EUR_EXCH_RT_TSN	View local currency/Euro exchange rate.	geteurrt.dat
GETIFSPR	GET_IFS_PROFILE_TSN	Get IFS profiles inquiry.	getifspr.dat
GETOBLG	GET_OOB_LOG_REF_TSN	OOB log reference request.	getooblg.dat
GETSPACI	GET_SPACI_TSN	Service provider ACI.	getspaci.dat
GETSPACP	GET_SPACP_TSN	Service provider ACP.	getspacp.dat
GETSRACI	GET_SRACI_TSN	Service request ACI.	getsraci.dat
GETTCALG	GET_TCA_LOG_TSN	Get TCA log records.	gettcalg.dat
GETTCAT	GET_TCA_TOT_TSN	Get TCA totals.	gettcacat.dat
GEXCHCAL	GET_EXCH_CAL_TSN	Get operating hours of supported markets.	gexchcal.dat
GFIHOL	GET_FI_HOLIDAY_TSN	Get FI holiday list.	gfihol.dat
GINTEGPR	GET_INTEG_PRFL_TSN	Get Integrion profile.	gintegpr.dat
GINVDTL	GET_INV_SVC_DTL_TSN	Request investment service profile.	ginvdtl.dat
GNPAYADV	GET_NO_PAY_ADV_TSN	Get no-pay advice.	gnpayadv.dat
GOBSRACI	GET_OOB_SRACI_TSN	OOB service request ACI.	gobsraci.dat
GORDCOMP	GET_ORD_COMPL_ADV_TSN	Retrieve order completion advice messages.	gordcomp.dat
GPCUSTFI	GP_CUST_FI_TSN	General purpose customer to FI communication.	gpcustfi.dat
GPINQ	GP_INQ_TSN	General purpose inquiry.	gpinq.dat
GPMON	GP_MON_TSN	General purpose monetary.	gpmon.dat
GPNONMON	GP_NON_MON_TSN	General purpose non-monetary.	gpnnonmon.dat

Request Type	Request Message	Meaning	Request Data File
GSCRTYLV	GET_SCRTY_LVL_TXN	Get security levels.	gscrtylv.dat
GSECDISC	GET_SEC_DESC_TXN	Retrieve securities information.	gsecdesc.dat
GTCASRLS	GET_TCASR_LST_TXN	Get TCASR list.	gtcasrls.dat
GTFRDTAV	GET_TFR_DT_ADJ_ADVC_TXN	Get transfer date adjustment advice.	gtfrdtav.dat
GTFRSRVC	GET_TFR_SRVC_DTL_TXN	Get transfer service details.	gtfrsrcv.dat
IACCTDTL	INV_ACCTDTL_TXN	Request account information details.	iacctdtl.dat
IACCTHST	INV_ACCTHST_TXN	Request investment account history.	iaccthst.dat
IFSPING	IFS_PING_TXN	IFS ping.	ifsping.dat
INQFILE	FILE_DOWNLOAD_INQR_TXN	File download inquiry.	inqfile.dat
INQMPBEN	INQR_MP_BENF_TXN	Multiple payment beneficiary inquiry.	inqmpben.dat
INQPPBEN	INQR_PP_BENF_TXN	Priority payment beneficiary inquiry.	inqppben.dat
INQSRVC	INQ_SRV_TXN	Service inquiry transaction.	inqsrcv.dat
INTRATE	INT_RATE_TXN	Interest rates.	intrate.dat
INVACCT	INV_ACCT_INFO_TXN	Associates settlement/charge account with an investment account.	invacct.dat
INVORDS	INV_ORD_STOCK_TXN	Transaction to buy/sell stock.	invords.dat
IORDCANC	INV_ORD_CANC_TXN	Requests cancellation of an existing order.	iordcanc.dat
IORDCHG	INV_ORD_CAN_CHG_TXN	Request order parameter changes (quantity, etc).	iordchg.dat
IORDINQ	INV_ORD_STAT_INQ_TXN	Request order status.	iordinq.dat
LISTMSG	LIST_FI_MSG	List FI initiated messages.	listmsg.dat
LOADADV	LOAN_ADV_TXN	Requests funds be credited to deposit account and line of credit incremented.	loanadv.dat
LOANRPYM	LOAN_REPYMT_TXN	Allows transfer of funds from deposit account to line of credit.	loanrpyr.dat
LOGOFF	LOGOFF_TXN	Logoff.	logoff.dat
LOGON	LOGON_TXN	Logon.	logon.dat
LOSTCARD	LOST_CARD_TXN	Report lost card.	lostcard.dat
MASSLGOF	MASS_LOGOFF_TXN	Mass logoff.	masslgof.dat
MPMTADD	MULTI_ACCT_PYMT_TXN	Multiple account payment.	mpmtadd.dat
MPMTDEF	MULTI_ACCT_PYMT_DEFN_TXN	Multiple account payment definition.	mpmtdef.dat
MPMTDEL	DEL_MULTI_ACC_PYM_DEF_TXN	Delete multiple account payment definition.	mpmtdel.dat
MPMTINQ	PNDG_MULTI_AC_PYM_INQ_TXN	Pending multiple account payment inquiry.	mpmtinq.dat
NEWCERT	NEW_DIGITAL_CRTF_TXN	New Digital Certificate.	newcert.dat
NICKNAME	ACCT_NICKNAME_TXN	Change account nickname.	nickname.dat
NOAYADV	NO_PAY_ADVC_TXN	No-pay advice.	noayadv.dat
OBOLOGON	OBO_LOGON_TXN	On behalf of (OBO) logon.	obologon.dat
ORDCOMPL	ORD_COMPL_ADV_TXN	Order status (from FI to IFS Message Server).	ordcompl.cat
PAYEEINQ	PAYEE_INQ_TXN	Payee inquiry.	payeeinq.dat
PAYINQ	PAYMENT_INQ_TXN	Payment instance inquiry.	payinq.dat
PINPSWD	ACCT_PINPSWD_TXN	PIN/password change.	pinpswd.dat
PPMTADD	PRTY_PYMT_ADD_TXN	Add priority payment.	ppmtadd.dat
PPMTDEL	PRTY_PYMT_DEL_TXN	Delete priority payment.	ppmtdel.dat

Request Type	Request Message	Meaning	Request Data File
PPMTINQ	PRTY_PYMT_INQR_TXN	Priority payment inquiry.	ppmtinq.dat
PYEEDTL	PAYEE_DTL_TXN	Payee detail.	payeedtl.dat
PYMTDEFI	PYMT_DEFN_INQR_TXN	Payment definition inquiry.	pymtdefi.dat
QRYPOSTX	QUERY_PSTD_TXN	Query posted transactions.	qrypostx.dat
QRYTCHPB	QUERY_TECH_PRBLM_TXN	Query technical problem.	qrytchpb.dat
RCATINQ	RGSTR_CAT_INQ_TXN	Register category inquiry.	rcatinq.dat
READMSG	GET_FI_MSG	Read FI initiated message.	readmsg.dat
REAUTH	IFS_REAUTH_TXN	Re-authentication.	reauth.dat
RECVFILE	RECV_FILE_TXN	Receive file transfer.	recvfile.dat
REGINQ	RGSTR_INQ_TXN	Register inquiry.	reginq.dat
REMACCT	REM_ACCT_TXN	Remove account from service profile.	remacct.dat
REMSRVC	REM_SRVC_TXN	Remove service transaction.	remsrvc.dat
RENUCERT	RENEW_DIGITAL_CRTF_TXN	Renew digital certificate.	renucert.dat
RESCERT	RESUME_DIGITAL_CRTF_TXN	Resume digital certificate.	rescert.dat
RESETUSR	CSR_RESET_USR_TXN	IFS CSR reset user.	resetusr.dat
REVCERT	REVOKE_DIGITAL_CRTF_TXN	Revoke digital certificate.	revcert.dat
RPAYDCSN	PAY_DCSN_RQST_TXN	Request pay no pay decision.	rpaydcsn.dat
RSCNSCRT	RSET_CNSMR_SCRTY_TXN	Reset consumer security.	rsncsrt.dat
RSTSSECN	RSET_SEC_CNSMR	Reset secondary consumer.	rstssecn.dat
RVBPRQST	REV_BP_RQST_TXN	Reverse bill pay request.	rvbprqst.dat
SALESSLP	SALES_CPY_TXN	Sales slip copy order.	salesslp.dat
SECNINQ	SEC_CNSMR_INQ	Secondary consumer inquiry.	secninq.dat
SECURITY	IFS_SECURITY_TXN	Security.	security.dat
SELFREGST	SELF_REGIS_TXN	Self registration.	selfrgst.dat
SETTLACT	SETTL_2_ACCT_TXN	Settles account (between back office and bank).	settlact.dat
SRCHCNSM	SRCH_CNSMR_PRFL_TXN	Search consumer profile.	srchcnsn.dat
SRVCPROF	SRVC_PROFILE_TXN	Service profile inquiry.	srvcprof.dat
SRVCSTAT	UPDT_SRVC_STAT_TXN	Service status change.	srvcstat.dat
SRVSHUTD	SERVER_STOP_TXN	Server shutdown.	srvshutd.dat
SRVSTART	SERVER_START_TXN	Server startup.	srvstart.dat
STATENOT	IFS_STATENOTIFY_TXN	State notify.	statenot.dat
STMTCOPY	STMT_COPY_TXN	Statement copy order.	stmtcopy.dat
STPCKPMT	STOP_CHK_PYMT_TXN	Stop check payment.	stpckpmt.dat
STPMTING	STOP_CHK_PYMT_INQR_TXN	Request details of all stop payment orders.	stpmting.dat
STPRCPMT	STOP_RCUR_DRFT_TXN	Allows cancel of Recurring Draft Payment.	stprcpmt.dat
STRCNSMR	CSR_CNSMR_STRT_TXN	CSR consumer start.	strcnsmr.dat
SUSCERT	SUSPEND_DIGITAL_CRTF_TXN	Suspend digital certificate.	suscert.dat
SUSPCNS	SUSP_CNSR_MSG	Suspends account with FI.	suspcns.dat
TAXDTL	ACCT_TAX_DTL_TXN	Account tax detail.	taxdtl.dat
TDEPINQR	TERM_DEP_INQR_TXN	Term deposit inquiry.	tdepinqr.dat
TDEPOPEN	TERM_DEP_OPN_TXN	Term deposit account open.	tdepopn.dat
TDEPRQST	TERM_DEP_RQST_TXN	Term deposit account request.	tdeprqst.dat
TFRDADV	TFR_DT_ADJ_ADV_TXN	Transfer date adjustment advice.	tfrdadv.dat



Request Type	Request Message	Meaning	Request Data File
TFRINQ	TFR_INQ_TXN	Transfer inquiry.	tfrinq.dat
TFRPROB	TFR_PROBLEM_TXN	Report a transfer transaction problem.	tfrprob.dat
TRANPRFL	TXN_PROFILE_TXN	Transfer profile.	tranprfl.dat
UACIADJN	UPDT_ACI_ADJ_NOTE_TXN	Update ACI adjustment note.	uaciadjn.dat
UACPTMPL	UPDT_ACP_TMPL_TXN	Update ACP template.	uacptmpl.dat
UBILELMT	UPDT_BILL_ELMT_TXN	Update billing elements.	ubilelmt.dat
UBPSRVC	UPDT_BP_SRVC_DTL_TXN	Update bill pay service details.	ubpsrvc.dat
UCLHSTYP	UPDT_CLHS_TP_TXN	CLEARINGHOUSE_TYPE table maintenance.	uclhstyp.dat
UCNSMRDL	UPDT_CNSMR_DTL_TXN	Update consumer details.	ucnsmrld.dat
UDESTCLS	UPDT_DEST_CLHS_TXN	DEST_FI_CLHS_TYPE table maintenance.	udestcls.dat
UDESTFI	UPDT_DEST_FI_TXN	Update DESTINATION_FI table.	udestfi.dat
UEXCHCAL	UPD_EXCH_CAL_TXN	Update market calendar.	uexchcal.dat
UFIHOL	UPDT_FI_HOLIDAY_TXN	Update FI holiday list.	ufihol.dat
UFIPRFL	UPDT_FI_PRFL_TXN	Update FI profile.	ufiprfl.dat
UILLENTR	UPDT_BILL_ENTR_TXN	Billing entry update.	uillentr.dat
UINTEGPR	UPDT_INTEG_PRFL_TXN	Update Integrion profile.	uintegpr.dat
UINVDTL	UPD_INV_SVC_DTL	Update Investment Service Profile (CTW).	uinvdtl.dat
UNDELIV	IFS_UNDELIVERED_TXN	Undelivered transaction.	undeliv.dat
UPDTGCDE	CSR_UPDT_GOLD_CODE_TXN	CSR update gold code.	updtgcde.dat
UPDTSRVC	UPDT_SRVC_TXN	Update service transaction.	updtsrvc.dat
UPSRREQPR	UPDT_SRVC_RQST_DTL_TXN	Update service request profile.	upsrreqpr.dat
UPSRVCPR	UPDT_SRVC_PRFL_DTL_TXN	Update service profile.	upsrvcpr.dat
USCRTYLV	UPDT_SCRTY_LVL_TXN	Update security levels.	uscrtylv.dat
USERPROF	USER_PROFILE_TXN	FI consumer profile inquiry.	userprof.dat
UTFRSRVC	UPDT_TFR_SRVC_DTL_TXN	Update transfer service details.	utfrsrvc.dat
WTFRCNCL	WIRE_TFR_CNCL_TXN	Cancel a wire transfer transaction.	wtfrcncl.dat
WTFRINIT	WIRE_TFR_INIT_TXN	Request to wire funds to an FI.	wtfrinit.dat
WTFRINQR	WIRE_TFR_INQR_TXN	Request details of standing wire transfers.	wtfrinqr.dat

## Messages and Codes

GMIT programs, **goldmine** and **goldbank**, issue error and informational messages. MTI issues error codes and sense data. The following sections detail these messages and codes.

### goldbank Messages

Table 13. goldbank Messages

goldbank Error	Explanation
Error reading <INIT> tag.	The <INIT> tag in goldbank.config is incorrect.
Error reading configuration file filename.	goldbank could not read the configuration file filename.
goldbank cannot open {bit stream formatted response response bit stream} file filename	goldbank cannot open file filename, which is one of the following files: bit stream file

<b>goldbank Error</b>	<b>Explanation</b>
bit stream} file filename.	formatted response file response bit stream file
<b>goldbank</b> found no <INIT> tag in file filename.	goldbank did not find an <INIT> tag in file filename.
<b>goldbank</b> is not compiled for AIF support.	goldbank is not compiled for AIF support.
MQ function failed with return code rc.	MQ function failed with return code rc.
The bit stream file filename does not contain a valid Gold message.	The bit stream file filename does not contain a valid Gold message. You may need to rerun <b>goldmine</b> to regenerate the .gold file.
The ConnectQM() failed with return code rc.	The ConnectQM function failed with return code rc.
<b>goldbank</b> filename Gold request is <i>n</i> bytes.	The <b>goldbank</b> file filename Gold request length is n.
<b>goldbank</b> read n bytes from bit stream file filename.	<b>goldbank</b> read n bytes from bit stream file filename.
<b>goldbank</b> received n bytes. [The AIF header is <i>n1</i> bytes. The Gold message length is <i>n2</i> bytes.]	<b>goldbank</b> received n bytes. The AIF header length is n1. The Gold message is n2.
<b>goldbank</b> successfully processed file filename.	<b>goldbank</b> successfully processed file filename.
<b>goldbank</b> file.gold target-queue target-qmgr	To run <b>goldbank</b> , enter: <b>goldbank</b> file.gold target-queue target-qmgr

## goldmine Messages

Table 14. goldmine Messages

<b>goldmine Error</b>	<b>Explanation</b>
<b>goldmine</b> cannot find a <Header> tag in file filename.	filename does not contain a <Header> section. Define a <Header> section.
<b>goldmine</b> cannot find a <Message_Body> tag in file filename.	filename does not contain a <Message_Body> section. Define a <Message_Body> section.
<b>goldmine</b> cannot find file filename.	goldmine cannot find file filename. Copy filename to the current directory.
<b>goldmine</b> cannot find IFS tag tag in file filename.	Tag tag not found in header file filename. Make sure that the header file is in the current directory and that the tag is spelled correctly.
<b>goldmine</b> cannot find the IFS define in file filename.	define not found in header file filename. Make sure that the header file is in the current directory and that the define is spelled correctly.
The tag tag does not match expected field field.	Tag tag is out of order or is not valid.
The tag tag is incorrect for this Gold MTI version.	This MTI level does not support tag tag.
The tag tag is incorrect within the <Header> section.	Tag tag is incorrect within the <Header> section.
Not expecting additional repeat structure instances, repeat_count = <value>	Number of structure instances added to Repeat_Structure exceeds Repeat_Structure count.
Repeat count value incorrect, value <value>	Count specified on <Repeat_Structure> line must be 1 or greater.
expecting additional repeat structure instances, repeat_count = <value>	Either reduce Repeat_Structure count to equal number of structure instances defined or add Structure instances to equal Repeat_Structure count.
The tag tag is incorrect within the <Message_Body> section.	Tag tag is incorrect within the <Message_Body> section.

goldmine Error	Explanation
The request data file name filename is incorrect.	Verify filename.

## MTI Codes

See MTI documentation for more information about these return codes and sense data.

The variables in these messages are:

*program*: Program issuing the message.  
*sensedata*: Sense data.  
*rc*: Return code.

*program* error: Sense data is *sensedata*.

*program* error: MTIAddField() failed with return code *rc*.

*program* error: MTIAddFieldInstance() failed with return code *rc*.

*program* error: MTIAddHeader failed() with return code *rc*.

*program* error: MTIAddStructure failed() with return code *rc*.

*program* error: MTIAddStructureInstance() failed with return code *rc*.

*program* error: MTIDumpMessage failed() with return code *rc*.

*program* error: MTIGetHeader failed() with return code *rc*.

*program* error: MTIGetMessageFromBitstream() failed with return code *rc*.

*program* error: MTIGetPreDefinedStructureDefinition() failed with return code *rc*.

*program* error: MTIInitialize() failed with return code *rc*

*program* error: MTIInitializeMessage() failed with return code *rc*

*program* error: MTIPutMessageToBitstream() failed with return code *rc*

*program* error: MTITerminate() failed with return code *rc*.

*program* error: Sense data is *sensedata*.

---

## Appendix D. FISim (Financial Institution Simulator)

---

### Introduction

FISim is a Financial Institution (FI) simulator and a Core Controller (CC) simulator for use with the Customizable Web Server (CWS). When FISim is used without a CC, it simulates both the FI and the CC. When FISim is used with the CC, it simulates the FI only.

FISim runs on an RS6000 with the AIX operating system, DB2, and MQ Series. A version of FISim that does not use MQ Series is available in the CWS Toolkit. This document only describes the MQ Series version of FISim for AIX.

When FISim is used without a Core Controller, FISim may reside be on a different AIX machine or the same machine as the CWS web server. Since the Core Controller runs on an MVS machine and FISim runs on an AIX machine, FISim used with a Core Controller will always be on a different machine from the Core Controller though it may still be on the same machine as the CWS. A previous version of FISim ran on an MVS machine with the Core Controller.

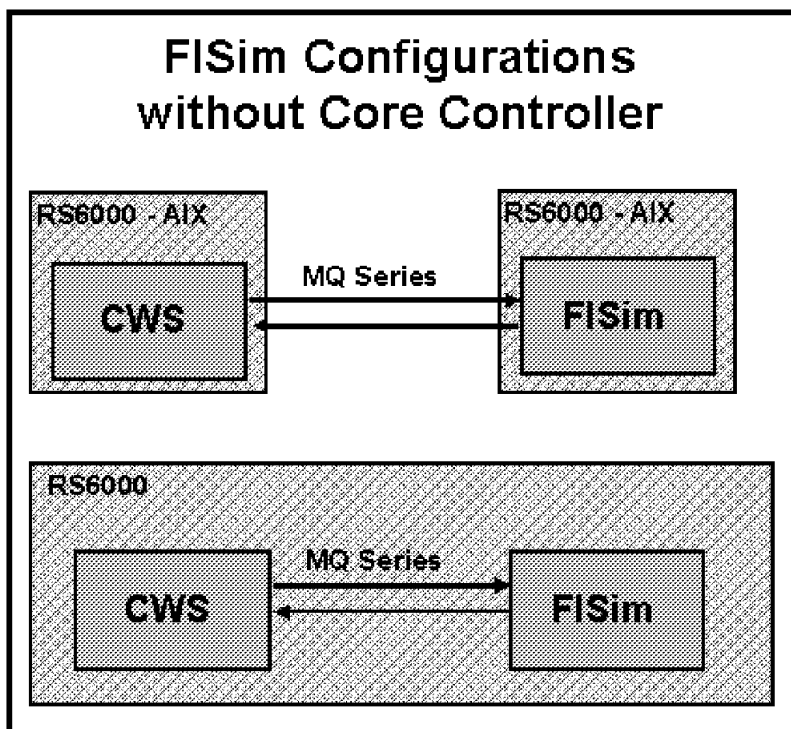


Figure 11. FISim Configurations without a Core Controller

Figure 13 shows configurations without a Core Controller and Figure 14 shows configurations with a Core Controller.

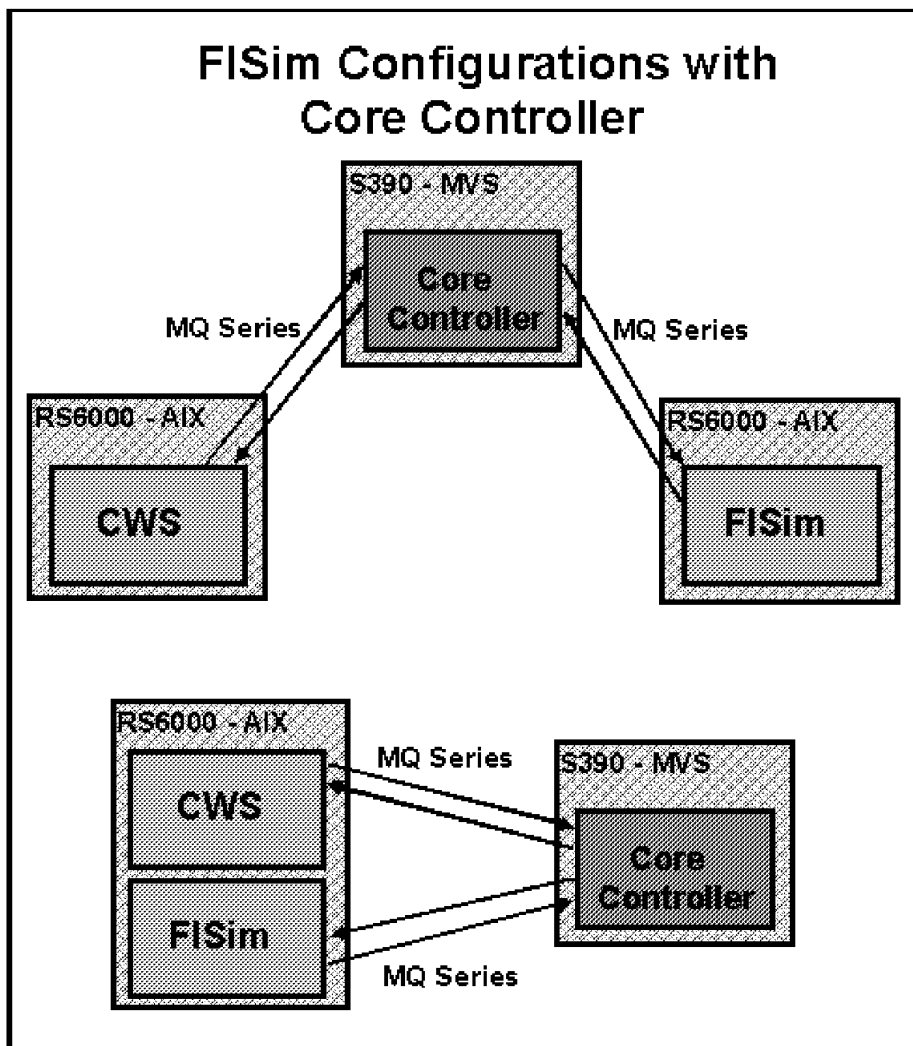


Figure 12. FISim Configurations with a Core Controller

## The Test Data Flow

FISim provides for testing the CWS web pages, the CWS web server and the Core Controller without a connection to a real FI. FISim receives Gold request messages through an MQ Series transmit queue and provides Gold response messages to an MQ Series reply queue.

Without a Core Controller, FISim receives requests from the CWS web server and returns the responses to the CWS web server. With a Core Controller, the CWS web server requests are first routed to the Core Controller. The Core Controller handles some requests itself and returns the response for these requests to the CWS web server. For all other transactions, the Core Controller sends the requests to FISim which responds to the Core Controller. The Core Controller may augment the FISim response before returning the response to the CWS web server. For the Logon and Self-Registration transactions, FISim responses to the Core Controller contain less data than the FISim responses to the CWS web server without a Core Controller.

When FISim receives a Gold request, it first checks to see if there is a predefined response file available for the request. If so, it attempts to deliver the response file as a response, as detailed in ,Predefined Responses-. If not, FISim examines the request to determine if the transaction is supported by the DB2 based portion of the simulation.

When the request is supported by the enhanced simulation capability, FISim examines the request to determine whether the request is valid based on the contents of the database. If the request is valid, the FISim returns a Gold transaction response based on data in the database. If the request is not valid based on the database contents, the FISim returns a Gold transaction to indicate the error. For example, the logon

request will provide the account summary data if the user ID, password, and PIN match the database or it will return a logon failure message.

For those transactions not supported by the DB2 based simulation, FISim returns a canned response that depends only on the transaction type (for example, logon and account history) regardless of the request message parameters. For these transactions, FISim neither examines nor verifies the correctness of the request message; it simply returns a static response for each request.

Figure 15 shows the data flow between the web page and FISim with a Core Controller.

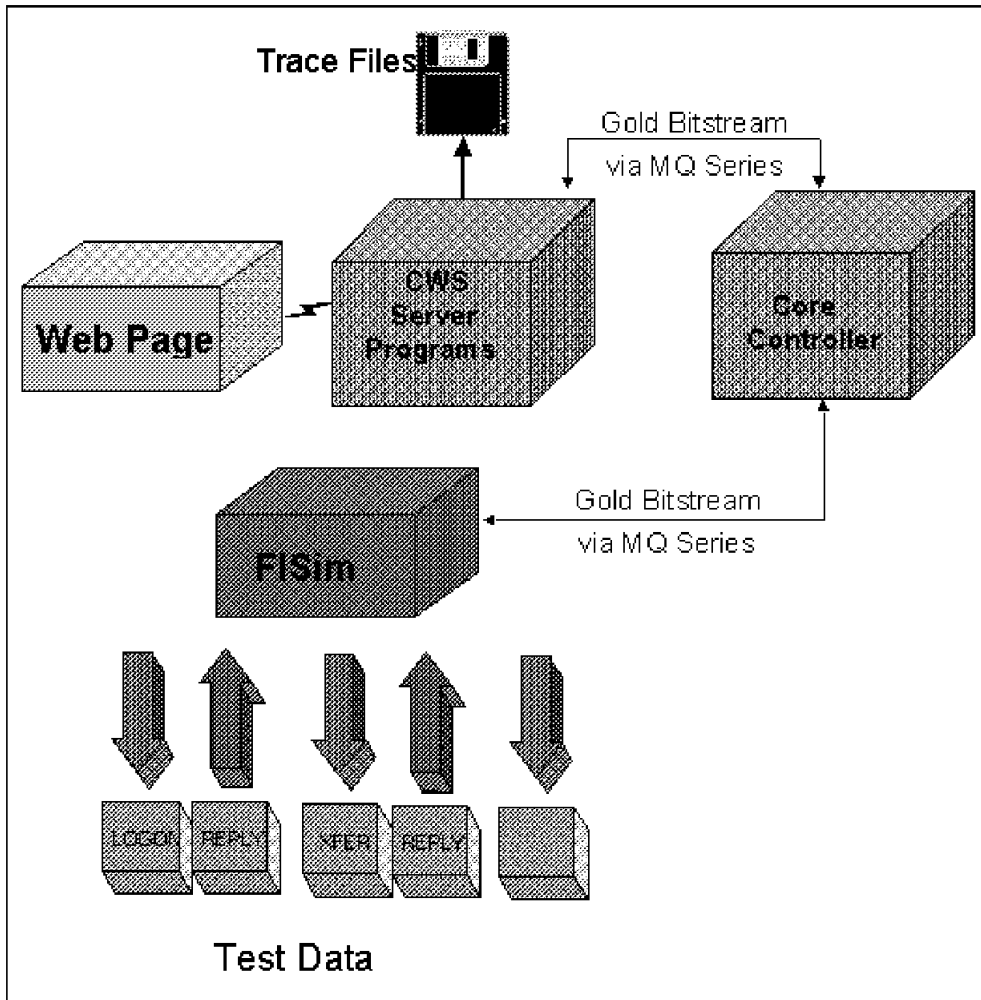


Figure 13. FISim Data Flow with a Core Controller

When a CWSAPI request is made from a page, the request is routed to FISim, which returns data based on the database or from an internal table for the transaction.

## Installing FISim

### Prerequisites

#### Hardware

The FISim must be installed on an IBM e-server pSeries or RS/6000 system. The following hardware environment is suggested as a minimum.

- Entry level IBM e-server pSeries or RS/6000 system

- 64MB+ RAM
- 4GB+ disk space
- Graphics attachment for local user access
- LAN attachment for remote user access

## Software

The list of prerequisite software needed by FISim is as follows:

- AIX Operating System, Version 4.3.3 (Includes TCP/IP)
- MQ Series, Version 5 - Configured using the System Management Tool (SMIT).
- The C++ Run Time Environment version 4.0.2 (or newer).  
Use the , lspp -L xIC.\* , command to determine the currently installed version.
- IBM DB2 UDB (Version 7.1 or newer) for AIX

Optional software to view DB2 data on a remote system

- IBM DB2 (Version 7.1 or newer) for Windows
- Lotus Approach or Microsoft Excel for Windows

## Installation

FISim can be installed by copying the subdirectory ,fis- and its subdirectories from the export subdirectory under the desired build.

```
cp -r /desiredbuild/export/fis destination
```

If FISim has been provided in a tar file, execute the tar command to extract the files in the desired directory:

```
tar xvf fisim_tarfilename.tar
```

Once the FISim software has been installed, MQ Series and DB2 must be configured to run with FISim. Please refer to ,MQSeries and FISim- to set up MQ Series for your configuration. Please refer to ,DB2 Configuration- to set up the DB2 database for use with FISim.

## The Package Contents

FISim is composed of the following elements:

- The FISim program
- Test data files for use with DB2
- Scripts for updating database with test data and starting/stopping FISim

FISim is organized using the following directory structure.

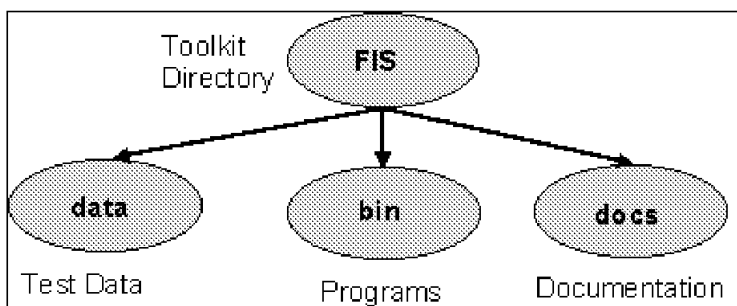


Figure 14. FISim Directory Structure

The ,fis- directory contains the ,data- directory for the DB2 based simulation data and the ,bin- directory for FISim executables.

---

## MQSeries and FISim

MQSeries allows AIX applications to use message queuing to participate in message-driven processing. MQWeb utilizes several MQSeries queues in its functionality. This appendix assumes the user has already installed MQSeries and the SMIT GUI panels. To install MQSeries, see [Beta TBD].

Three different MQ Series configurations will be described. One configuration has the CWS web server on the same machine as FISim without a Core Controller. This may be thought of as an all local configuration. The second MQ Series configuration is for FISim receiving request from a remote web server or Core Controller. The third MQ Series configuration is for the CWS web server sending requests to FISim running on a different machine.

The MQ Series configuration for the Core Controller sending requests to a remote FISim is basically the same as this third case except that MVS doesn't provide SMIT panels. In the case of the CWS web server and FISim in the same AIX machine and a Core Controller on an MVS machine, both machines have two pairs of queues defined. One pair is used by the CWS web server and the Core Controller, and the other pair is used by the Core Controller and FISim.

### All Local MQSeries Configuration

Without a Core Controller and with FISim on the same machine as the CWS web server, the MQ Series setup is simplified since everything is local. For this local only configuration, you need to define

- A queue manager
- A transmission queue, which is a local queue, for FISim to get messages from the web server
- Another local queue for the web server to receive messages from FISim
- A remote queue which relates a remote queue manager name to the local transmission queue

In this case, the remote queue is defined on the local machine as a trick to simplify setup. The reason is that the CWS web server addresses requests to multiple queues on the remote queue manager with one queue per transaction type (logon, account history, etc.). With a remote queue definition, the local queue manager places all these requests from the web server in a single transmission queue which FISim reads. The alternative would be to define alias queues (for nearly 200 transaction types) that point to the transmission queue.

**Adding a Queue Manager:** [If you will be using an already defined queue manager then you should skip to ,Queue Definitions-.]

Once you install MQSeries, add a **queue manager**. This queue manager will provide queuing services to applications and manage the queues that belong to it. The queue manager also ensures that messages are put on the correct queue.

Each queue manager must have a unique name. A queue manager may be shared by multiple users. Only the first user on each machine needs to add the queue manager.

In the following steps, `qmgrname` (all lowercase) is the name you assign to the queue manager and `userid` (all lowercase) is your user ID.

Use **SMIT** to add a queue manager as follows:

1. Type `smit`. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queue Managers*.
5. On the *Work with MQSeries Queue Managers* screen, select *Add a queue manager*.
6. On the *Add a queue manager* screen, in the *Queue Manager Name* entry field, type `qmgrname`



**Note:** Names in MQSeries are case sensitive.

In the *Description* entry field, type *qmgrname* queue manager for CWS

In the *Dead Letter Queue* entry field, type SYSTEM.DEAD.LETTER.QUEUE

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: ok** and **MQSeries queue manager created** or **Command: failed** and indicate why it failed.

**Note on dead letter queues:** A dead letter queue is a local queue where messages are routed if they cannot be routed to their correct destination. If you do not specify a dead letter queue, application program errors may cause channels to be closed. For example, if an application tries to put a message on a queue on another queue manager but a failure occurs, the channel is stopped and the message remains on the transmission queue. Then, other applications are blocked from using that channel for their messages. If you have a dead letter queue, the undelivered message is simply put on the dead letter queue at the receiving end, leaving the channel and its transmission queue available.

**Starting a Queue Manager:** Once you add the queue manager, start it through SMIT as follows:

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queue Managers*.
5. On the *Work with MQSeries Queue Managers* screen, select *Start a queue manager*.
6. On the *Queue Manager Name* screen, move the cursor to select the queue manager you want to start: *qmgrname*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** and **MQSeries queue manager started** or **Command: failed** and indicate why it failed.

**Adding Default Objects:** [Default objects may be added automatically during the Add queue manager step. If the default objects are already added then the *Add default objects to a queue manager* option mentioned in step 5 below will not be available and this section should be skipped.]

Once you start the queue manager, add default objects to it through SMIT as follows:

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queue Managers*.
5. On the *Work with MQSeries Queue Managers* screen, select *Add default objects to a queue manager*.
6. On the *Queue Manager Name* screen, move the cursor to select the queue manager you want: *qmgrname*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** and display several messages about MQSeries queues, channels, and processes created or **Command: failed** and indicate why it failed.

**Queue Definitions:** After you start the queue manager and add the default objects, add the queues you need to be able to use the CWS and FISim programs.

In MQSeries, there are three general types of queues:

Local queues, for transmitting and receiving messages

Alias queue objects

Remote queues

A **local queue** belongs to the queue manager to which the application is connected. CWS web server and FISim share two local queues: one for transmitting messages from CWS to FISim and one for receiving messages by CWS from FISim.

An **alias queue object** allows applications to access a queue by referring to it indirectly in MQI calls. An alias queue object is not a queue, but an object that you can use to access another queue. When an alias queue object is used in an MQI call, the name is resolved to the name of either a local or a remote queue at run time. This way, you can change the queues that your application uses without changing the application, you just have to change the alias queue object definition to reflect the name of the new queue.

The MQSeries setup for FISim could use a number of alias queues to map the output queue names from the Core Controller or from the CWS web server into the single transmit queue from which FISim reads. However, it is easier to define single remote queue which effects the desired concentration.

A remote queue is a type of alias queue that relates a remote queue manager name to a local transmission queue. A remote queue may be defined so that an application can send messages to a remote application as if it were a local application. A remote queue can also be used to direct requests for a remote queue manager to a local transmission queue. FISim uses a remote queue definition for this second reason and is thereby able to read web server requests addressed to multiple queue on a remote queue manager from a single transmission queue.

**Adding Local Queues:** For FISim to exchange messages with the CWS web server, add two local queues as follows:

- A queue for transmitting messages from the web server to FISim
- A queue for receiving messages from FISim by the web server

1. Type *smi t*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Local Queues*.
6. On the *Local Queues* screen, select *Add a local queue*.
7. On the *Queue Manager Name* screen, click the cursor to select the queue manager to which you want to add a local queue: *qmgrname*.
8. On the *Add a local queue* screen, in the *Queue Name* entry field, type the transmission queue name *userid.xmit*. This is the queue that FISim will read from.

In the *Description* entry field, type FISim transmission queue.

Toggle In the *Usage* field to transmission queue

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('userid.xmit') +
: DESCR ('FISim Transmission queue') +
: USAGE (XMITQ) AMQ8006: MQSeries queue created
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Use the same SMIT screens to add the queue for the web server to receive messages. Unlike the transmission queue, this queue will use the defaults for all fields except the queue name.

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Local Queues*.
6. On the *Local Queues* screen, select *Add a local queue*.
7. On the *Queue Manager Name* screen, click the cursor to select the queue manager to which you want to add a local queue: *qmgrname*.
8. On the *Add a local queue* screen, in the *Queue Name* entry field, type the queue name *userid.reply*. This is the queue that FISim will write responses to.

In the *Description* entry field, type *FISim reply queue*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('userid.reply') +
: DESCR ('FISim reply queue')
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

**Adding the Remote Queue:** Add the definition of the remote queue, which is used for concentrating the CWS web server requests.

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Remote Queues*.

6. On the *Remote Queues* screen, select *Add a remote queue*.
7. On the *Queue Manager Name* screen, click the cursor on the desired queue manager name *qmgrname*.
8. On the *Add a remote queue* screen, in the *Queue Name* entry field, type *userid*.

In the *Name of remote queue manager* entry field, type *userid*.

In the *Transmission queue* entry field, type *userid.xmit*

Click on OK.

SMIT displays the **COMMAND STATUS** screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: QREMOTE ('userid') +
: RQMNAME ('userid') +
: XMITQ ('userid.xmit')
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

## Remote MQ Series Configuration for FISim

The MQ Series configuration for FISim handling request from either a CWS web server or a Core Controller on a remote machine requires:

- A queue manager
- A transmission queue for sending messages to the remote machine
- A transmission queue for receiving messages from the remote machine
- A remote queue for the queue manager on the other machine
- A remote queue for relating a remote queue manager to the transmission queue that FISim reads
- A sender channel
- A receiver channel
- A trigger process for sending messages

Like the all local configuration, a remote queue is used to concentrate the requests into a single queue since they are addressed to multiple queues on a fictitious remote queue manager. The MQ Series configuration for the requestor (web server or Core Controller) is setup to send the requests to FISim machine for a multi-hop transfer to the fictitious queue manager. The remote queue used for concentrations points to a transmission queue that has no channel enabling FISim to read from it. The other remote queue does point to a transmission queue with a channel for returning responses to the requestor.

The following steps use the symbolic names which follow:

*requestor\_qm* - queue manager on the requestor machine running the CWS web server or the Core Controller

*responder\_qm* - queue manager on the responder machine running FISim

*fictitious\_qm* - nonexistent queue manager

*userid* - user ID running FISim

It may be convenient to name the queue managers the same as the TCP/IP hostname when you need only one queue manager on a machine.

**The Queue Manager:** Creating the queue manager, starting the queue manager, and adding default objects are the same as for All Local Configuration described above. On the FISim machine, use *responder\_qm* for the queue manager name instead of *qmgrname* used in the above section.

**Queue Definitions:** See , Queue Definitions- for a brief description of different queue types.

**Adding Local Queues:** For FISim to exchange messages with the CWS web server or Core Controller, add two local queues as follows:

- A queue for transmitting reply messages to the requestor machine
- A queue for receiving request messages from the requestor machine

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Local Queues*.
6. On the *Local Queues* screen, select *Add a local queue*.
7. On the *Queue Manager Name* screen, click the cursor to select the queue manager to which you want to add a local queue: *responder\_qm*.
8. On the *Add a local queue* screen, in the *Queue Name* entry field, type the transmission queue name *userid.reply*. This is the queue that will be used to send response messages to the remote machine.

In the *Description* entry field, type reply transmission queue.

Toggle In the *Usage* field to transmission queue.

Toggle the *MQGET operations* and the *MQPUT operations* fields to ENABLED.

Toggle the *Trigger Enabled* field to yes.

In the *Initiation Queue* field, type SYSTEM.CHANNEL.INITQ.

In the *Trigger Process* field, type userid\_CHL\_PROCESS.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('userid.reply') +
: DESCR ('reply transmission queue') +
: USAGE (XMITQ)
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Use the same SMIT screens to add the queue for receiving messages from the requestor machine.

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Local Queues*.

6. On the *Local Queues* screen, select *Add a local queue*.
7. On the *Queue Manager Name* screen, click the cursor to select the queue manager to which you want to add a local queue: *responder\_qm*.
8. On the *Add a local queue* screen, in the *Queue Name* entry field, type the queue name *userid.xmit*. This is the queue that receives messages from the remote machine.

In the *Description* entry field, type request transmission queue.

Toggle In the *Usage* field to transmission queue.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('userid.xmit') +
: DESCR ('request transmission queue') +
: USAGE (XMITQ)
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Once you have added the necessary local queues, add the remote queues

**Adding the Remote Queue:** Add the definition of the remote queue, which is used for the transmission of responses to the remote queue manager.

1. Type *smi t*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Remote Queues*.
6. On the *Remote Queues* screen, select *Add a remote queue*.
7. On the *Queue Manager Name* screen, click the cursor on the desired queue manager name *requestor\_qm*.
8. On the *Add a remote queue* screen, in the *Queue Name* entry field, type the name of the queue manager for the other machine, *requestor\_qm*.

In the *Name of remote queue manager* entry field, type the name of the queue manager for the other machine, *requestor\_qm*

In the *Transmission queue* entry field, type *userid.reply*

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: QREMOTE ('requestor_qm')
AMQ8006: MQSeries queue created
1 MQSC commands read.
```

```
0 commands have a syntax error.  
0 commands cannot be processed.
```

Add the definition of the remote queue, which is used for receiving requests from the remote queue manager.

1. Type *smi t*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Remote Queues*.
6. On the *Remote Queues* screen, select *Add a remote queue*.
7. On the *Queue Manager Name* screen, click on the desired queue manager name *responder\_qm*.
8. On the *Add a remote queue* screen, in the *Queue Name* entry field, type the name of the queue manager for the other machine, *fictitious\_qm*.

In the *Name of remote queue manager* entry field, type the name of the queue manager for the other machine, *fictitious\_qm*.

In the *Transmission queue* entry field, type *userid.xmit*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +  
: QREMOTE ('fictitious_qm')  
AMQ8006: MQSeries queue created  
1 MQSC commands read.  
0 commands have a syntax error.  
0 commands cannot be processed.
```

## Adding Channel Definitions

1. Type *smi t*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Channels*.
5. On the *Work with MQSeries Channels* screen, select *Add/Change/Show/Delete channel definitions*.
6. On the *Add/Change/Show/Delete channel definitions* screen, select *Sender channel definitions*.
7. On the *Sender channel definitions* screen, select *Add a Sender channel*.
8. On the *Queue Manager Name* screen click on the name of the queue manager, *responder\_qm*.
9. SMIT displays the *Add a sender channel* screen.

In the *Channel Name* field, type a name composed of the local queue manager, *responder\_qm*, a slash, and the remote queue manager, *requestor\_qm*.

In the *Connection Name* field, type the IP address and MQSeries port number for the remote (other) machine in the format *nnn.nnn.nnn.nnn(pppp)*. The default port number for MQSeries is 1414.

In the *Name of transmission queue* field, type *userid.reply*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: CHANNEL ('responder_qm/requestor_qm') +
: CHLTYPE (SDR) +
: CONNAME ('nnn.nnn.nnn.nnn(pppp)') +
: TRPTYPE (TCP) +
: XMITQ ('userid.reply')
AMQ8014: MQSeries channel created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

10. Back on the *Add/Change/Show/Delete channel definitions* screen, select *Receiver channel definitions*.
11. On the *Receiver channel definitions* screen, select *Add a Receiver channel*.
12. On the *Queue Manager Name* screen click on the name of the queue manager, *responder\_qm*.
13. SMIT displays the *Add a receiver channel* screen. In the *Channel Name* field, type a name composed of the remote queue manager, *requestor\_qm*, a slash, and the local queue manager, *responder\_qm*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: CHANNEL ('requestor_qm/responder_qm ') +
: CHLTYPE (RCVR) +
: TRPTYPE (TCP)
AMQ8014: MQSeries channel created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

## Adding Process Definitions

1. On the *MQSeries* screen, select *Work with MQSeries Process Definitions*.
2. On the *Work with MQSeries Process Definitions* screen, select *Add a process definition*.
3. On the *Queue Manager Name* screen, click the cursor on the queue manager name, *responder\_qm*.
4. On the *Add a process definition* screen, in the *Process Name* field, type *userid\_CHL\_PROCESS*.

In the *Application Type* field, type UNIX.

In the *User Data* field, type the sender channel name defined above.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
```



```

: PROCESS ('userid_CHL_PROCESS') +
: APPLTYPE (UNIX) +
: USERDATA ('responder_qm/requestor_qm')
AMQ8010: MQSeries process created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.

```

## Starting a Channel Initiator

1. On the *MQSeries* screen, select *Work with MQSeries Channels*.
2. On the *Work with MQSeries Channels* screen, select *Start/Stop channel initiators*.
3. On the *Start/Stop channel initiators* screen, select *Start channel initiator*.
4. On the *Queue Manager Name* screen, click on the queue manager name, *responder\_qm*.
5. On the *Channel Name* screen, click on the channel you want to start, *SYSTEM.CHANNEL.INITQ*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```

1 : start chinit +
: INITQ ('SYSTEM.CHANNEL.INITQ') +
AMQ8024: MQSeries channel initiator started
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.

```

To use *inetd* to start channels, configure two files.

- Add a line in the */etc/services* file:

```
MQSeries 1414/TCP
```

- Add a line in the */etc/inetd.conf* file:

```
MQSeries stream tcp nowait mqm /usr/lpp/mqm/bin/amqcrsta amqcrsta
-m responder_qm
```

## Displaying Channel Status

1. On the *MQSeries* screen, select *Work with MQSeries Channels*.
2. On the *Work with MQSeries Channels* screen, select *Display channel status*.
3. On the *Queue Manager Name* screen, click on the queue manager name, *responder\_qm*.
4. On the *Queue Name* screen, click on the queue name for which you want to display status.

Click on OK.

If the status of the channels looks incorrect, the sender or receiver machine is set up incorrectly, the process definitions are incorrect, or the channels are not started.

## Remote MQ Series Configuration for CWS Web Server

The MQ Series configuration for the CWS web server sending requests to FISim on a remote machine requires:

- A queue manager
- A transmission queue for sending messages to the remote machine
- A reply queue for receiving messages from the remote machine

- A remote queue for the queue manager on the other machine
- A sender channel
- A receiver channel

A trigger process for sending messages The following steps use the symbolic names which follow:

*requestor\_qm* - queue manager on the requestor machine running the CWS web server or the Core Controller

*responder\_qm* - queue manager on the responder machine running FISim

*fictitious\_qm* - nonexistent queue manager

*userid* - user ID running the CWS web server (or the Core Controller)

It may be convenient to name the queue managers the same as the TCP/IP hostname when you need only one queue manager on a machine.

**The Queue Manager:** Creating the queue manager, starting the queue manager, and adding default objects are the same as for the All Local Configuration described above. On the FISim machine, use *requestor\_qm* for the queue manager name instead of *qmgrname* used in the above section.

**Queue Definitions:** See ,Queue Definitions- for a brief description of different queue types.

**Adding Local Queues:** For FISim to exchange messages with the CWS web server or Core Controller, add two local queues as follows:

- A queue for transmitting request messages to the FISim machine
- A queue for receiving response messages from the FISim machine

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Local Queues*.
6. On the *Local Queues* screen, select *Add a local queue*.
7. On the *Queue Manager Name* screen, click on the queue manager to which you want to add a local queue: *requestor\_qm*.
8. On the *Add a local queue* screen, in the *Queue Name* entry field, type the transmission queue name *userid.xmit*. This is the queue that will be used to send response messages to the remote machine.

In the *Description* entry field, type *request transmission queue*.

Toggle In the *Usage* field to *transmission queue*.

Toggle the *MQGET operations* and the *MQPUT operations* fields to *ENABLED*.

Toggle the *Trigger Enabled* field to *yes*.

In the *Initiation Queue* field, type *SYSTEM.CHANNEL.INITQ*.

In the *Trigger Process* field, type *userid\_CHL\_PROCESS*.

Click on OK.

SMIT displays the **COMMAND STATUS** screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('userid.xmit') +
: DESCR ('request transmission queue') +
: USAGE (XMITQ)
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Use the same SMIT screens to add the queue for receiving messages from the requestor machine.

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen, select *Local Queues*.
6. On the *Local Queues* screen, select *Add a local queue*.
7. On the *Queue Manager Name* screen, click the cursor to select the queue manager to which you want to add a local queue: *requestor\_qm*.
8. On the *Add a local queue* screen, in the *Queue Name* entry field, type the queue name *userid.reply*. This is the queue that receives messages from the remote machine.

In the *Description* entry field, type requestor reply queue.

Click on OK.

SMIT displays the **COMMAND STATUS** screen. At first, the screen will read **Command: running**. Then it will read **Command: OK** or **Command: failed** with information about the success or failure of the command. You will see information like the following:

```
1 : define +
: QLOCAL ('userid.reply') +
: DESCR ('requestor reply queue')
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

Once you have added the necessary local queues, add the remote queues.

**Adding the Remote Queue:** Add the definition of the remote queue, which is used for the transmission of responses to the remote queue manager.

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Queues*.
5. On the *Work with MQSeries Queues* screen select *Remote Queues*.
6. On the *Remote Queues* screen, select *Add a remote queue*.

7. On the *Queue Manager Name* screen, click the cursor on the desired queue manager name *responder\_qm*.
8. On the *Add a remote queue* screen, in the *Queue Name* entry field, type the name of the queue manager for the other machine, *fictitious\_qm*.

In the *Name of remote queue manager* entry field, type the name of the queue manager for the other machine, *fictitious\_qm*.

In the *Transmission queue* entry field, type *userid.xmit*

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
l : define +
: QREMOTE ('fictitious_qm')
AMQ8006: MQSeries queue created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

## Adding Channel Definitions

1. Type *smit*. Press ENTER.
2. On the *System Management* screen, select *Communications Applications and Services*.
3. On the *Communications Applications and Services* screen, select *MQSeries*.
4. On the *MQSeries* screen, select *Work with MQSeries Channels*.
5. On the *Work with MQSeries Channels* screen, select *Add/Change/Show/Delete channel definitions*.
6. On the *Add/Change/Show/Delete channel definitions* screen, select *Sender channel definitions*.
7. On the *Sender channel definitions* screen, select *Add a Sender channel*.
8. On the *Queue Manager Name* screen click the cursor on the name of the queue manager, *requestor\_qm*.
9. SMIT displays the *Add a sender channel* screen. In the *Channel Name* field, type a name composed of the local queue manager, *requestor\_qm*, a slash, and the remote queue manager, *responder\_qm*.

In the *Connection Name* field, type the IP address and MQSeries port number for the remote (other) machine in the format *nnn.nnn.nnn.nnn(pppp)*. The default port number for MQSeries is 1414.

In the *Name of transmission queue* field, type *userid.xmit*

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads **Command: RUNNING**. Then it reads either **Command: OK** or **Command: failed** with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
l : define +
: CHANNEL ('requestor_qm/responder_qm') +
: CHLTYPE (SDR) +
: CONNAME ('nnn.nnn.nnn.nnn(pppp)') +
: TRPTYPE (TCP) +
: XMITQ ('userid.xmit')
AMQ8014: MQSeries channel created
1 MQSC commands read.
```

```
0 commands have a syntax error.
0 commands cannot be processed.
```

10. Back on the *Add/Change/Show/Delete channel definitions* screen, select *Receiver channel definitions*.
11. On the *Receiver channel definitions* screen, select *Add a Receiver channel*.
12. On the *Queue Manager Name* screen click the cursor on the name of the queue manager, *requestor\_qm*.
13. SMIT displays the *Add a receiver channel* screen. In the *Channel Name* field, type a name composed of the remote queue manager, *responder\_qm*, a slash, and the local queue manager, *requestor\_qm*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads *,Command: RUNNING-*. Then it reads either *,Command: OK-* or *,Command: failed-* with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: CHANNEL ('responder_qm/requestor_qm ') +
: CHLTYPE (RCVR) +
: TRPTYPE (TCP)
AMQ8014: MQSeries channel created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

## Adding Process Definitions

1. On the *MQSeries* screen, select *Work with MQSeries Process Definitions*.
2. On the *Work with MQSeries Process Definitions* screen, select *Add a process definition*.
3. On the *Queue Manager Name* screen, click on the queue manager name, *requestor\_qm*.
4. On the *Add a process definition* screen, in the *Process Name* field, type *userid\_CHL\_PROCESS*.

In the *Application Type* field, type *UNIX*.

In the *User Data* field, type the sender channel name defined above.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads *,Command: RUNNING-*. Then it reads either *,Command: OK-* or *,Command: failed-* with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : define +
: PROCESS ('userid_CHL_PROCESS') +
: APPLTYPE (UNIX) +
: USERDATA ('requestor_qm/responder_qm')
AMQ8010: MQSeries process created
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

## Starting a Channel Initiator

1. On the *MQSeries* screen, select *Work with MQSeries Channels*.
2. On the *Work with MQSeries Channels* screen, select *Start/Stop channel initiators*.
3. On the *Start/Stop channel initiators* screen, select *Start channel initiator*.
4. On the *Queue Manager Name* screen, click on the queue manager name, *requestor\_qm*.

5. On the *Channel Name* screen, click the cursor on the channel you want to start, *SYSTEM.CHANNEL.INITQ*.

Click on OK.

SMIT displays the *COMMAND STATUS* screen. First, the screen reads *,Command: RUNNING-*. Then it reads either *,Command: OK-* or *,Command: failed-* with information about the success or the failure of the command. Toward the bottom of the screen, you should see something like:

```
1 : start chinit +
: INITQ ('SYSTEM.CHANNEL.INITQ') +
AMQ8024: MQSeries channel initiator started
1 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

To use *inetd* to start channels, configure two files.

- Add a line in the */etc/services* file:

```
MQSeries 1414/TCP
```

- Add a line in the */etc/inetd.conf* file:

```
MQSeries stream tcp nowait mqm /usr/lpp/mqm/bin/amqcrsta amqcrsta
-m responder_qm
```

## Displaying Channel Status

1. On the *MQSeries* screen, select *Work with MQSeries Channels*.
2. On the *Work with MQSeries Channels* screen, select *Display channel status*.
3. On the *Queue Manager Name* screen, click on the queue manager name, *requestor\_qm*.
4. On the *Queue Name* screen, click the cursor on the queue name for which you want to display status.

Click on OK.

If the status of the channels looks incorrect - the sender or receiver machine is set up incorrectly, the process definitions are incorrect, or the channels are not started.

---

## Using FISim

FISim requires DB2 UDB version 7.1 installation and setup.

## DB2 Configuration

This appendix assumes the user has already installed DB2 UDB 7.1.

1. Add this line to your *.profile* to setup the default DB2 environment:

```
. /home/db2inst1/sqllib/db2profile
```

Due to enhancements to support NLS, some tables storage requirements for a single row exceed four kilobytes, which is a default limitation of DB2. In order to setup the FISim database, the following administrative steps (Steps 2 - 9) will need to be followed when creating or preparing a database for use with V1R6 FISim. Once Steps 2 through 9 have been completed, **it will not be necessary to repeat these steps unless the database is dropped**. These new configuration steps will not prevent you from using the database for previous versions of FISim, such as V1R5.

2. To create a database, type the following command at the AIX prompt:

```

db2 [invokes db2 interactive command mode]
db2 => db2start [only necessary if db2 manager has not been started]
db2 => create database dbname

```

3. To connect to the database, type the following command at the DB2 prompt:

```

db2 => connect to dbname

```

4. To create a DB2 bufferpool for use within your database, type the following command at the DB2 prompt:

```

db2 => create bufferpool buffpoolname size 1000 pagesize 16k

```

where *buffpoolname* is a unique name for your bufferpool. 16k is the minimum pagesize required for the V1R6 FISim database.

Creating a new bufferpool is not necessary if a 16k pagesize bufferpool already exists for the current database. To view a list of bufferpools, type the following at the DB2 prompt:

```

db2 => select * from syscat.bufferpools

```

5. To use the new bufferpool, the DB2 manager must be stopped and restarted. This step requires that ALL connections to the database manger be terminated. To view a list of connected applications, type the following command at the DB2 prompt:

```

db2 => list applications

```

To stop and restart the database manager, type the following command at the DB2 prompt:

```

db2 => stop database manager
db2 => start database manager

```

6. To reconnect to the database, type the following command at the DB2 prompt:

```

db2 => connect to dbname

```

7. To create a tablespace that uses the new bufferpool, type the following command at the DB2 prompt:

```

db2 => create tablespace tbspacename pagesize 16k managed
by system using ('tbspacename') bufferpool buffpoolname

```

- The *buffpoolname* specified must match the name of the bufferpool created in Step 4.
- The pagesize specified (16k in this example) must match the pagesize of the specified bufferpool.
- The *tbspacename* must be unique. To view a list of defined tablespaces, type the following command at the DB2 prompt:

```

db2 => select * from syscat tablespaces

```

8. Test the usability of the new tablespace. Create a wide table by typing the following command at the DB2 prompt:

```

db2 => create table widetest (x varchar (16000))

```

If the table is successfully created, the database *dbname* is ready for use with V1R6 FISim. Drop the test table by typing the following command at the DB2 prompt:

```

db2 => drop table widetest

```

9. To exit DB2 interactive command mode, type the following at the DB2 prompt:

```

db2 => quit

```

10. A few environment variables should be set to avoid the need for the command line parameters when setting up and running FISim. Type the following command at the AIX prompt:

```

export FISDB2INST=dbname
export FISQMGR=qmgrname
export FISXMIT=userid.xmit

```

where *dbname* is the name of the database, *qmgrname* is the name of the local queue manager, and *userid.xmit* is the name of the queue that FISim reads from.

You may want to set these environment variables in your .profile.

11. To setup the database and load the initial change to the `.../fis/data` directory and type the following command at the AIX prompt:

```
setupfis -d dbname -u userid -p password
```

where *dbname* is the name of the database, and *userid* and *password* correspond to a user that can access the database. The command line parameters may be omitted if the environment variables were set as described in Step 3 above and the user executing the command has permissions to write to the database *dbname*.

A large number of messages from the database manager will be displayed. Some of these messages will indicate a failure to delete a table the first time you run *setupfis*. These ,delete- commands will be needed when you rerun *setupfis* to restore the initial data.

12. When using FISim with an actual Core Controller, the FISim DB2 database must be modified to indicate this configuration. The `REAL_CC` column of the `IFS_SYSTEM` DB2 table is used to indicate whether or not FISim is being used in conjunction with a Core Controller. The default value of this column when the database is initially set up is `NULL`, which indicates that a Core Controller is **not** being used. Update this column to any non-NULL non-zero value to indicate that a Core Controller **is** being used. If this modification is not made, FISim will populate fields of certain Gold transactions that the Core Controller should be populating, which causes an error in the Core Controller.

Type the following command at the AIX prompt:

```
db2
db2 => connect to dbname
db2 => update ifs_system set real_cc = <null for no core controller,
non-zero for core controller>
db2 => quit
```

Each time you reboot the machine, you will need to start the database manager again. This can be automated if root would add this line to `/etc/inittab`

```
rcdb2:2:wait:/etc/rc.db2 >/dev/console 2>&1 # Start db2
```

and create a file in the `/etc` directory containing the following:

```
#!/bin/ksh
#-----
# Start db2 database manager
#
# Note: We do this by becoming the db2inst1 user,
# and running the script:
# /home/db2inst1/sqllib/adm/db2start
#
#-----
su - db2inst1 '-c \'
. /home/db2inst1/sqllib/db2profile; \'
/home/db2inst1/sqllib/adm/db2start '
```

## Running FISim

To run FISim after running *setupfis*, go to the `.../fis/bin-` directory under the directory where FISim was installed and type

```
runfis -d dbname -u userid -p password -q qmgrname -x userid.xmit &
```

where *dbname* is the name of the database, *userid* and *password* correspond to a user that can access the database, *qmgrname* is the name of the local queue manager, and *userid.xmit* is the name of the queue that FISim reads from. The command line parameters may be omitted if the environment variables were set as described in Step 3 above and the user executing the command has permissions to write to the database *dbname*.

You should see a number of messages displayed on the terminal. Look for these messages in particular:

```
.
.
Binding was ended with "0" errors and "0" warnings.
.
```



```
.  
FiSimulator initializing  
FiSimulator initialization complete
```

You should now be able to start the CWS web server and optionally the Core Controller and receive responses from FISim.

Now, you will need a user ID, password, and pin that match the DB2 database. For start, use ,ARF1001, 1001, and 1001-. These values are case sensitive.

You can stop FISim with either the *killfis* command or by getting its process id (pid) with a ,ps x- command and using the Unix command ,kill -9 pid-.

## Basic Data Management

The simplest way to view the contents of the DB2 database is to issue direct queries. Type the following at the AIX prompt:

```
db2  
db2 => db2start (if needed)  
db2 => connect to dbname  
db2 => select user_id from ifs_user  
db2 => select * from ifs_user where user_id='ARF1001'  
db2 => list tables  
db2 => quit
```

Note that DB2 is not case sensitive except inside of the quotes which must be single quotes. As you can see, ,ifs\_user- is a table name and ,user\_id- is a column name. Using the ,select- statement, you can view all of the data in the tables provided with the enhanced simulation.

Using the ,update- and ,insert- statements, you could modify and augment the data in the tables provided although these updates will have to satisfy the rules of the database. See the DB2 [Command Reference](#) for syntax of these commands.

A recommended approach for modifying data is to create a db2 script containing only the required changes. This script can then be rerun to restore the values to a known state after installing a new release and after experimenting with temporary changes. Please note, however, that future updates of FISim will have to add/alter columns as Gold message definitions change. Also, design changes in FISim could significantly alter table layouts. This means that your data updates may have to be reworked to accommodate table definition changes.

## Data Management using Approach or Excel

A Lotus Approach or Microsoft Excel spread sheet on a Windows system can be used to view and update data in the DB2 tables on the AIX system with some additional setup.

On the Windows system, do the following:

1. Install Lotus Approach or Microsoft Excel
2. Install IBM Universal Database Version 7.1 - DB2 Connect Personal Edition
3. Using the DB2 Client Configuration Assistant, add the database on the AIX system using manual configuration, TCP/IP, LAN based, your host name and port 50000, your database name, and register for ODBC.
4. Using the spreadsheet program, Open with file type of ODBC and find your database name.
5. When viewing a table, overwrite an entry and press enter to change the value in the database on the AIX system. Use control-r to refresh the view from AIX.

Please note that future updates of FISim will have to add/alter columns as Gold messages are added or modified. Also, design changes in FISim could significantly alter table layouts. This means that your data updates through the spreadsheet will be lost when you move to a new version of FISim because you will have to reload new tables from the new version.

---

## Altering Response Codes

FISim can simulate error conditions by responding to transactions with response codes that you specify. By issuing a configuration command to FISim, you can indicate the type of transaction to modify, the values to be sent in the response message, and optionally, when the new response should take effect.

Commands are issued to FISim through CSTEMAIL text messages. These can be sent either through GMIT or, if CWS is being used, through the bank messages web page. The message body should contain the keyword \$fisim, followed by the desired FISim command. For example, the message text:

```
$fisim clrrc txn *
```

will clear all previous configuration commands from FISim.

The message subject is ignored.

After processing a command, FISim sends back an email message containing its completion status. If you are issuing commands through CWS, you can return to the message list page and view the last message in the list to see FISim's response, including any error messages.

## Command Descriptions

The available commands are:

*setrc* - enter an instruction to force the compcode, respcode, sense data and/or BRC on a subsequent transaction.

*clrrc* - clear the instructions made through setrc.

From the web pages, the arguments to the SETRC and CLRRRC commands must be followed with a BLANK character in order to be accepted by FISim. This restriction is because the email page appends data to the end of the input message.

Commands can be entered in any case, but they are converted to uppercase before processing. Commands may be terminated with a semicolon, but this is not required.

In the command syntax diagrams, the following conventions are used:

*UPPERCASE TEXT* indicates keywords to be entered as shown,

*<description>* indicates a parameter argument,

*[ something ]* indicates an optional parameter, and

*xxx | yyy* indicates alternatives.

**SETRC:** The setrc command makes an entry in a table (IFS\_SETRC). All subsequent transactions received by FISim are compared against the table. When a match is found, FISim generates an error response, instead of performing its normal processing. After the entry is used, it is deleted from the table. The command syntax is:

```
SETRC TXN <transaction_name> [SKIP <skip count>]  
      [CC <compcode value>]  
      [RC <respcode value>]  
      [SD <sensedata value>]
```

[BRC <business response code value>]  
[WARNING <T | F>]

Table 15. SETRC parameters

Parameter Keyword	Argument Description
TXN	The short name (<= 8 chars) of the transaction as it appears in the goldtrace log.
SKIP	The number of transactions of this type to skip before applying this response. If this parameter is not specified, it defaults to 0. For example, if this parameter is set to 1 for the <b>avartfr</b> transaction, process one <b>avartfr</b> normally and then apply this rule to the next <b>avartfr</b> transaction.
CC	A decimal value indicating the completion code value for the header.
RC	A decimal value indicating the response code value for the header.
SD	A string value (up to 256 characters) indicating the sense data string for the header. Since parameter values are delimited by blanks, the sensedata string cannot contain any embedded blanks.
BRC	The business response code as a 3-character string of decimal digits. If this parameter is specified, FISim will generate an IFS_RESP_CODE or IFS_RESP_DTL structure in the response message with this value in the resp_code field.
WARNING	When set to T, FISim will execute the transaction, and will generate the specified error response if the transaction completes successfully. When set to F, FISim will not run the transaction, and will always return the specified error response. This is the default behavior.

**CLRRRC:** - The *clrrc* command deletes entries from the IFS\_SETRC table.

*CLRRC TXN <transaction\_name> | \**

Table 16. CLRRRC parameters

Parameter Keyword	Argument Description
transaction_name	All entries for this transaction are deleted.
*	All entries in the table are deleted.

## Examples

*\$fisim setrc txn accthist brc 078 skip 2*

Generate a response to the third ACCTHIST transaction, producing an IFS\_RESP\_CODE value of ,078-.

*\$fisim setrc txn ifsping cc 2*

Generate a response to the next IFSPING transaction, with a completion code of 2.

*\$fisim clrrc txn accthist*

Clear all ACCTHIST entries from the setrc table.

*\$fisim clrrc txn \**

Clear all entries from the setrc table.

*\$fisim setrc txn avartfr cc 4 rc 2012 sd 1:12345:12345*

Respond to the next avartfr with a completion code of 4, a reason code of 2012 and sense data of 1:12345:12345.

## Entering Commands Directly Through DB2

Alternatively, instead of issuing a *setrc* instruction, you can use DB2 to insert a row into the IFS\_SETRC table, bypassing the *cstemail* transaction completely. Each option corresponds to a column in the table. For example, to enter the equivalent of the last sample transaction, enter the following text at a shell prompt:

```
db2 ":insert into ifs_setrc
(txn_name,skip_count,comp_code,reason_code,sense_data,brc,warning)
values ('AVARTFR',0,4,2012,'12345:12345',null,0)"
```

The column names correspond to the command arguments as follows:

Table 17. DB2 Column Names for IFS\_SETRC Table

Column name	Description
txn_name	The short name of the transaction used in the message header.
skip_count	The number of transactions to be skipped before applying this rule.
comp_code	The completion code to be returned
reason_code	The reason code to be returned
sense_data	The sense data string to be returned
brc	The BRC to be returned (or NULL if no BRC should be returned)
warning	0 = False 1 = True

## Troubleshooting

### FISim Trace Files and Logs

FISim creates trace files that track the details of execution of FISim. The default location of these files is the */tmp* directory. Three different files are written to during the execution of FISim:

1. *<AIXUserID>.ifsdlog* - provides a runtime log of general trace and debug information regarding the startup of FISim and any Gold Transactions that are not associated with a particular IFS user (e.g., SRVSTART or CODERFSH).
2. *<AIXUserID>.fisgold* - provides a trace of the contents of all Gold Message Requests that are received by FISim and all Gold Message Responses that are generated by FISim.
3. *<AIXUserID>.<IFSUserID>* - provides a runtime log of trace and debug information corresponding to a particular IFS User executing Gold Transactions.

The *AIXUserID* corresponds to the AIX user ID that started the FISim executable.

The *IFSUserID* corresponds to the IFS user ID indicated in the SessionKey.userID in the header of a Gold Message.

A location other than the default (*/tmp*) may be specified prior to starting FISim by setting the *FISIM\_LOG\_PATH* environment variable. *FISIM\_LOG\_PATH* should be set to an existing subdirectory, for example:

```
export FISIM_LOG_PATH=/tmp/mytraces
```

where */tmp/mytraces* is a subdirectory that has already been created and can be written to by the AIX user that starts FISim. If *FISIM\_LOG\_PATH* has been set when FISim is started, the logs and traces will be written to the directory specified without the *AIXUserID* as part of the name, so the names of the files will be *ifsdlog*, *fisgold* and *IFSUserID*.

### FISim Returns a BRC of 247 Due to SQLCODE of -954

BRC 247 indicates a ,technical- problem, and for FISim, this is often due to a DB2 error. The FISim log file for the particular user should be examined to determine what exactly failed. The following is an example of a DB2 error as it would appear in an FISim log file:

```
2000-04-17 14:26:31 FISIM:
2000-04-17 14:26:31 FISIM: ** DB2 Error Condition Encountered
```

```

2000-04-17 14:26:31 FISIM: ** -----
2000-04-17 14:26:31 FISIM: ** Message - Select failed
2000-04-17 14:26:31 FISIM: ** sqlcode - -954
2000-04-17 14:26:31 FISIM: ** sqlerrm -

2000-04-17 14:26:31 FISIM: ** sqlwarn -
2000-04-17 14:26:31 FISIM: ** sqlstate - 57011
2000-04-17 14:26:31 FISIM:
2000-04-17 14:26:31 FISIM:
2000-04-17 14:26:31 FISIM: DB2 specific failure/problem. . .

```

SQLCODE -954 indicates that not enough storage space is available in the application heap to process the statement, because all available memory for the application has been used.

When this error is encountered, the heap space should be expanded to increase the memory available to the application.

Type the following DB2 command at the AIX prompt to check the heap size of your DB2 configuration:

```
db2 ,get db cfg for dbname-
```

Type the following DB2 commands at the AIX prompt to change the heap size to 256:

```

db2 "update db cfg for dbname using APPLHEAPSZ 256"
db2 "update db cfg for dbname using APP_CTL_HEAP_SZ 256"

```

FISim must be stopped and restarted to take advantage of the new heap size.

If this error continues to occur after increasing the heap size to 256, we recommend increasing the size to 600.

---

## Predefined Responses

Transaction responses are generated in one of the following ways:

1. If there is a predefined response file indicated for the particular transaction in the user configuration file, then FISim attempts to deliver the contents of the predefined file as the Gold Response. The response file can be either a binary file containing Gold bitstream data, or a text file containing a description of a Gold Response in the required format.
2. If no predefined response file was indicated, FISim's internal business logic is used to generate a Gold response based on the contents of the database.
3. If FISim has not implemented any logic for a particular transaction, FISim will pass to the MQWeb library and return a ,canned- Gold response.

This section deals with responses of the first type, and shows how to configure predefined responses for FISim.

In order to use predefined responses, the user submitting the request must have an associated *user configuration file* that maps response files to transaction types. These files are described in more detail below.

## Important Points to Consider

- Response files can contain either Gold bitstream data or goldmine input. If the response file is a text file, FISim will attempt to convert the file to Gold bitstream format by passing it to an internal goldmine converter. This can delay the delivery of the response message. (The default timeout value in the goldbank configuration file can be increased accordingly to avoid timeout errors.)
- The response file contains the whole response, including the message header. Any fields left unspecified are not returned in the response.
- If any errors are encountered during the processing or delivery of a response file, FISim returns an IFS\_RESP\_CODE structure containing a BRC 247 (Technical Difficulties) instead.

## Environment Variables

The FISIM\_PATH environment variable can be set using a directory name or a colon-separated list of directory names to indicate the directories to be searched when the system looks for user configuration files. The directory or directories specified in FISIM\_PATH will be searched first, followed by the current directory.

Path names should be fully qualified to avoid ambiguity.

*For example, if FISIM\_PATH=/home/fisim/test:/home/fisim/old/test, then /home/fisim/test will be searched first, followed by /home/fisim/old/test, followed by the current directory.*

### User Configuration Files

There must be one user configuration file for the user submitting the transaction. This is a text file that maps the user's response files to their respective transactions, and names the directory containing all of the response files.

FISim will attempt to use a response file as the transaction response if **both** of the following conditions are met:

1. FISim locates a configuration file for the user in one of the directories specified in *FISIM\_PATH*, as described above.
2. The configuration file contains a *transaction line* for the specified transaction (described below).

If either of these conditions is not met, then the transaction response must be constructed from FISim business logic.

Configuration files are named *fis.USERID.config*, where *USERID* must match the user ID specified in the SessionKey.userID in the header of the incoming Gold Request Message. The file name *fis.config* is reserved for use with transactions (e.g., SRVSTART or CODERFSH) where the user ID is not specified.

User configuration files may be edited while FISim is running. FISim will automatically reload data from modified configuration file. (Note, however, that FISim relies on file timestamps to detect modified files. Files accessed over network-mounted file systems may have unreliable timestamps.)

The contents of a user configuration file follow the following rules:

Blank lines and lines starting with a *#* character (used for comments) are ignored by FISim.

The first non-blank, non-comment line in the file must be the name of a directory. This indicates the location of all of the user's response files. This line must begin with a whitespace character, and the directory name should be fully qualified.

- A line that does not begin with a whitespace character is a *transaction line*. It consists of the short name of a transaction in all capital letters. This line is followed by one or more *file specification lines*, which indicate response files for this type of transaction.
- A line that begins with a whitespace character is a *file specification line*. It contains the name of a file (which must be present in the previously-defined directory).
- There may be multiple file specification lines for a single transaction type. When FISim needs to send out a response for a transaction of this type, it will rotate through the different responses in the order that they appear in the file.

A sample user configuration file appears below:

```
# Sample UserConfig file for FISIM (fis.ARF1001.config)
```

```
# Comment line
```

```
# Search Directory for response files
```

```
/home/fisim/test1001
```

```
# First Transaction (LOGON) - Alternate responses  
LOGON
```

```

ARF1001.LOGON.resp.dat
ARF1001.LOGON_FAIL.resp.dat
# (LOGOFF) - Always respond with same response
LOGOFF
ARF1001.LOGOFF.resp.dat
# (BALINQ) Cycle responses through CHECKING/SAVINGS/IRA
BALINQ
ARF1001.BALINQ_CHECKING.resp.dat
ARF1001.BALINQ_SAVINGS.resp.dat
ARF1001.BALINQ_IRA.resp.dat

```

## Response Data Files

All response files for a particular user are stored in a single directory. As seen above, the user configuration file indicates the location of the response files and maps the response files to their respective transactions.

The *fis/testdata* directory contains response file templates for every transaction supported by the current MTI level. The names of all of the files in this directory end with the suffix, *.resp.dat*. The simplest approach to creating new response data files is to copy a response file of the appropriate type from this directory, edit its contents as necessary, and place the file in the user's response directory, taking care to ensure that the file name matches the entry in the user configuration file.

These response data files contain two sections; a Header and a Message Body. The fields in the Header are the Gold header fields defined by IFS. *<Header>* indicates the beginning of the Header section, and *</Header>* indicates the end of the Header section. The following are examples of fields that may be edited in the Header section:

Table 18. *<Header> Record Functions*

<b>&lt;Header&gt; Section</b>	<b>Function</b>
RequestType	Defines the request/response type for the Gold Message being built. For example: <i>RequestType LOGON</i>
SessionKey.fi	Defines the FI bank name used for testing. For example: <i>SessionKey.fi IBANKA</i>
SessionKey.userID	Defines the name of a consumer's userid. For example: <i>SessionKey.userID ARF1001</i>

The Message Body contains the structures and fields as defined in the Gold Message Responses. *<Message\_Body>* indicates the beginning of the Message Body section, and *</Message\_Body>* indicates the end of the Message Body section. The structures in the Message Body are composed of tag-value pairs representing the various fields in the format:

```
<Tag Name> <Tag Data> # <Tag Description> (<Data Type> <Length>)
```

Comments are always preceded with #. Do not place comments within structure level tags unless they follow a tag-value pair.

## Repeat Items

Structures that can have multiple occurrences are defined in the data files with the comment value *,...(Repeat Item)-*. Each occurrence is referred to as a single structure instance. By default the data files are generated with only a single occurrence of Repeat Item structures. Data files that contain Repeat Items structures can be modified to include multiple structure instances of a Repeat Item structure. To include multiple structure instances of a Repeat Item structure the structure instance(s) must be enclosed by a **<Repeat\_Structure>** and **</Repeat\_Structure>** statements with the number of structure instances specified on the **<Repeat\_Structure>** line.

Using the ADDBSINV (Add Business Invoice Transaction) data file as an example the structure tag *InvLineDtITg* is defined as a Repeat Item, as shown;

```

<Structure> InvLineDtITg          #...(Repeat Item)
  InvLineNbrIdTg 1                # Invoice Line Number ( char [12] )
  <Structure> InvLineAmtTg        #...
    <Structure> AmtTg              # Amount
      Dcm1DataValTg 6835          # Decimal Data (char [15])
      Dcm1PrecValTg 2             # Decimal Precision (long)

```

```

        DrCrIndTg 1                # Debit/Credit Indicator (long)
    </Structure>
    IsoCurrCodeTg USD              # ISO Currency Code (char[3])
</Structure>
    InvLineMemoNmTg memo txt      # Line Memo Text ( char[80])
</Structure>

```

To create multiple structure instances of the Repeat Item structure edit the data file as shown;

```

    <Repeat_Structure> 2           # repeat count of 1 or greater must be specified
    <Structure> InvLineDt1Tg       #...(Repeat Item) Structure Instance 1
        InvLineNbrIdTg 1         # Invoice Line Number ( char [12] )
    <Structure> InvLineAmtTg       #...
        <Structure> AmtTg         # Amount
            Dcm1DataValTg 6835    # Decimal Data (char [15])
            Dcm1PrecValTg 2       # Decimal Precision (long)
            DrCrIndTg 1           # Debit/Credit Indicator (long)
    </Structure>
    IsoCurrCodeC1Tg USD          # ISO Currency Code (char[3])
</Structure>
    InvLineMemoNmTg memo txt      # Line Memo Text ( char[80])
</Structure>
    <Structure> InvLineDt1Tg       #...(Repeat Item) Structure Instance 2
        InvLineNbrIdTg 2         # Invoice Line Number ( char [12] )
    <Structure> InvLineAmtTg       #...
        <Structure> AmtTg         # Amount
            Dcm1DataValTg 2503    # Decimal Data (char [15])
            Dcm1PrecValTg 2       # Decimal Precision (long)
            DrCrIndTg 1           # Debit/Credit Indicator (long)
    </Structure>
    IsoCurrCodeC1Tg USD          # ISO Currency Code (char[3])
</Structure>
    InvLineMemoNmTg memo txt      # Line Memo Text ( char[80])
</Structure>
</Repeat_Structure>

```

The Repeat Count specified on the **<Repeat\_Structure>** line must equal the number of Structure Instances you add; otherwise an error will occur informing you that there are too many or too few structure instances defined. The structure instances must be defined inside the **<Repeat\_Structure>** and **</Repeat\_Structure>** statements.

## Goldmine

A response file can be converted into Gold binary format by running it through **goldmine** first, and using the resulting **.gold** file as the response file. In any case, it is a good idea to use **goldmine** to verify your source file syntax prior to running any tests. For more information on **goldmine**, see Appendix C, ,Gold Message Interface Tool (GMIT)-.



---

## Appendix E. IFS PvC Starter Kit

---

### Introduction

The Interactive Financial Services (IFS) platform is designed to facilitate on-line connections between consumer access devices and Financial Institutions. IFS enables banks to offer their customers a variety of electronic banking services, including balance inquiries, transfers of funds, bill presentment and bill payment.

Currently, PCs using web browsers or Personal Financial Management (PFM) applications can access the IFS platform. This appendix covers the IFS ,PvC Starter Kit-, that enables Pervasive Computing (PvC) handheld devices, such as a Personal Digital Assistant (PDA) to also access the IFS platform. The PvC Starter Kit supports account inquiries and fund transfers, but does not support bill presentment and bill payment.

### Solution Architecture

The functionality of the IFS Customizable Web Server (CWS) is extended by providing PvC Java Servlets to support PalmOS and WinCE handheld devices. The PvC Servlets enable the CWS to interface to leading industry mobile network servers. AvantGo Enterprise Interactive, Version 3.1 (or higher release) is the IFS recommended prerequisite software for the mobile network server.

Mobile Network software packages like AvantGo Enterprise Interactive typically include three components:

Client

Server

Connect

The Client component has a web-based user interface and integrates data security. For example, the AvantGo client uses Certicom's Elliptic Curve Cryptography (ECC) for data security. The Server component also integrates data security and provides management of the remote devices. The Connect component provides the mobile connection between the client and server components. This appendix describes the CWS extensions required to satisfy the interface to the Mobile Network Server and to provide the PvC Starter Kit level of functionality for the handheld clients.

The following figure provides an architectural view of the IFS PvC Solution. The CWS architecture depicts the Java Virtual Machine executing the IFS PvC Servlets to provide the functionality for handheld devices. The handheld devices communicate via a modem (not shown in the figure) to a Mobile Network Server (for example, AvantGo) that in turn interfaces with the CWS. From an interface standpoint, the Mobile Network Server looks like a web browser to CWS.

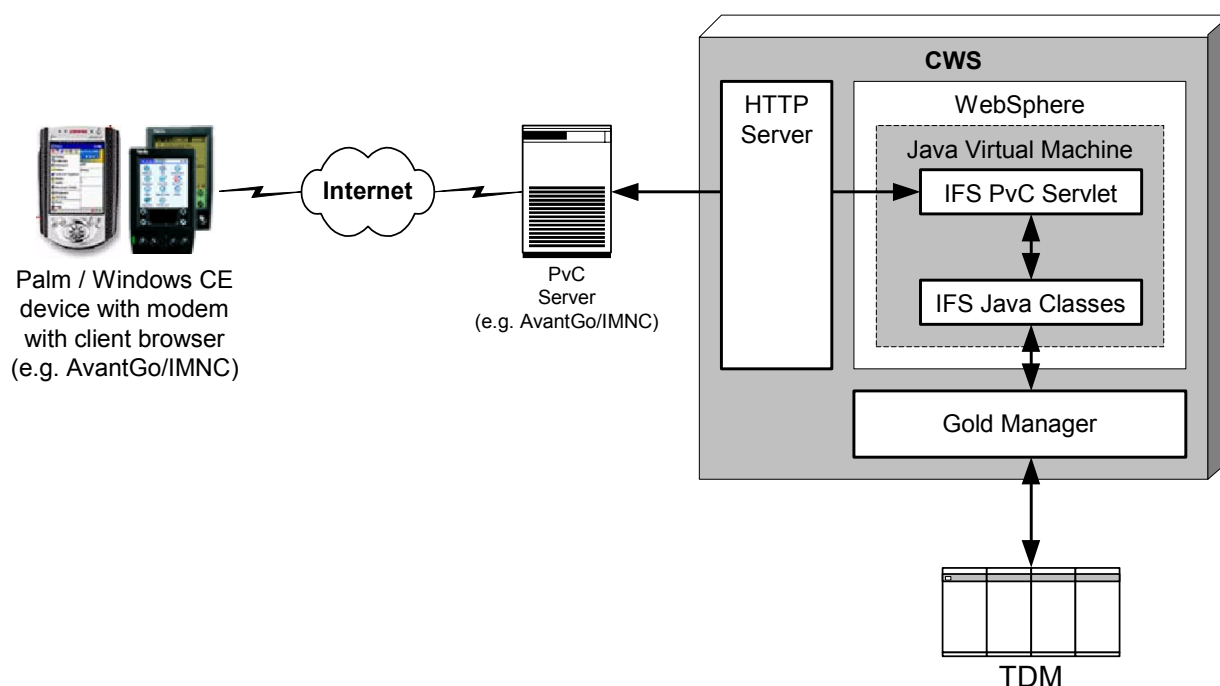


Figure 15. IFS Architecture for PvC Support

## Functions Supported

This section describes the functions that are implemented with the IFS PvC Servlets. It should be noted that NLS/DBCS and SME functions are not supported in the PvC Starter Kit.

### User Authorization Functions

- Logon to IFS system
- Logoff IFS system

### Core Banking Functions

The following defines the specific core banking functions and account types supported by the PvC Servlets.

#### 1. Account Inquiries

- Account Summary: display a summary of all accounts.
- Account Details: display detailed information on the selected account.
- Transaction Search: display selected account transaction history.
- Transaction History: display complete account transaction history.
- Balance Inquiry: display various balances associated with a specific account.
- Interest Rates: display interest rates.

#### 2. Account Types Supported

- Checking
- Saving
- Money Market
- Certificate of Deposit
- Individual Retirement Account
- Credit Card
- Equity Line of Credit
- Consumer Loan
- Commercial Loan
- Mortgage

#### 3. Fund Transfers

- Add fixed amount recurring transfer.

- Add variable amount single transfer.
- Change scheduled transfer, including recurring and single transfer.
- Delete scheduled transfer, including recurring and single transfer.
- Transfer inquiry: display the customer's fund transfer history.

#### 4. Customer Services

- Check Copy Order: request a copy of a check used in a particular transaction.
- Check Book Reorder: allows the consumer to re-order checks from the FI.
- Stop Cheque Payment: allows a consumer to place a stop payment on a check.

**Customization Approach:** In this release, if the FI wants to customize the user interface, they have to modify some of the PvC servlets.

## System Requirements

The following sections define the hardware and software prerequisites for the IFS PvC support.

### Hardware Requirements:

- IFS Customizable Web Server
  - IBM e-server pSeries or RS/6000 system per IFS System Requirements
- Mobile Network Server (for AvantGo)
  - Pentium class machine
  - 128 MB RAM
- Handheld device with modem (operating with PalmOS or WinCE as per software requirement)
  - Device examples are:

IBM WorkPad Companion or IBM WorkPad c3 Companion

3Com Palm Pilot (Palm III, Palm V or VII in wired modem mode)

HP Jornada 420 or 430SE Palm-sized PC or HP Jornada 820 or 690 Handheld PCs

### Software Requirements:

- IFS Customizable Web Server
  - AIX Version 4.3.3 or higher release
  - WebSphere Application Server 4.0, Advanced Edition for AIX
  - Java Development Kit for Java/2 Extended Edition (JDK) 1.3
  - MQSeries for AIX - Version 5.0.0.6 or higher release
  - IFS V1R6M1 Java Classes Package
  - IFS PvC Servlets
- Mobile Network Server
  - Windows NT 4.0 with Service Pack 4
  - AvantGo Enterprise Interactive, Version 3.1 or higher release
- Handheld Devices
  - Palm OS 2.04 or higher release with 320KB of free memory for the application plus additional space for data.
  - Microsoft Windows CE 2.0 or higher release with 460KB of free memory for the application and data.
  - AvantGo Enterprise Interactive, Version 3.1 or higher release

---

## IFS PvC Support Installation

### Configuration on CWS for IFS PvC Support

It is assumed that there is an existing IFS CWS already setup, configured and working. And SSL on the web server has been enabled.

## Steps for IFS PvC Support configuration on CWS:

1. Copy all the IFS PvC Support servlets files from cws/docs/servlets/pvc directory to the machine where Websphere is installed. Add the path of the servlets to the Application Server Classpath of Websphere.
2. Copy all the IFS PvC Support image files to the cws/docs/ifs.gif directory of the web server.
3. The IfsSystem class must be modified and recompiled. The changes are listed in Table 19. IfsSystem settings for PvC Servlets.
4. Restart the web server.
5. Attempt a logon. The logon screen URL will vary by installation, but should look similar to <http://machine:port/Java/IfsWpLogon>.

Table 19. IfsSystem settings for PvC Servlets

Setting	Value
IfsSystem.LogonPage	IfsWpLogon
IfsSystem.LogoffPage	IfsWpLogoff
IfsSystem.RedoPage	IfsWpRedoPage
IfsSystem.ErrorPage	IfsWpErrorPage
IfsSystem.GOLD_DATE_FORMAT	ISO
IfsSystem.DATE_HINT	yyyy-MM-dd

## Solution Design

The focus on the implementation of IFS PvC Servlets will be for WebSphere. All these servlets will interface with IFS Java Classes and create output in HTML format for the handheld device. The AvantGo client/server components are included in the following solution description, since the mobile network software used in the development and testing of the PvC Starter Kit was based on AvantGo. Mobile network packages from other vendors may also work but have not been tested.

## User Interface Design

The AvantGo client can run on both Palm OS and Windows CE devices. These devices come in a variety of shapes and sizes. Usually, the screen size of Palm OS device is 160 x 160, but screens on a windows CE device can range from 800 x 600 on the largest HPC Pro units down to 240 x 320 on the smallest palm-size PCs.

In order to accommodate the different screen sizes and shapes, the HTML pages were designed with special attention paid to the following issues:

- Limit page size to avoid too much vertical scrolling. Provide multi-page drill-down navigation instead.
- Keep the width of data within one screen length to avoid sometimes cumbersome horizontal scrolling.
- Limit the use of large images as they consume precious screen space.
- Provide a consistent and accessible navigation scheme throughout different pages.

Refer to the section ,IFS User Interface on Handheld Device- which shows the handheld device user interface designs provided with the PvC Starter Kit.

## IFS PvC Servlets Design

IFS PvC Servlets are a set of servlets that run on IFS Customizable Web Server. They are divided into several groups according to the functions they provided, including accounts inquiry, funds transfer and customer services. Usually, one function is supported by one or more servlets, and basically all servlets have the same flow chart as illustrated in Figure 18.

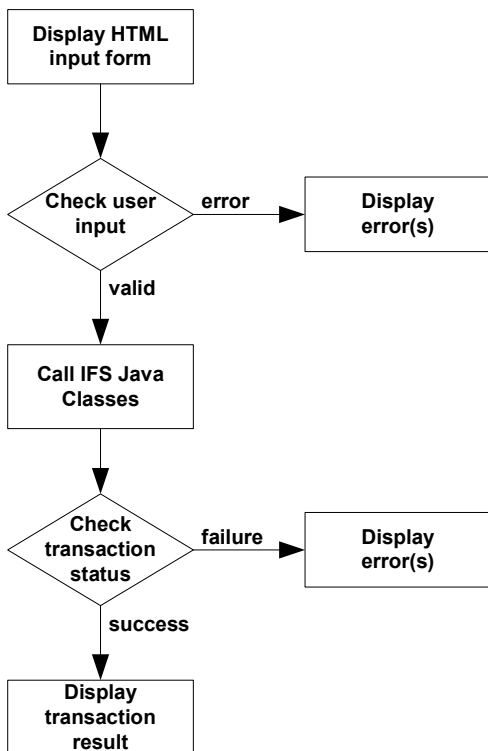


Figure 16. Flow chart of IFS PvC servlet

## Components of IFS PvC Servlets

IFS Java Classes package provides a base class of `IfsHttpServlet`. It defines a set of common service methods to handle common functions for servlets, such as:

- creating Gt\* and Gs\* objects,
- caching objects,
- setting request parameters,
- page navigation,
- error handling,
- sort, etc.

The IFS PvC Starter Kit includes 27 servlets. They are all subclasses of `IfsWpApplication` which in turn is a subclass of `IfsHttpServlet`. Figure 19 shows the inheritance hierarchy of IFS PvC servlets.

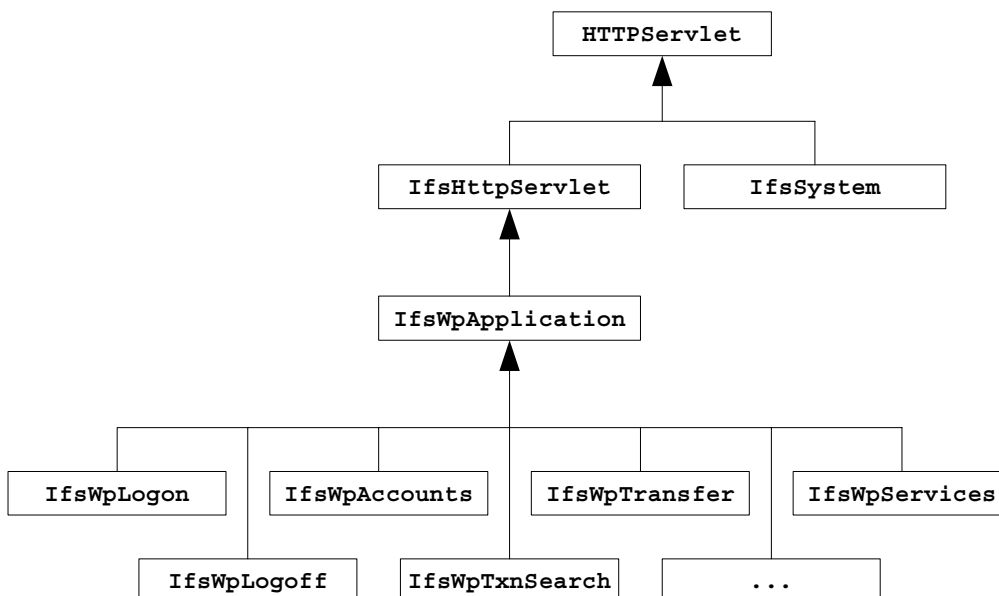


Figure 17. The Inheritance Hierarchy of IFS PvC Servlets

,Class Definition- gives more detailed description for each of these classes.

**Caching Objects:** During page navigation, objects need to be passed from one page to another. This is achieved by caching the objects and passing their identifiers used in the cache over HTTP requests. Base object Id (oid) is a string used as the key to store or retrieve an object to/from the session cache. All the cached objects are destroyed when the current user session ends.

**Session Management and Cookies:** HTTP session cookies are the glue for these servlets to run smoothly. The AvantGo application uses cookies for session management and preference settings just like any web application. The only difference in the handling of cookies is that all cookies in an AvantGo deployment are stored at the AdvantGo server. No cookies are stored on the AvantGo client.

**Error Handling:** IFS PvC Servlets will report error information to the user when there is an error executing a transaction. Following are the scenarios for error handling:

- Transaction execution resulted in some error in FI, Core Controller, or Gold Manager. This exception will be caught by PvC Servlets and reported to the user with an error message.
- Transaction execution was successful but resulted in non-zero business response code. Business response codes (BRC) are generally FI-specific and are defined in the message table that maps every possible BRC to descriptive message. This table is part of the CWS configuration files. There is a ,default- message table provided by IFS. Any IFS can customize it to define additional BRCs. When a BRC is not equal to 0, PvC Servlets will get the related descriptive message from the `ifsBrcErrorTextBundle` property file and report it to the user.

## Data Security Consideration

Data security is very important for the IFS solution that provides electronic banking services. The data exchanged between devices and IFS Web Server needs to be encrypted. Security between the AvantGo Server and the IFS web server is established via HTTP/SSL. AvantGo uses Certicom's secure technology for handheld devices based on its implementation of an advanced security technology called Elliptic Curve Cryptography (ECC).

## NLS Enabling

The HTML created by IFS PvC servlets contains static parts and dynamic parts, so the NLS enabling will be considered as follows.

**Static HTML Parts:** The static HTML parts created by servlets are not NLS enabled. Therefore, the IFS PvC servlets need to be customized to support different languages. This will result in multiple sets of servlets.

**Dynamic HTML Parts:** The IFS CWS Toolkit is NLS enabled for creating dynamic HTML visual components. At runtime, each dynamic visual component is rendered based on the current locale:

- The format of number, date, currency will be gotten from the customized `ifsSystem` class.
- `ifsHttpServlet` class provides utility functions using localized algorithms for search, compare, and concatenation.

---

## Class Definition

### Common Services

#### `ifsWpApplication` class

**public class `ifsWpApplication` extends `ifsHttpServlet`**

This is the super class for the IFS pervasive computing (PvC) application. It contains common methods for the application.

## Public Methods:

**public void callPage\_ErrorPage(HttpServletResponse response, Exception e, String message) throws IOException**

Passes the message to the error page. Servlet page flow continues with the error page.

**public void callPage\_NoSession(HttpServletResponse response) throws IOException**

Passes the no session message to the redo page. Servlet page flow continues with the redo page.

**public void callPage\_RedoPage(HttpServletResponse response, String nextPage, String redoMsg) throws IOException**

Passes the message to the redo page. Servlet page flow continues with the redo page.

**public void displayAccounts(GtUserProfile up, GsAccount account, PrintWriter out)**

Displays the list of available accounts within the user profile in a pull-down select list.

**public void displayAccountNum(GtUserProfile up, GsAccount account, PrintWriter out)**

Displays the list of available accounts within the user profile in a pull-down select list.

**public void displayCheckBookStyleList(String checkBookStyle, PrintWriter out)**

Displays the list of check book style options in a pull-down select list.

**public void displayFrequencySelectList(String frequencyType, PrintWriter out)**

Displays the list of frequency types in a pull-down select list.

**public void displayPostalSelectList(String postalCode, PrintWriter out)**

Displays the list of postal delivery options in a pull-down select list.

**public void displayStopPaymentList(String stopCode, PrintWriter out)**

Displays the list of stop check payment options in a pull-down select list.

**public void displayTransactionPeriodList(String transactionPeriod, PrintWriter out)**

Displays the list of transaction periods in a pull-down select list.

**public void displayTransactionSourceList(String transactionSource, PrintWriter out)**

Displays the list of transaction sources in a pull-down select list.

**public void displayTransactionTypeList(String transactionType, PrintWriter out)**

Displays the list of transaction types in a pull-down select list.

**public void endHtmlOutput(PrintWriter out)**

Puts page links to print writer and closes it.

**public String getAmountString(GsCurrency amt)**

Returns the currency amount in string format.

**public double getAndCheckAmount(GsCurrency amount)**

If the currency amount exists, it is checked for validity. A zero is returned when the input is null or an exception occurs.

**public String getDateString(GsDate date)**

Returns the date in string format.

**public String getTextString(int key, String resourceFile, IfsSession ifsSession, PrintWriter out) throws IfsException**

Returns the text corresponding to the integer ,key- in the properties ,resourceFile- file.

**public String getTextString(String key, String resourceFile, IfsSession ifsSession, PrintWriter out) throws IfsException**

Returns the text corresponding to the string ,key- in the properties ,resourceFile- file.

**public boolean isValidAmount(GsCurrency amount)**

Verifies that the currency amount exists and is valid.

**public boolean isValidDate(GsDate gsDate)**

Checks if the date is valid.

**public boolean isValidNumber(int intNumber)**

Checks if the integer number is greater than zero.

**public boolean isValidNumber(String strNumber)**

Checks if the string number is valid.

**public void show2DivBlocks(PrintWriter out, String strTitle, GsAccount acct)**

Display item title and it s account number.

**public void show2DivBlocks(PrintWriter out, String strTitle, GsCurrency amtValue)**

Display item title and it s currency value.

**public void show2DivBlocks(PrintWriter out, String strTitle, GsDate dateValue)**

Display item title and it s date value.

**public void show2DivBlocks(PrintWriter out, String strTitle, GsDecimal decValue)**

Display item title and it s decimal value.

**public PrintWriter startHtmlOutput(HttpServletResponse response, String title) throws IOException**

Setup for outputting HTML information.

## **IfsWpErrorPage class**

**public class IfsWpErrorPage extends IfsWpApplication**

This class provides a general error display. Whenever an error occurs, the page flow is redirected to this servlet.

Public Methods:

**public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**

Displays the information associated with the error. The following Http Request Parameters are posted to this page:

txn - Value posted from any page when there is an error executing the transaction.

Exception - Value posted from any page when an exception occurred.

**public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**

Displays the information associated with the error.

The same Http Request Parameters are posted to this page as for doGet method.

## **IfsWpRedoPage class**

**public class IfsWpRedoPage extends IfsWpApplication**

This class provides a general error display for REDO and INFO errors. Whenever a REDO or INFO error occurs, the page flow is redirected to this servlet.

Public Methods:

**public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**



Displays the information associated with the error. The following Http Request Parameters may be posted to this page:

- txn - Value posted from any page when there is an error executing the transaction.
- Exception - Value posted from any page when an exception occurred.
- redomsg - The REDO message which is to be displayed.
- nextpage - Controls page flow. It specifies the next page to be displayed.
- op - Search transfer operation (change or delete).
- transfer - Change transfer object identifier (oid) for GtTransferHistoryInquiry object.
- i - Change transfer index GtTransferHistoryInquiry object.

**public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**

Displays the information associated with the error. The same Http Request Parameters are posted to this page as for doGet method.

## User Authorization

### IfsWpLogon class

**public class IfsWpLogon extends IfsWpApplication**

This class displays the logon form for users to logon to the IFS system.

Public Methods:

**public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**

Displays the logon form for user to logon to the IFS system. No Http Request Parameters are expected to be posted to this page.

### IfsWpLogoff class

**public class IfsWpLogoff extends IfsWpApplication**

This class logoffs the user from the IFS system.

Public Methods:

**public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**

Displays the logoff confirmation page for the consumer. No Http Request Parameters are expected to be posted to this page.

**public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException**  
Logs the user off of the IFS system. No Http Request Parameters are expected to be posted to this page.

### IfsWpMain class

**public class IfsWpMain extends IfsWpApplication**

This class is used to logon to IFS system and display the main functions screen.

Public Methods:

**public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException**

Executes the logon transaction and displays the main function screen. The following Http Request Parameters are posted to this page by IfsWpLogon servlet:

userID - The ID of the user being logged on.

pin - The user s pin value.

password - The user s password.

## **IfsWpMainDisplay class**

**public class IfsWpMainDisplay extends IfsWpApplication**  
This class displays the main functions screen.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**  
Displays the main functions screen. No Http Request Parameters are expected to be posted to this page.

## **Account Inquiries**

### **IfsWpAccounts class**

**public class IfsWpAccounts extends IfsWpApplication**  
This class displays the menu items for the account functions.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**  
Displays the menu items for the account transactions. No Http Request Parameters are expected to be posted to this page.

### **IfsWpActSummary class**

**public class IfsWpActSummary extends IfsWpApplication**  
This class displays a summary of all accounts including deposit accounts, credit accounts and loan accounts.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**  
Displays a summary of all the accounts for the user including deposit accounts, credit accounts and loan accounts. No Http Request Parameters are expected to be posted to this page.

### **IfsWpActDetails class**

**public class IfsWpActDetails extends IfsWpApplication**  
This class displays the details of a specific account.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**  
Displays details of a specific account or a list of all of the user s accounts. The following Http Request Parameters are posted to this page:

ReqTag - Specifies whether a list of all accounts or the details for a specific account is to be displayed. It can have values of "AllAccounts" or "Details".

AccountIndex - When ReqTag equals "Details", it specifies the index of the account to be displayed.

## **IfsWpTxnHistory class**

### **public class IfsWpTxnHistory extends IfsWpApplication**

This class displays the posted transactions for a specific account by transaction date range selection.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a History transaction. No Http Request Parameters are expected.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a History transaction or executes it. The following Http Request Parameters are posted to this page:

ReqTag - Indicates whether to display confirmation or execute.

It can have the values of "DispFirst" or "DispMore".

transfer - Specifies the object identifier (oid) of a GtAccountHistory object.

req???? - The input values from the History Transaction form.

## **IfsWpTxnSearch class**

### **public class IfsWpTxnSearch extends IfsWpApplication**

This class displays the posted transactions for a specific account based on selection criteria inputted by the consumer, such as the transaction amount.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Search transaction. No Http Request Parameters are expected.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Search transaction or executes it. The following Http Request Parameters are posted to this page:

ReqTag - Indicates whether to display confirmation or execute.

It can have the values of "DispFirst" or "DispMore".

transfer - Specifies the object identifier (oid) of a GtAccountHistory object.

req???? - The input values from the Search Transaction form.

## **IfsWpBalanceInquiry class**

### **public class IfsWpBalanceInquiry extends IfsWpApplication**

This class is used for the consumer to request various balances associated with a specific consumer account.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result)** throws ServletException, IOException

Displays balance of a specific account or a list of all of the user s accounts. The following Http Request Parameters are posted to this page:

ReqTag - Specifies whether a list of all accounts or the balance for a specific account is to be displayed. It can have values of "AllAccounts" or "Details".

AccountIndex - When ReqTag equals "Details", it specifies the index of the account to be displayed.

## **IfsWpInterestRates class**

**public class IfsWpInterestRates extends IfsWpApplication**

This class displays a list of the current interest rates.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result)** throws ServletException, IOException

Gets and displays the interest rates. No Http Request Parameters are expected.

## **Fund Transfers**

### **IfsWpTransfer class**

**public class IfsWpTransfer extends IfsWpApplication**

This class displays the menu items for fund transfers.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result)** throws ServletException, IOException

Displays the menu items for the fund transfer s transactions. No Http Request Parameters are expected.

### **IfsWpAddFixedTransfer class**

**public class IfsWpAddFixedTransfer extends IfsWpApplication**

This class is used to handle the Add Fixed Transfer transaction.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Add Fixed Transfer transaction. No Http Request Parameters are expected.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Add Fixed Transfer transaction or executes it. The following Http Request Parameters are posted to this page:

whereFlag - Indicates whether to display confirmation or execute. It can have the values of "fromForm" or "fromConfirm".

transfer - Specifies the object identifier (oid) of a GtAddFixTransfer object.

req???? - The input values from the Add Fix Transfer form.

## IfsWpAddVariableTransfer class

**public class IfsWpAddVariableTransfer extends IfsWpApplication**

This class is used to handle the Add Variable Transfer transaction.

Public Methods:

```
public void doGet(HttpServletRequest request,  
    HttpServletResponse result) throws ServletException, IOException  
Displays the input form for generating a Add Variable Transfer  
transaction. No Http Request Parameters are expected.
```

```
public void doPost(HttpServletRequest request, HttpServletResponse result) throws  
ServletException, IOException
```

Either displays the confirmation page for a Add Variable Transfer transaction or executes it. The following Http Request Parameters are posted to this page:

```
whereFlag - Indicates whether to display confirmation or  
execute. It can have the values of "fromForm" or "fromConfirm".  
transfer - Specifies the object identifier (oid) of a  
GtAddVariableTransfer object.  
req???? - The input values from the Add Variable Transfer form.
```

## IfsWpSearchTransfer class

**public class IfsWpSearchTransfer extends IfsWpApplication**

This class handles the Search Transfer transaction and prepares the data for Change or Delete Transfer transactions.

Public Methods:

```
public void doGet(HttpServletRequest request, HttpServletResponse result) throws  
ServletException, IOException
```

Displays the input form page for a Search Transfer transaction. The following Http Request Parameter is posted to this page:

```
op - Indicates the transfer change type. It can have the values  
of "search" or "delete".
```

```
public void doPost(HttpServletRequest request, HttpServletResponse result) throws  
ServletException, IOException
```

Displays the results of a Search Transfer transaction. The following Http Request Parameters are posted to this page:

```
ReqTag - Indicates which group of results is to be displayed.  
It can have the values of "DispFirst" or "DispMore".  
transfer - Specifies the object identifier (oid) of a  
GtTransferHistoryInquiry object.  
req???? - The input values from the Search Transfer Transaction  
form.  
SourceAccount - The index of the selected account to be  
searched. It is posted when ReqTag=DispFirst.  
NUM - The starting index of the next transfer to display. It is  
posted when ReqTag=DispMore.  
LEFTNUM - The number of transfers left to display. It is posted  
when ReqTag=DispMore.
```

## IfsWpChangeFixedTransfer class

**public class IfsWpChangeFixedTransfer extends IfsWpApplication**

This class is used to handle the Change Fixed Transfer transaction.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Change Fix Transfer transaction. The following Http Request Parameters are posted to this page:

transfer - Specifies the object identifier (oid) of a

GtTransferHistoryInquiry object in the session cache.

i - The transfer history detail index.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Change Fix Transfer transaction or executes it. The following Http Request Parameters are posted to this page:

whereFlag - Indicates whether to display confirmation or execute. It can have the values of "fromForm" or "fromConfirm".

transfer - Specifies the object identifier (oid) of a GtChangeFixTransfer object.

req???? - The input values from the Change Fix Transfer form.

### **IfsWpChangeVariableTransfer class**

**public class IfsWpChangeVariableTransfer extends IfsWpApplication**

This class is used to handle the Change Variable Transfer transaction.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Change Variable Transfer transaction. The following Http Request Parameters are posted to this page:

transfer - Specifies the object identifier (oid) of a GtTransferHistoryInquiry object in the session cache.

i - The transfer history detail index.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Change Variable Transfer transaction or executes it. The following Http Request Parameters are posted to this page:

whereFlag - Indicates whether to display confirmation or execute. It can have the values of "fromForm" or "fromConfirm".

transfer - Specifies the object identifier (oid) of a GtChangeVariableTransfer object.

req???? - The input values from the Change Variable Transfer form.

### **IfsWpDeleteFixedTransfer class**

**public class IfsWpDeleteFixedTransfer extends IfsWpApplication**

This class handles the transaction for Delete Fixed Transfer.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Delete Fix Transfer transaction. The following Http Request Parameters are posted to this page:

- transfer - Specifies the object identifier (oid) of a GtTransferHistoryInquiry object in the session cache.
- i - The transfer history detail index.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Delete Fix Transfer transaction or executes it. The following Http Request Parameters are posted to this page:

- transfer - Specifies the object identifier (oid) of a GtDeleteFixTransfer object.
- req???? - The input values from the Delete Fix Transfer form.

### **IfsWpDeleteVariableTransfer class**

**public class IfsWpDeleteVariableTransfer extends IfsWpApplication**  
This class handles the transaction for Delete Variable Transfer.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Delete Variable Transfer transaction. The following Http Request Parameters are posted to this page:

- transfer - Specifies the object identifier (oid) of a GtTransferHistoryInquiry object in the session cache.
- i - The transfer history detail index.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Delete Variable Transfer transaction or executes it. The following Http Request Parameters are posted to this page:

- transfer - Specifies the object identifier (oid) of a GtDeleteVariableTransfer object.
- req???? - The input values from the Delete Variable Transfer form.

### **IfsWpTransferInquiry class**

**public class IfsWpTransferInquiry extends IfsWpApplication**

This class handles the transaction for Transfer Inquiry which allows user to request information about Transfers.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form page for a Transfer Inquiry transaction. No Http Request Parameters are expected to be posted to this page.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the results of a Transfer Inquiry transaction. The following Http Request Parameters are posted to this page:

ReqTag - Indicates which group of results is to be displayed.

It can have the values of "DispFirst" or "DispMore".

transfer - Specifies the object identifier (oid) of a GtTransferHistoryInquiry object.

req???? - The input values from the Transfer Inquiry Transaction form.

SourceAccount - The selected account to be searched. It is posted when ReqTag=DispFirst.

NUM - The starting index of the next transfer to display. It is posted when ReqTag=DispMore.

LEFTNUM - The number of transfers left to display. It is posted when ReqTag=DispMore.

## Customer Services

### IfsWpServices class

**public class IfsWpServices extends IfsWpApplication**

This class displays the menu items for customer services.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the menu items for the customer service s transactions. No Http Request Parameters are expected to be posted to this page.

### IfsWpCheckCopy class

**public class IfsWpCheckCopy extends IfsWpApplication**

This class is used to handle the Check Copy Order transaction.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for user to generate a Check Copy transaction. No Http Request Parameters are expected to be posted to this page.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Check Copy transaction or executes it. The following Http Request Parameters are posted to this page:

ReqTag - Indicates whether to display confirmation or execute.

It can have the values of "Confirm" or "Result".

transfer - Specifies the object identifier (oid) of a GtCheckCopy object.

req???? - The input values from the Check Copy form.



## IfsWpCheckBook class

**public class IfsWpCheckBook extends IfsWpApplication**

This class is used to handle the Check Book Reorder transaction.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for user to generate a Order Check Book transaction. No Http Request Parameters are expected to be posted to this page.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Order Check Book transaction or executes it. The following Http Request Parameters are posted to this page:

ReqTag - Indicates whether to display confirmation or execute.  
It can have the values of "Confirm" or "Result".  
transfer - Specifies the object identifier (oid) of a  
GtOrderCheckBook object.  
req???? - The input values from the Order Check Book form.

## IfsWpStopPayment class

**public class IfsWpStopPayment extends IfsWpApplication**

This class is used to handle the Stop Check Payment transaction.

Public Methods:

**public void doGet(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Displays the input form for generating a Stop Check Payment transaction. No Http Request Parameters are expected to be posted to this page.

**public void doPost(HttpServletRequest request, HttpServletResponse result) throws ServletException, IOException**

Either displays the confirmation page for a Stop Check Payment transaction or executes it. The following Http Request Parameters are posted to this page:

ReqTag - Indicates whether to display confirmation or execute.  
It can have the values of "Confirm" or "Result".  
transfer - Specifies the object identifier (oid) of a  
GtStopCheckPayment object.  
req???? - The input values from the Stop Check Payment form.

---

## IFS PvC Servlets Normal Page Flow

The servlets page flow starts with the logon page. All subsequent page navigation depends on what transactions are selected to be executed (servlet\_name - method\_name).

For example, the page flow to logon, order a check copy, and logoff would be:

IfsWpLogon doGet

- > IfsWpMain doPost
  - > IfsWpServices doGet
  - > IfsWpCheckCopy doGet
  - > IfsWpCheckCopy doPost

->IfsWpLogoff doGet  
-> IfsWpLogoff doPost

Following is the normal flow for the transactions starting at the logon page:

### **IfsWpLogon**

IfsWpLogon doGet  
-> IfsWpMain doPost  
-> IfsWpAccounts doGet  
-> IfsWpTransfer doGet  
-> IfsWpServices doGet

### **IfsWpAccounts**

IfsWpAccounts doGet  
-> IfsWpActSummary doGet  
-> IfsWpActDetails doGet ?ReqTag=AllAccounts  
-> IfsWpBalanceInquiry doGet ?ReqTag=AllAccounts  
-> IfsWpInterestRates doGet  
-> IfsWpTxnSearch doGet  
-> IfsWpTxnHistory doGet

### **IfsWpTransfer**

IfsWpTransfer doGet  
-> IfsWpAddFixedTransfer doGet  
-> IfsWpAddVariableTransfer doGet  
-> IfsWpChangeTransfer doGet ?op=change  
-> IfsWpChangeTransfer doGet ?op=delete  
-> IfsWpTransferInquiry doGet

### **IfsWpServices**

IfsWpServices doGet  
-> IfsWpCheckCopy doGet  
-> IfsWpCheckBook doGet  
-> IfsWpStopPayment doGet

### **IfsActSummary**

IfsWpActSummary doGet  
-> IfsWpActDetails doGet ?AccountIndex=xx&ReqTag=Details

### **IfsWpActDetails**

IfsWpActDetails doGet ?ReqTag=AllAccounts  
-> IfsWpActDetails doGet ?AccountIndex=xx&ReqTag=Details

### **IfsWpBalanceInquiry**

IfsWpBalanceInquiry doGet ?ReqTag=AllAccounts  
-> IfsWpActDetails doGet ?AccountIndex=xx&ReqTag=Details

### **IfsWpInterestRates**

IfsWpInterestRates doGet

### **IfsWpTxnSearch**

IfsWpTxnSearch doGet  
-> IfsWpTxnSearch doPost (1 or more pages of transactions)

### **IfsWpTxnHistory**

IfsWpTxnHistory doGet  
-> IfsWpTxnHistory doPost (1 or more pages of transactions)

### **IfsWpAddFixedTransfer**

IfsWpAddFixedTransfer doGet  
-> IfsWpAddFixedTransfer doPost

**IfsWpAddVariableTransfer**

IfsWpAddVariableTransfer doGet  
-> IfsWpAddVariableTransfer doPost

**IfsWpChangeTransfer**

IfsWpChangeTransfer doGet ?op=change  
-> IfsWpSearchTransfer doGet (one or more pages of transfers)  
-> IfsWpChangeVariableTransfer doGet  
-> IfsWpChangeFixedTransfer doGet

**IfsWpChangeTransfer—doGet ?op=delete**

-> IfsWpSearchTransfer doGet (one or more pages of transfers)  
-> IfsWpDeleteVariableTransfer doGet  
-> IfsWpDeleteFixedTransfer doGet

**IfsWpTransferInquiry**

IfsWpTransferInquiry doGet  
-> IfsWpTransferInquiry doPost (one or more pages of transfers)

**IfsWpChangeVariableTransfer**

IfsWpChangeVariableTransfer doGet  
-> IfsWpChangeVariableTransfer doPost

**IfsWpChangeFixedTransfer**

IfsWpChangeFixedTransfer doGet  
-> IfsWpChangeFixedTransfer doPost

**IfsWpDeleteVariableTransfer**

IfsWpDeleteVariableTransfer doGet  
-> IfsWpDeleteVariableTransfer doPost

**IfsWpCheckCopy**

IfsWpCheckCopy doGet  
-> IfsWpCheckCopy doPost

**IfsWpCheckBook**

IfsWpCheckBook doGet  
-> IfsWpCheckBook doPost

**IfsWpStopPayment**

IfsWpStopPayment doGet  
-> IfsWpStopPayment doPost

---

## IFS User Interface on Handheld Device

### Logon Screen

To enter the IFS system, users must input their customer ID, PIN and password. Click the Logon button, the system will carry out the Logon transaction. Only the valid user can enter the system. See Figure 20.



Figure 18. Logon Screen

## Main Function Screen

The PvC Starter Kit provides three functions for the PDA: Accounts, Transfers and Services. See Figure 21. Click any of the function icons to access the corresponding screen.



Figure 19. Main Function Screen

## Accounts Function Screen

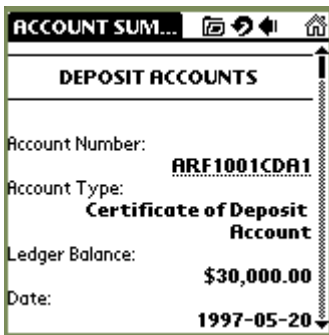
Provides six functions: Account Summary, Account Details, Transaction Search, Transaction History, Balance Inquiry and Interest Rates. See Figure 22. Click any of the function items, to access the corresponding screen.



Figure 20. Accounts Function Screen

## Account Summary Screen

Provides the customer's account summary information. Click this function item to display the account type, account number, account name, balance and date information of all accounts. See Figure 23.



**ACCOUNT SUM...**

**DEPOSIT ACCOUNTS**

Account Number: **ARF1001CDA1**

Account Type: **Certificate of Deposit Account**

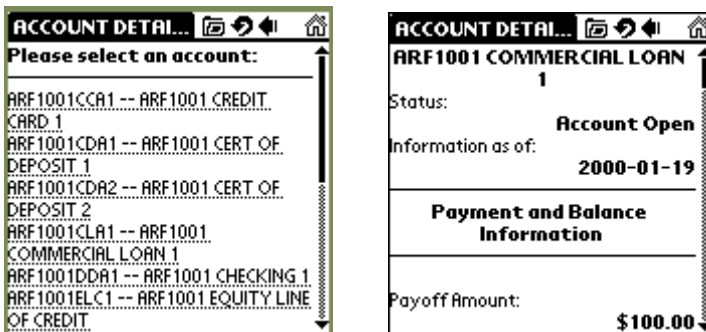
Ledger Balance: **\$30,000.00**

Date: **1997-05-20**

Figure 21. Account Summary Screen

## Account Details Screen

Provides account details of one specific consumer account. Click this function item to display all accounts belonging to the consumer. Click any of the accounts to display account details for that account. See Figure 24.



**ACCOUNT DETAL...**

Please select an account:

- ARF1001CCA1 -- ARF1001 CREDIT CARD 1
- ARF1001CDA1 -- ARF1001 CERT OF DEPOSIT 1
- ARF1001CDA2 -- ARF1001 CERT OF DEPOSIT 2
- ARF1001CLA1 -- ARF1001 COMMERCIAL LOAN 1
- ARF1001DDA1 -- ARF1001 CHECKING 1
- ARF1001ELC1 -- ARF1001 EQUITY LINE OF CREDIT

**ACCOUNT DETAL...**

**ARF1001 COMMERCIAL LOAN 1**

Status: **Account Open**

Information as of: **2000-01-19**

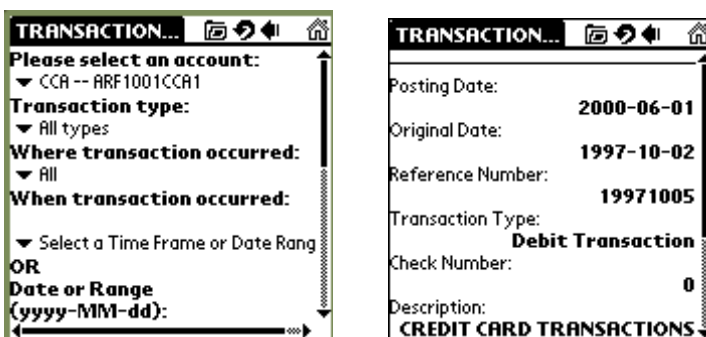
**Payment and Balance Information**

Payoff Amount: **\$100.00**

Figure 22. Account Details Screen

## Transaction Search Screen

Click transaction search item to display the user input screen. When the user completes the data input and clicks the ,Enter- button, it submits the request and shows the search result. See Figure 25.



**TRANSACTION...**

Please select an account:

▼ CCA -- ARF1001CCA1

Transaction type:

▼ All types

Where transaction occurred:

▼ All

When transaction occurred:

▼ Select a Time Frame or Date Range

OR

Date or Range (yyyy-MM-dd):

←-----→

**TRANSACTION...**

Posting Date: **2000-06-01**

Original Date: **1997-10-02**

Reference Number: **19971005**

Transaction Type: **Debit Transaction**

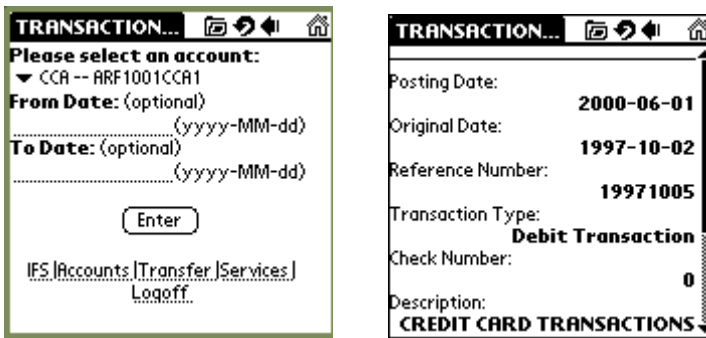
Check Number: **0**

Description: **CREDIT CARD TRANSACTIONS**

Figure 23. Transaction Search Screen

## Transaction History Screen

Click transaction history item to display the user input screen. When the user completes the data input and clicks the ,Enter- button, it submits the request and shows the transaction history. See Figure 26.



**TRANSACTION...**

Please select an account:  
 ▼ CCA -- ARF1001CCA1  
**From Date:** (optional)  
 (yyyy-MM-dd)  
**To Date:** (optional)  
 (yyyy-MM-dd)  
 Enter

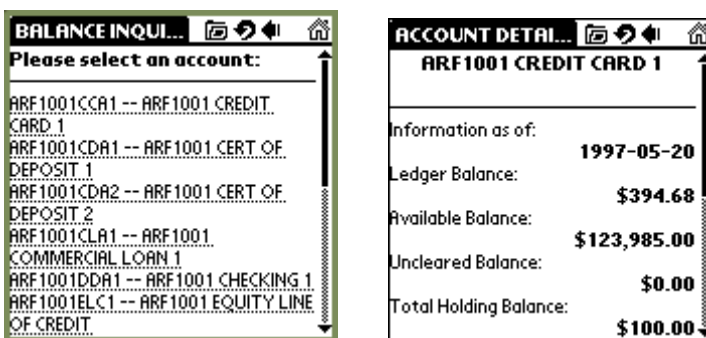
IFS |Accounts|Transfer|Services|  
 Logoff

Posting Date: 2000-06-01  
 Original Date: 1997-10-02  
 Reference Number: 19971005  
 Transaction Type: Debit Transaction  
 Check Number: 0  
 Description: CREDIT CARD TRANSACTIONS

Figure 24. Transaction History Screen

## Balance Inquiry Screen

Click balance inquiry item to display all accounts belonging to the consumer. Click any of the accounts to display balances of the account. See Figure 27.



**BALANCE INQUI...**

Please select an account:  
 ARF1001CCA1 -- ARF1001 CREDIT CARD 1  
 ARF1001CDA1 -- ARF1001 CERT OF DEPOSIT 1  
 ARF1001CDA2 -- ARF1001 CERT OF DEPOSIT 2  
 ARF1001CLA1 -- ARF1001 COMMERCIAL LOAN 1  
 ARF1001DDA1 -- ARF1001 CHECKING 1  
 ARF1001ELC1 -- ARF1001 EQUITY LINE OF CREDIT

**ACCOUNT DETAL...**

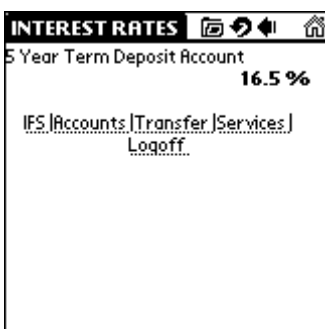
ARF1001 CREDIT CARD 1

Information as of: 1997-05-20  
 Ledger Balance: \$394.68  
 Available Balance: \$123,985.00  
 Uncleared Balance: \$0.00  
 Total Holding Balance: \$100.00

Figure 25. Balance Inquiry Screen

## Interest Rates Screen

Click interest rates item to display a list of current interest rates. See Figure 28.



**INTEREST RATES**

5 Year Term Deposit Account  
 16.5 %

IFS |Accounts|Transfer|Services|  
 Logoff

Figure 26. Interest Rates Screen

## Fund Transfer Function Screen

The Transfer item provides five functions: Add Fixed Transfer, Add Variable Transfer, Change Transfers, Delete Transfers and Transfer History Inquiry. See Figure 29. Click any of the function items to access the corresponding screen.



Figure 27. Fund Transfer Screen

## Add Fixed Transfer Function Screen

Click Add Fixed Transfer item to display the user input screen. When the user completes the data input and clicks the ,Enter- button, it displays the confirmation screen. Click ,OK- button to submit the request and show the result. See Figure 30.

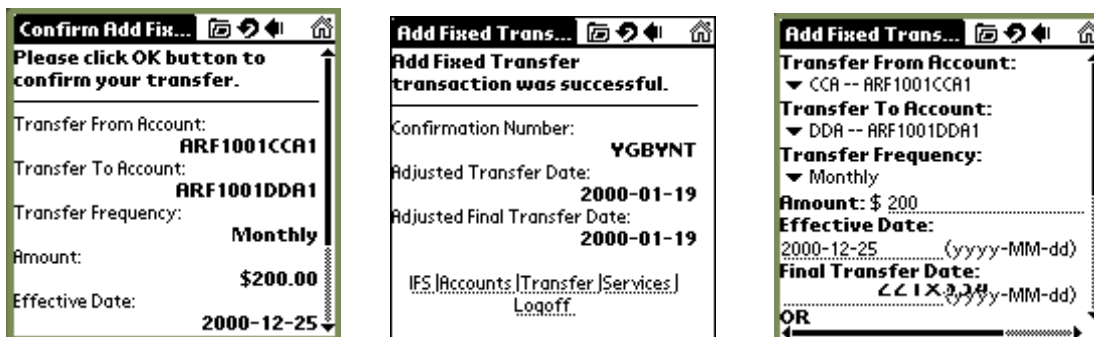


Figure 28. Add Fixed Transfer Screen

## Add Variable Transfer Function Screen

Click Add Variable Transfer item to display the user input screen. When the user completes the data input and clicks the ,Enter- button, it will display the confirmation screen. Click ,OK- button to submit the request and show the result. See Figure 31.



Figure 29. Add Variable Transfer Screen

## Change Transfer Function Screen

Click Change Transfer item to display Search Transfer input screen. When the user completes the data input and clicks the ,Search- button, it displays the Search results screen. See Figure 32.

Click on the Transfer Number to display Change Fixed Transfer input screen or Change Variable Transfer input screen. When the user completes the data input and clicks the ,Change- button, it displays the

confirmation screen. Click ,OK- button to submit the request and show the result. See Figure 33 and Figure 34.

Figure 30. Search Transfer Screen

Figure 31. Change Fixed Transfer Screen

Figure 32. Change Variable Transfer Screen

## Delete Transfer Function Screen

Click Delete Transfer item to display Search Transfer input screen. When the user completes the data input and clicks the ,Search- button, it displays the Search results screen. Click the Transfer Number to display Delete Fixed Transfer screen or Delete Variable Transfer screen. When the user clicks the ,OK- button, it submits the request and shows the results. See Figure 35.



Figure 33. Delete Fixed Transfer Screen

Confirm Delete...	
Do you really want to delete the following Transfer?	
Transfer From Account:	ARF1001CCA1
Transfer To Account:	ARF1001CDA1
Transfer Frequency:	Weekly
Amount:	\$5.00
Effective Date:	2005-05-05

Delete Variable...	
Delete variable transfer was successful	
Confirmation Reference Number:	000000000309
Service Provider Reference:	000000000309
IFS  Accounts Transfer Services	
Logoff	

Figure 34. Delete Variable Transfer Screen

## Transfer Inquiry Function Screen

Click Transfer Inquiry item to display the user input screen. When the user completes the data input and clicks the ,Search- button, it submits the request and shows the results. See Figure 36.

Transfer Inquiry	
Please select an account:	
▼ All Funding Accounts	
Date Range: (Optional)	
From:	
To:	yyyy-MM-dd
Amount Range: (Optional)	
From: \$	
To: \$	
Transfer Type and Status:	
▼ Scheduled - All	
Search	

Transfer Inquiry	
Type:	Single Transfers
Source:	ARF1001CCA1
Target:	ARF1001SDA1
Amount:	\$6.00
Effective Date:	1999-12-31
Type:	

Figure 35. Transfer Inquiry Screen

## Services Function Screen

The Services item provides three functions: Check Copy Order, Check Book Reorder and Stop Cheque Payment. See Figure 37. Click any of the function items to access the corresponding screen.

Services
► Check copy order
► Check book reorder
► Stop cheque payment
IFS  Accounts Transfer Services
Logoff

Figure 36. Service Function Screen

## Check Copy Order

Click check copy order item to display the user input screen. When the user completes the data input and clicks the ,Enter- button, it displays the confirmation screen. Click ,OK- button to submit the request and show the result. See Figure 38.

**Check Copy Ord...**

Select an Account:  
▼ CCA -- ARF1001CCA1

Check Number:  
1005

Amount: \$  
1500

Posting Date: (optional)  
2000-9-12

Reference Number: (optional)

Check Copy Delivery Method:

**Confirm Request**

Your check copy request will be submitted. To confirm, click on **OK**, otherwise, click on **Cancel**.

**Note: Your check copy will be mailed to the address on the account selected within 10 working days. There will be a \$3.00 fee charged to your account for each check copy.**

Account:  
ARF1001CCA1

**Check Copy Req...**

Your check copy has been requested successfully!

FI Reference Number:  
000000000311

Service Provider Reference Id:  
000000000311

IFS |Accounts |Transfer |Services |  
Logoff

Figure 37. Check Copy Order Screen

## Check Book Reorder

Click check book reorder item to display the user input screen. When the user completes the data input and clicks the ,Enter- button, it displays the confirmation screen. Click ,OK- button to submit the request and show the result. See Figure 39.

**Check Book Reo...**

Select an Account:  
▼ ILA -- ARF1001ILA1

Select Check Book Style:  
▼ Style B

Number of Checks: 100

Check Book Delivery Method:  
▼ Postal Delivery

Consumer Nickname:  
A10

Consumer Account Name:

**Confirm Request**

Your check book reorder will be submitted. To confirm, click on **OK**, otherwise, click on **Cancel**.

**Note: Your check book will be mailed to the address on the account selected within 10 working days. There will be a \$5.00 fee charged to your account.**

Account:  
ARF1001ILA1

**Check Book Req...**

Your check book has been requested successfully!

FI Reference Number:  
000000000313

Service Provider Reference Id:  
000000000313

IFS |Accounts |Transfer |Services |  
Logoff

Figure 38. Check Book Reorder Screen

## Stop Cheque Payment

Click stop cheque payment item to display the user input screen. When user completes the data input and clicks the ,Enter- button, it displays the confirmation screen. Click ,OK- button to submit the request and show the result. See Figure 40.

**Stop Check Pay...**

Select an Account:  
▼ MLA -- ARF1001MLA1

Select Stop Payment Type:  
▼ by Check Number and Amount

Check Number or Range:  
From: 1005  
To:

Amount or Range:  
From: \$ 1500  
To: \$

Posting Date or Range:  
From:

**Confirm Request**

Your check stop payment will be submitted. To confirm, click on **OK**, otherwise, click on **Cancel**.

**Note: There will be a \$5.00 fee charged to your account for each stop payment.**

Account:  
ARF1001MLA1

Stop Payment Type:  
by Check Number and Amount

Check Number:

**Stop Payment...**

Your stop check payment has been requested successfully!

FI Reference Number:  
000000000314

Service Provider Reference Id:  
000000000314

IFS |Accounts |Transfer |Services |  
Logoff

Figure 39. Stop Cheque Payment Screen

## Logoff Screen

Logoff from the IFS system. The first screen displays a confirmation screen. Tap the **Logoff** button to submit the request and logoff the system.

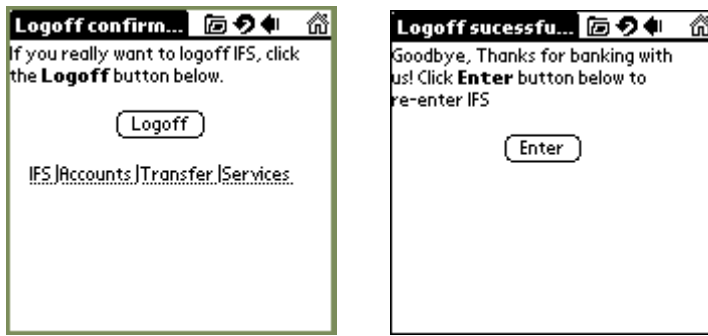


Figure 40. Logoff Screen

---

## Appendix F. Basic WebSphere 4 Administration

---

### WebSphere 4.0.2 Overview

This appendix describes some of the basic WebSphere administrative tasks that have been mentioned in this guide. The descriptions and instructions provided are not intended to be exhaustive, but should instead be used as quick references to help novice users get started with WebSphere administration and to find out where they can obtain more information. The following topics will be discussed:

- Starting WebSphere
- Launching a GUI administration client
- Stopping WebSphere using the GUI client
- Starting an application server using the GUI client
- Stopping an application server using the GUI client
- The wscp.sh command line utility

More information on these, and many more, topics can be found in the WebSphere 4.0.2 InfoCenter documentation that comes bundled with the product or on-line at <http://www-3.ibm.com/software/webservers/appserv/infocenter.html>. Four other good sources of information include the WebSphere 4.0.2 release notes found at <http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/was/relnotesindx.html>, the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*, the *IBM RedBook-RedPiece SG24-6520-00 WebSphere Advanced Edition: Security* and *IBM RedBook SG24-6134-00 WebSphere Version 4 Application Development Handbook*.

---

### Starting WebSphere Administration Server

Before an application server can be started or the graphical or command line administration clients used, the administration server must be started. To start the WebSphere 4.0.2 administration server follow the instructions below:

1. Log on to the system as user `root`.
2. Change into the `/usr/WebSphere/AppServer/bin` directory  
`cd /usr/WebSphere/AppServer/bin`
3. Execute the `startupserver.sh` script in the background  
`./startupserver.sh &`
4. Observe progress by tailing the generated tracefile  
`tail -f /usr/WebSphere/AppServer/logs/tracefile`
5. The server has been successfully started when you see the message  
`Server __adminServer open for e-business`

---

### Launching the Graphical Administrative Client

All WebSphere administrative tasks that one might like to perform can be accomplished through the use of the graphical client or the `wscp.sh` command line interface. The graphical client tends to be more intuitive and user friendly, while the command line tool is great for automating tasks through scripting.

Both administrative clients use Remote Method Invocation technology to communicate with the administration server. This allows the administrative client to run on a separate machine than the administrative server providing a great deal of flexibility. The graphical client is a Java Swing application and can be processor and memory intensive. To increase the responsiveness of the client it is recommended that you install and run it on a separate machine running AIX or Windows.

Launching the administrative client is the same whether you are administering a local or remote WebSphere node and can be accomplished using the following instructions:

#### AIX/Linux Instructions

1. Log onto the system as user `root`.
2. Change into the bin directory off of the WebSphere installation root.  
  
`# cd /usr/WebSphere/AppServer/bin`
3. Run the adminclient command specifying the administrative server hostname and port.  
  
`# ./adminclient.sh test.ibm.com 900`

## Windows Instructions

1. Log onto the system.
2. Change into the bin directory off of the WebSphere installation root.  
  
`C:\> cd WebSphere\AppServer\bin`
3. Run the adminclient command passing the administrative server hostname and port.  
  
`C:\WebSphere\AppServer\bin> adminclient test.ibm.com 900`

The specified hostname must be the fully qualified hostname of the system running the administrative server. Port 900 is the default administrative server port.

---

## Stopping WebSphere

Stopping WebSphere entails stopping all application servers as well as the administration server as instructed below:

1. Launch a graphical administration client using the instructions above.
2. In the left pane there will be an icon labeled *WebSphere Administrative Domain* left click on the plus sign to the left of it.
3. In the left pane left click on the plus sign to the left of the *Nodes* icon.
4. In the left pane, immediately below the *Nodes* icon, there should be an item with the same name as the hostname of the system running the administration server, left click on the name to highlight it.
5. Left click on the Stop button (circle with an x in the center) in the toolbar.
6. A dialog box will open to confirm your choice, click the *Yes* button.

Please note that as a result of clicking the *Yes* button in step 6 your graphical administrative client will exit.

---

## Starting an Application Server

In WebSphere 4.0.2 all EJBs (Enterprise Java Beans), Java Servlets, JSPs and other web resources are contained inside of an enterprise application which runs inside of an application server. The application server is actually a JVM (Java virtual machine) that provides an environment for, as well as, interprets and runs Java components. You will not be able to run CWS without first starting the application server it was

deployed into. CWS comes with tools to start and stop the application server (`ifs.war/tools/startweb` and `stopweb` scripts discussed in Chapter 3. Installing CWS Components) but follow the instructions below if you wish to start the server manually:

1. Launch a graphical administration client using the instructions above.
2. In the left pane there will be an icon labeled *WebSphere Administrative Domain*” left click on the plus sign to the left of it.
3. In the left pane, left click on the plus sign next to the *Nodes* icon.
4. In the left pane, immediately below the *Nodes* icon, there should be an icon labeled with the hostname of the system containing the application server you wish to start, left click on the plus sign next to it.
5. In the left pane there will be an icon labeled *Application Servers*, left click the plus sign next to it.
6. Immediately underneath the Application Servers icon there will be a list of the application servers that have been created. To the left of each name there will be a red icon if the server is stopped or a green icon if the server is started. Left click on the name of the server you wish to start; it should become highlighted.
7. Left click on the Start button (circle with an arrow in the center) in the toolbar.

A dialog box will open to notify you when the server has been started or of any errors that were encountered. Click the *OK* button.

---

## Stopping an Application Server

Follow the instructions below to stop an application server using the administration GUI client.

1. Launch a graphical administration client using the instructions above.
2. In the left pane there will be an icon labeled *WebSphere Administrative Domain* left click on the plus sign next to it.
3. In the left pane, left click on the plus sign next to the *Nodes* icon.
4. In the left pane, immediately below the *Nodes* icon, there should be an icon labeled with the hostname of the system containing the application server you wish to stop, left click on the plus sign next to it.
5. In the left pane there will be an icon labeled *Application Servers*, left click the plus sign next to it.
6. Immediately underneath the Application Servers icon there will be a list of the application servers that have been created. To the left of each name there will be a red icon if the server is stopped or a green icon if the server is started. Left click on the name of the server you wish to stop; it should become highlighted.
7. Left click on the Stop button (circle with an x in the center) in the toolbar.
8. A dialog box will open to notify you when the server has been stopped or of any errors that were encountered. Click the *OK* button.

---

## The wscp.sh Command Line Utility

The `wscp.sh` (WebSphere Control Program) is a command-line administrative tool for WebSphere 4.0.2 Advanced Edition. All administrative tasks that can be performed with the graphical administration client can also be performed with the `wscp.sh` command line utility.

The `wscp.sh` command is based on the scripting language Tcl. The `wscp.sh` launches a Tcl interpreter which parses the command line arguments and executes the specified Tcl statements.. Below is an example of how to use the `wscp.sh` command to start an application server named *testServer* on the host *test1.ibm.com*.

## AIX

```
# cd /usr/WebSphere/AppServer/bin
# ./bin/wscp.sh -c
"ApplicationServer start
/Node:test1/ApplicationServer:testServer/"
```

The `ifs.war/tools/startweb` and `stopweb` scripts distributed with CWS use the `wscp.sh` command to obtain status information about and to start and stop the CWS application server. For more information on the `wscp.sh` command and Tcl syntax please see chapter 23 of the *IBM RedBook-RedPiece SG24-6176-00 IBM WebSphere V4.0 Advanced Edition Handbook*.

---

## Appendix G. Migrating from Java 1.1.8 to Java 1.3.0

---

### Java API Changes

Much change has occurred in the Java APIs supported by WebSphere from version 2.0.3 to version 4.0.2. Examples of such changes include the removal of support for some classes, changes in the behavior of methods, deprecation of methods, creation of new methods and the creation of entire new APIs. In addition to the Java API changes some of the classes and methods in the JCWSAPI have also been altered to function properly in the WebSphere 4.0.2 environment. The following sections discuss the APIs that are supported in WebSphere 4.0.2, where you can obtain additional information on said APIs, the changes made to the JCWSAPI and potential problems and resolution you may experience when migrating your Java code from WebSphere 2.0.3 to WebSphere 4.0.2.

### WebSphere 4.0.2 Supported APIs

#### J2EE Components

API	Supported Level	More Information
Servlet	2.2	<a href="http://java.sun.com/products/servlet/">http://java.sun.com/products/servlet/</a>
JSP	1.1	<a href="http://java.sun.com/products/jsp/">http://java.sun.com/products/jsp/</a>
EJB	1.1	<a href="http://java.sun.com/products/ejb/">http://java.sun.com/products/ejb/</a>

#### J2EE Services

API	Supported Level	More Information
JDBC	2.0	<a href="http://java.sun.com/products/jdbc/">http://java.sun.com/products/jdbc/</a>
JTA/JTS	1.1	<a href="http://java.sun.com/products/jta/">http://java.sun.com/products/jta/</a> <a href="http://java.sun.com/products/jts/">http://java.sun.com/products/jts/</a>
JNDI	1.2.1	<a href="http://java.sun.com/products/jndi/">http://java.sun.com/products/jndi/</a>
JAF	1.0	<a href="http://java.sun.com/products/javabeans/glasgow/jaf.html">http://java.sun.com/products/javabeans/glasgow/jaf.html</a>
XML4J	3.1.1	<a href="http://www.alphaworks.ibm.com/tech/xml4j">http://www.alphaworks.ibm.com/tech/xml4j</a>
XSL	2.0	<a href="http://www.w3.org/Style/XSL/">http://www.w3.org/Style/XSL/</a>

#### J2EE Communication

API	Supported Level	More Information
RMI-IIOP	1.0	<a href="http://java.sun.com/products/rmi-iiop/">http://java.sun.com/products/rmi-iiop/</a>
JMS	1.0.1	<a href="http://java.sun.com/products/jms">http://java.sun.com/products/jms</a>
Java Mail	1.1	<a href="http://java.sun.com/products/javamail/">http://java.sun.com/products/javamail/</a>

### Changes in the JCWSAPI

Changes in the behavior and levels of the Java APIs supported by WebSphere 4.0.2 have made some changes to the JCWSAPI a necessity. What follows is a discussion of the changes that were made, the rationale for making the changes and examples of how to update your code to make use of the changes.

#### JDBC Changes

There have been many changes in the way WebSphere handles database connection pooling from versions 2.0.3 to 4.0.2. In version 2.0.3 classes like `com.ibm.Servlet.connmgr.IBMConnMgr` and `com.ibm.Servlet.IBMJdbcConnSpec` were used inside of the `com.ibm.ifs.gold.IfsJDBC` class to create and manage connection pools. Furthermore, to create the connections in the pool we were required to specify connection information such as username, password, JDBC driver and JDBC URL. This system has been completely changed in WebSphere 4.0.2, now, connection pooling is handled by WebSphere itself.

To use connection pooling in WebSphere 4.0.2 you must first create a JDBC provider and data source using



either the WebSphere administrative GUI client or the wscp.sh command line interface. A JDBC provider is equivalent to the JDBC driver from the version 2.0.3 implementation and a data source contains all of the required connection information. The data source has a JNDI (Java Naming and Directory Interface) name and is stored by a naming service within WebSphere. Client code can retrieve a connection from the pool by performing a lookup of said data source using its unique JNDI name. Of course, this functionality is encapsulated in the `com.ibm.ifs.gold.IfsJDBC` class which is the only JCWSAPI class that had any changes made to it as a result of the new connection pooling implementation.

The `com.ibm.ifs.gold.IfsJDBC` class had several obsolete methods removed and new methods added to accommodate for the new connection pooling style. As a result, code that used the old `IfsJDBC` class will now fail to compile and run. What follows is a list of the differences in the public API of the `IfsJDBC` class and how you must change your existing code so it will compile and run with the new implementation.

#### `com.ibm.ifs.gold.IfsJDBC` - Methods That Have Been Removed

- 1) `public void setUserID(String id)`
- 2) `public String getUserID()`
- 3) `public void setPassword(String password)`
- 4) `public String getPassword()`
- 5) `public void setPoolName(String name)`
- 6) `public String getPoolName()`
- 7) `public void setDriverName(String driver)`
- 8) `public String getDriverName()`
- 9) `public Enumeration getDrivers()`
- 10) `public void setUrl(String url)`
- 11) `public String getUrl()`

#### `com.ibm.ifs.gold.IfsJDBC` - Methods That Have Been Added

- 1) `public void setDataSourceName(String newName) throws IfsException`
- 2) `public String getDataSourceName()`

#### Updating Code That Uses `IfsJDBC`

A necessary first step that must be performed prior to updating any code is to create the JDBC provider and data sources that will manage our database connections. Instructions on how to create these objects can be found in the WebSphere InfoCenter documentation that comes bundled with the product or can be found online at <http://www-3.ibm.com/software/webervers/appserv/doc/v40/ae/infocenter/>

The change that most classes will have to make is to replace the calls to the `setDriverName()`, `setUrl()`, `setPassword()` and `setUserID()` methods with a call to the `setDataSourceName()` method. For example, code that looks like this:

```
.
.
.
    try {
IfsWebTransactionActivity webTxn =
(IfsWebTransactionActivity)createInstance(ifsSession,
"com.ibm.ifs.gold.IfsWebTransactionActivity", "webTxn")
setFields(ifsSession, "webTxn", webTxn);
webTxn.setDriverName("COM.ibm.db2.jdbc.app.DB2Driver");
webTxn.setUrl("jdbc:db2:DB71");
webTxn.setUserID("webuser1");
webTxn.setPassword("password");
.
.
.
```

Would be replaced with the following code:

```
.  
.   
.   
try {  
    IfsWebTransactionActivity webTxn =  
        (IfsWebTransactionActivity)createInstance(ifsSession,  
        "com.ibm.ifs.gold.IfWebTransactionActivity", "webTxn")  
    setFields(ifsSession, "webTxn", webTxn);  
    webTxn.setDataSourceName("dataSource1");  
.  
.  
.
```

Assuming that the data source named *dataSource1* had been previously created using the values above.

A second necessary change is to remove calls to the methods that have been deleted from the JCWSAPI. The method calls that have to be removed are listed in the section above entitled *com.ibm.ifs.gold.IfJDBC - Methods That Have Been Removed* and the exact line numbers in the source file where the method calls appear can be obtained from the Java compiler by compiling it against the new IfsJDBC class. Instructions on how to compile Java code in the WebSphere 4 environment can be found in chapter 1.

### **IfsHttpServlet.addCopyright( HttpServletResponse )**

The addCopyright() method attempts to write to a *java.io.PrintWriter* Object obtained from the passed in *javax.servlet.http.HttpServletResponse* parameter. This situation can lead to problems which are further explained in problem 2 of the *Common Migration Problems and Resolutions* section further below.

To correct the situation the old *IfsHttpServlet.addCopyright()* method has been deprecated and replaced by two methods with the following signatures:

- 1) public static String getCopyright()
- 2) public void addCopyright(HttpServletRequest)

The first variation returns the copyright information as a String Object and is the recommended solution. The second variation will place the copyright information inside of the *javax.servlet.http.HttpServletRequest* parameter with the name *COPYRIGHT* and can later be retrieved with a call to the *javax.servlet.http.HttpServletRequest.getAttribute(String name)* method.

It is recommended that the generation of the HTML be inside of a JSP or a static HTML file. A servlet should only be used to generate binary data, for exception handling and for data gathering. This is called the MVC or Model View Controller paradigm and is used in the following two examples.

#### Example 1: IfsHttpServlet.getCopyright() method

The *IfsHttpServlet.getCopyright()* method returns the copyright information in the form of a String and can be directly inserted into JSP code as follows:

```
<%@ page import="com.ibm.ifs.servlets.IfHttpServlet" %>  
  
<%= IfHttpServlet.getCopyright() %>  
  
<HTML>  
  
<HEAD>  
  
<TITLE>  
Example use of IfHttpServlet.getCopyright()  
</TITLE>  
</HEAD>  
<BODY>
```

```
<P>
This is a test of the IfsHttpServlet.getCopyright() method. Please      view
the HTML source to see if the copyright information was included.
</BODY>
</HTML>
```

### Example 2: IfsHttpServlet.addCopyright(HttpServletRequest) method

The proper use IfsHttpServlet.addCopyright(HttpServletRequest) method can only be illustrated with both a Java Servlet and JSP. The following example is also a good illustration of how data can be obtained by a Servlet and passed to a JSP for HTML generation (MVC paradigm).

#### **Test.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.ifs.IfHttpServlet;

public class TestServlet extends IfHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        addCopyright(req);
        req.getRequestDispatcher("/Test.jsp").forward(req, res);
    }
}
```

#### **Test.jsp**

```
<%@ page import="com.ibm.ifs.gold.IfsSystem" %>
<%= request.getAttribute(IfsSystem.COPYRIGHT_PROP_NAME) %>
<HTML>
<HEAD>
<TITLE>
Example use of IfHttpServlet.getCopyright()
</TITLE>
</HEAD>
<BODY>
<P>
This is a test of the IfHttpServlet.getCopyright() method. Please      view
the HTML source to see if the copyright information was included.
</BODY>
</HTML>
```

As a further enhancement the copyright text is now obtained from the *IfSystem.properties* file as the value of the *ifs.system.copyright.info* property. This allows for the actual copyright string to be changed without having to recompile any Java code.

In general, it should be noted that all information written to the output stream should come from the same file, be it a Servlet or JSP. That is, if you try to write to the output stream then forward to a secondary resource you will encounter problems in the WebSphere 4.0.2 environment.

### **IfsHttpServlet.handleErrors(IfsTxn, String, String, String)**

The com.ibm.ifs.IfHttpServlet.handleErrors() method will make a call to the javax.servlet.http.HttpServletResponse.sendRedirect() method if a problem was encountered during the processing of the specified transaction. This will lead to problems and/or unexpected behavior if the calling Servlet also makes a call to javax.servlet.http.HttpServletResponse.sendRedirect(). This problem is further explained in problem 3 of the Common Migration Problems and Resolutions section found below. To correct the problem, the handleErrors() method has been updated to return a boolean; true if there were no errors (no call to sendRedirect() was made) or false if there were errors (a call to sendRedirect() was made). Code that uses the handleErrors() method inside of a Servlet's doGet() or doPost() that used to look like the following:

```

.
.
handleErrors(webTxn, "errorPage", "infoPage", "redoPage");

response.sendRedirect("successPage");
.
.
.

```

Must be changed to:

```

.
.
.
// if there were no errors redirect to the successPage else, handleErrors
// will have already requested a redirect to an error page
if (handleErrors(webTxn,"errorPage","infoPage","redoPage")) {
    response.sendRedirect("successPage");
}
.
.
.

```

## **lfsHttpServlet.handleReauthentication(HttpServletRequest, HttpServletResponse)**

Problems similar to the `com.ibm.ifs.servlets.lfsHttpServlet.handleErrors()` method are also encountered when using the `com.ibm.ifs.servlets.lfsHttpServlet.handleReauthentication()` method. The method has been updated to return a boolean; true if no errors were encountered (no redirect was made) and false if there were errors (a redirect was made). That is, if this method returns false then the calling Servlet should not make any calls to `javax.servlet.http.HttpServletResponse.sendRedirect()` or `javax.servlet.http.HttpServletRequest.getRequestDispatcher().forward()`. For example, code that may have looked like this:

```

.
.
.
handleReauthentication(request, response);

response.sendRedirect("successPage");
.
.
.

```

Must be converted to the following:

```

.
.
.
if (handleReauthentication(request, response)) {
    response.sendRedirect("successPage");
}
.
.
.

```

## **Other Class Changes**

### GsDecimalFormatSymbols

This class has been removed, use the `DecimalFormatSymbols` from `java.text` package instead.

### GsDecimalFormat

The constructor:

```
public GsDecimalFormat(String pattern, GsDecimalFormatSymbols symbols)
```

has changed to:

```
public GsDecimalFormat(String pattern, DecimalFormatSymbols symbols).
```

The `getFormatSymbols()` method has been removed. Use `getDecimalFormatSymbols()` instead.

The `setFormatSymbols(GsDecimalFormatSymbols symbols)` method has been removed. Use `setDecimalFormatSymbols()` instead.

## GsCurrencyFormat

If both `applyPattern` and `setDecimalFormatSymbols` are both being called, be sure to call `setDecimalFormatSymbols` before `applyPattern`.

The constructor:

```
public GsCurrencyFormat(String pattern, GsDecimalFormatSymbols symbols)
```

has changed to:

```
public GsCurrencyFormat(String pattern, DecimalFormatSymbols symbols).
```

---

## Common Migration Problems and Resolutions

### The Invoker Servlet

The invoker Servlet is an internal mechanism used by WebSphere to enable the serving of user created Servlets by their classname. The structure of a URL used by WebSphere 4.0.2 to load a Servlet has the following form:

*http://<host>[:<port>]<context-root><servlet-mapping>*

An example of this might be

*http://test.ibm.com:8080/testapp/testServlet*

Where `test.ibm.com` is the host listening on port `8080`, `testapp` is the context root for a defined web application and `testServlet` is a Uniform Resource Identifier (URI) that is mapped to an instance of the `javax.Servlet.HttpServlet` class. In this example, the actual name of the class file executed is hidden. WebSphere keeps an internal mapping of URIs to class names and when a request is received it uses this mapping to resolve the URI to a Java class file. For instance, `testServlet` may map to an instance of `com.ibm.test.TestHttpServlet` and that is the code that will be executed when the above URL is entered into a web browser.

If instead, you want to be able execute a Servlet by its class name you must use the special invoker Servlet provided by WebSphere. In WebSphere 2.0.3, the URI of the invoker Servlet was configurable by editing the `<WAS2_ROOT>/properties/server/Servlet/adminservice/rules.properties` file. By default, a URI of `/servlet` is used, but can be changed to anything, for example, previous versions of the CWS User's Guide instructed the user to change the URI to `/java`. In WebSphere 4.0.2 the invoker URI is no longer configurable and is set to `/servlet`. Therefore, if the invoker Servlet was used to locate and execute Servlets in your WebSphere 2.0.3 environment and `/servlet` was not used as the invoker URI then some changes will be required before old code will work properly. Returning to the example above, if you wanted to use the invoker Servlet to execute the `com.ibm.test.TestHttpServlet` code in WebSphere 4.0.2, the correct URL would be:

*http://test.ibm.com:8080/testapp/servlet/com.ibm.test.TestHttpServlet*

## Solutions:

Below are two possible solutions that can be used independently of each other, or in combination, to get older code working in the WebSphere 4.0.2 environment.

### Solution 1

The first solution is to replace all references to the old invoker URI with the new URI. For example, if you used absolute URLs in your static HTML or JSP files then you could make the following substitution:

```
<A href="/testapp/java/TestHttpServlet">Link</A>
```

with

```
<A href="/testapp/servlet/TestHttpServlet">Link</A>
```

This may be accomplished in an automated fashion with a search and replace tool. The downside to this solution is that you will have to update all files that make such references and then test the files to ensure that they work properly.

### Solution 2

The second solution is to create Servlet mappings for all code that is loaded by the invoker Servlet. You can add Servlet mappings using the Application Assembly Tool (AAT) or by manually editing the web.xml file located in the WEB-INF subdirectory of the ifs.war file (see the WebSphere InfoCenter documentation for more information about creating Servlet mappings). Here s an excerpt from a web.xml file that maps the URI */java/com.ibm.test.TestHttpServlet* to the Servlet *com.ibm.test.TestHttpServlet*.

```
<Servlet id="Servlet_1">
<servlet-name>Test Servlet</servlet-name>
<display-name>Simple Test Servlet</display-name>
<description>Outputs Environment Variables</description>
<servlet-class>com.ibm.test.TestHttpServlet</servlet-class>
</Servlet>
... // Other Servlet Definitions
<servlet-mapping id="ServletMapping_1">
<servlet-name>Test Servlet</servlet-name>
<url-pattern>/java/com.ibm.test.TestHttpServlet</url-pattern>
</servlet-mapping>
... // Other Servlet Mappings
```

## Cannot Forward After Obtaining Stream

When writing a Java Servlet, WebSphere 2.0.3 allowed for data to be written to the `java.io.PrintWriter` Object, obtained from the `javax.servlet.http.HttpServletResponse` Object, and then forward control to another resource via the `javax.servlet.http.HttpServletResponse.sendRedirect()` or `javax.servlet.http.HttpServletRequest.getRequestDispatcher.forward()` methods. For example, the deprecated `addCopyright(HttpServletResponse)` method in the `com.ibm.ifs.Servlets.IfsHttpServlet` class obtains the `java.io.PrintWriter` Object from the passed in `javax.servlet.http.HttpServletResponse` parameter and writes a copyright information String to writer. Then control is passed back to the calling Servlet which can then forward control to another Servlet or JSP for HTML generation. This situation worked fine in WebSphere 2.0.3 but in the WebSphere 4.0.2 environment will produce a runtime Exception Error 500: Cannot forward. Writer or Stream already obtained.

A second example is illustrated by the following Java Servlet code:

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
import com.ibm.ifs.Servlets.IfsHttpServlet;
```

```

public class TestHttpServlet extends IfsHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        out.println("<!--");
        out.println("Copyright notice");
        out.println("-->");
        req.getRequestDispatcher( "/test.jsp" ).forward( request, response );
    }
}

```

The problem is that the Servlet wrote to the output stream before it forwarded control to the JSP. A slight variation of this would be to call a method, passing the *HttpServletResponse* Object as a parameter, that obtained the *PrintWriter* Object and wrote to the stream. This is not allowed in WebSphere 4.0.2 and will result in the *Error 500: Cannot forward. Writer or Stream already obtained* message being returned.

## Solution:

The solution is to avoid obtaining and writing to the Servlet's *PrintWriter* if said Servlet forwards to another resource. The code that obtains and writes to the *PrintWriter* will have to be moved from the Servlet and into the resource that the Servlet forwards to. This can be accomplished by placing the data inside of the *HttpServletRequest* Object through the use of the *setAttribute(String, Object)* method and retrieving it with the *HttpServletRequest.getAttribute(String)* method.

## Example: WebSphere 2.0.3 Implementation

### **Test.java**

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.ifs.servlets.*;

public class Test extends IfsHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException
    {
        PrintWriter out = res.getWriter();
        out.println("Some header information");
        ((com.sun.server.http.HttpServiceResponse) res).callPage("Test.jsp",
        req);
    }
}

```

### **Test.jsp**

```

<HTML>
<HEAD>
<TITLE>
Test JSP
</TITLE>
</HEAD>
<BODY>
<P>
This is a simple JSP to test if information can be written to the
PrintStream.
</BODY>
</HTML>

```

## Example: WebSphere 4.0.2 Implementation

### **Test.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.ifs.servlets.*;

public class Test extends IfsHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException
    {
        req.setAttribute("HEADER_INFO", "Some header information");
        req.getRequestDispatcher("/Test.jsp").forward(req, res);
    }
}
```

### **Test.jsp**

```
<%= request.getAttribute("HEADER_INFO") %>
<HTML>
<HEAD>
<TITLE>
Test JSP
</TITLE>
</HEAD>
<BODY>
<P>
This is a simple JSP to test if information can be written to the
PrintStream.
</BODY>
</HTML>
```

## **Effective Call To sendRedirect() Method**

The effective call to the *javax.servlet.http.HttpServletResponse.sendRedirect()* method for WebSphere 4.0.2 are in reverse of that for WebSphere 2.0.3. If two or more calls to the *sendRedirect()* method are made in a Java Servlet or JSP in WebSphere 2.0.3 then the first call was the effective call, but in WebSphere 4.0.2 it is the last call. That is, in WebSphere 2.0.3 control would redirect to the resource specified in the first call to *sendRedirect()*, but in WebSphere 4.0.2 control would pass to the resource specified in the second call to *sendRedirect()*. This scenario can be better explained through examples.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.ifs.servlets.*;

public class Test extends IfsHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.sendRedirect("/test1.jsp");
        res.sendRedirect("/test2.jsp");
    }
}
```

If the above code was executed in the WebSphere 2.0.3 environment, then control would eventually be directed to test1.jsp, however, in WebSphere 4.0.2 control would pass to test2.jsp.

A second example uses the *com.ibm.ifs.Servlets.IbmHttpServlet.handleErrors(IbsTxn, String, String, String)* method.

```
import java.io.*;
import javax.servlet.*;
```



```

import javax.servlet.http.*;
import com.ibm.ifs.servlets.*;

public class Test extends IfsHttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        // Create and execute IfsTxn Object
        handleErrors(IfsTxn, "errorPage", "infoPage", "redoPage");

        res.sendRedirect("successPage");
    }
}

```

The *handleErrors()* method will examine the *IfsTxn* Object to determine if there were any problems processing the transaction. If there were problems, a call is made to the *sendRedirect()* method redirecting to either the *errorPage*, *infoPage* or *redoPage*, depending on the type of problem detected. Control is then passed back to the above code which in turn makes a call to the *sendRedirect()* method. Let us assume that there was a problem with the transaction and that *handleErrors()* made a call to the *sendRedirect()* method redirecting to the *redoPage*, control then passes back to the *doGet()* method above and a second call is made to *sendRedirect()* redirecting to the *successPage*. In the WebSphere 2.0.3 environment the first call is effective and everything would work fine, that is, control would be passed to the *redoPage* as it should since an error was detected. However, in WebSphere 4.0.2 the last call to *sendRedirect()* is effective so control would erroneously be passed to the *successPage*.

In summary, only one call to either *sendRedirect()* or *forward()* should be made in a Servlet's *doGet()* or *doPost()* method and it should be the last line of code to be executed. Furthermore, if a method is called that could possibly make a call to *sendRedirect()* or *forward()* there must be a mechanism in place to make the calling code aware so that the calling code does not make a second call that would effectively cancel out the first.

## JSP .91 Syntax not supported

### Problem:

The JSP specification has gone through a number of revisions since the release of WebSphere 2.0.3. WebSphere 2.0.3 supports the JSP .91 specification, WebSphere 3.x supports the .91, 1.0 and 1.1 specification but the support for .91 and 1.0 has been dropped in WebSphere 4.0.2. JSPs must be updated to remove elements that have no longer exist in the 1.1 specification and fortunately, the necessary updates are rather straight forward.

### Solutions:

The solution is to remove the use of the JSP .91 tags (specifically the <SERVLET>, <BEAN>, <REPEAT> and <INSERT> tags) with their 1.1 equivalent. The following tables are slightly modified versions from the WebSphere InfoCenter and include tips for migrating from the JSP .91 specification to 1.1. For more information about WebSphere 4.0.2 support for JSPs please see the InfoCenter documentation.

#### Replacing <SERVLET> with <jsp: include> directive

Use the JSP 1.1 equivalent of <SERVLET> to include data in a JSP page from another file.

JSP .91	<SERVLET CODE= WebSphereSamples.Counter.CounterServlet> </SERVLET>
JSP 1.1	<jsp: include page= /Servlet/WebSphereSamples.Counter.CounterServlet />
Discussion	In both cases the output of the

	WebSphereSamples.Counter.CounterServlet class will be inserted into the current Servlet.
--	--

### Replacing <BEAN> with <jsp:useBean>

Use the JSP 1.1 equivalent of <BEAN> to make an existing or newly created bean available from within the JSP file. Four variations are possible.

Variation 1: JSP is to create the bean

JSP .91	<pre>&lt;BEAN NAME= getQuestionDBBean TYPE= WebSphereSamples.Poll.GetQuestionDBBean CREATE= YES SCOPE= request &gt; &lt;/BEAN&gt;</pre>
JSP 1.1	<pre>&lt;jsp:useBean id= getQuestionDBBean type= WebSphereSamples.Poll.GetQuestionDBBean class= WebSphereSamples.Poll.GetQuestionDBBean scope= request /&gt;</pre>
Discussion	Both examples make available a bean named getQuestionDBBean to the JSP.

### *Variation 2: JSP is to use existing bean*

JSP .91	<pre>&lt;BEAN NAME= pollQueryDBBean TYPE= WebSphereSamples.Poll.PollQueryDBBean CREATE= NO INTROSPECT= NO SCOPE= request &gt; &lt;/BEAN&gt;</pre>
JSP 1.1	<pre>&lt;jsp:useBean id= pollQueryDBBean type= WebSphereSamples.Poll.PollQueryDBBean scope= request /&gt;</pre>
Discussion	Both versions allow you to use an existing bean named pollQueryDBBean

### *Variation 3: Properties are to be set for a bean*

JSP .91	<pre>&lt;BEAN NAME= getQuestionDBBean TYPE= WebSphereSamples.YourCo.Poll.GetQuestionDBBean CREATE= YES INTROSPECT= NO SCOPE= request &lt;PARAM NAME= userID VALUE= wsdemo &gt; &lt;/BEAN&gt;</pre>
---------	--

JSP 1.1	<pre> &lt;jsp:useBean id= getQuestionDBBean type= WebSphereSamples.YourCo.Poll.GetQuestionD BBean class= WebSphereSamples.YourCo.Poll.GetQuestionD BBean scope= request /&gt; &lt;jsp: setProperty name= getQuestionDBBean property= userID value= wsdemo /&gt; </pre>
Discussion	Both versions set the userID property of the getQuestionDBBean Object to wsdemo.

#### Variation 4: Invoke methods on a bean

JSP .91	<pre> &lt;% try { java.lang.String _p0_1 = feedbackQuery.getWSDemo_FEEDBACK_ NAME(0); }%&gt; </pre>
JSP 1.1	No change from JSP .91 to 1.1
Discussion	The NAME attribute in the <BEAN> tag and the id attribute in the <jsp:useBean> tag are equivalent. Both identify a bean named feedbackQuery. For either JSP specification, invoking a method on a bean is identical.

#### Replacing <REPEAT> with <tsx:repeat>

<tsx:repeat> provides for repeating information and is useful in the creation of HTML tables. <REPEAT> from JSP .91 is usually used with the <INSERT> tag to actually insert data from a specified bean. <REPEAT> from JSP .91 does not have an equivalent in the JSP 1.1 specification, however, IBM extension <tsx:repeat> provides much of the same functionality. The following example shows how to replace both the <REPEAT> and <INSERT> tags.

JSP .91	<pre> &lt;REPEAT INDEX=i&gt; &lt;%timeoutBean.getBalance(i);%&gt; &lt;TD&gt;&lt;INSERT BEAN=)timeoutBean) PROPERTY=)balance)&gt;&lt;/INSERT&gt;&lt;/TD&gt; &lt;/REPEAT&gt; </pre>
JSP 1.1	<pre> &lt;tsx:repeat index= i &gt; &lt;TD&gt; &lt;%= timeoutBean.getBalance(i) %&gt; &lt;/TD&gt; &lt;/tsx:repeat&gt; </pre>
Discussion	Note that there is an actual call, using Java syntax, of the getBalance method of the timeoutBean within the loop of <tsx:repeat>. This was done rather than using the IBM extension <tsx:getProperty> because the getBalance method requires an explicit argument.

# Unsupported API Calls

## Problem:

Since the release of WebSphere 2.0.3 there has been numerous changes to the Servlet specification and Java library classes. The WebSphere 2.0.3 environment uses the Servlet 2.0 and JDK 1.1.8 APIs which have been upgraded to Servlet 2.2 and JDK 1.3 APIs in WebSphere 4.0.2. This change results in the fact that some classes and methods that were available in the WebSphere 2.0.3 environment are no longer present in WebSphere 4.0.2. Runtime exceptions could be encountered if old class files are executed or compilation failure if Java files are recompiled in the WebSphere 4.0.2 environment.

## Solution:

The solution is to replace the WebSphere 2.0.3 code with the WebSphere 4.0.2 equivalent. What follows is a list of the problems that may be encountered when transitioning code from WebSphere 2.0.3 to 4.0.2.

### Problem 1

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.ifs.servlets.*;

public class Test extends IfsHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ((com.sun.server.http.HttpServiceRequest)
            req).setAttribute("attrName", object);
    }
}
```

The *com.sun.server.http.HttpServiceRequest* class is no longer available in the WebSphere 4.0.2 environment. The recommended fix is to replace the method call

```
((com.sun.server.http.HttpServiceRequest) req).setAttribute("attrName",
object);
```

with the following:

```
req.setAttribute("attrName", object);
```

### Problem 2

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.ifs.servlets.*;

public class Test extends IfsHttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ((com.sun.server.http.HttpServiceResponse) res).callPage("Test.jsp", req);
    }
}
```

The *com.sun.server.http.HttpServiceResponse* class is no longer available in the WebSphere 4.0.2 environment. The recommended fix is to replace the method call

```
((com.sun.server.http.HttpServiceResponse) res).callPage("Test.jsp", req);
```

with the following:

```
req.getRequestDispatcher("Test.jsp").forward(req, res)
```

## Recompilation and Deprecated Methods

### Problem:

Although Java byte code generated by JDK1.1.8 compilers for use in a WebSphere 2.0.3 environment, is loadable by the 1.3 JVM used by WebSphere 4.0.2, differences in the APIs that the code uses exist. If a Java class file that was generated in a WebSphere 2.0.3 environment was loaded and executed in the WebSphere 4.0.2 environment there is a chance that runtime errors and exception will be encountered if the code used classes and/or methods that are no longer available or supported.

### Solution:

The solution is to recompile all existing Java files using a JDK1.3 compliant compiler and the most recent version of the JCWSAPI and WebSphere 4.0.2 library files. To be specific, you must compile Java code with the j2ee.jar file included with WebSphere 4.0.2 as opposed to the servlet.jar file that came with WebSphere 2.0.3 in your classpath. For more information on building Java code for V1R6M1 - WebSphere 4.0.2, please see *Appendix H: Building And Deploying Java Code In WebSphere 4.0.2*.

Replacement of deprecated methods and classes is not mandatory but is highly recommended. If a class or method is deprecated it is currently supported but support will be removed in future Java releases. A list of the deprecated classes and methods used by a Java class can be obtained by using the *-deprecation* flag with the Java compiler. The following is a list of deprecated classes and methods, and their equivalent replacements, that may be encountered when recompiling WebSphere 2.0.3 Java code in the WebSphere 4.0.2 environment:

#### Deprecated Classes

Class	Replacement
java.security.Certificate	java.security.cert.Certificate

#### Deprecated Methods

Method	Replacement
javax.Servlet.http.HttpSession.putValue(String, Object)	javax.Servlet.http.HttpSession.setAttribute(String, Object)
javax.Servlet.http.HttpSession.getValue(String)	javax.Servlet.http.HttpSession.getAttribute(String)
com.ibm.ifs.gold.GtLogon.setLanguageCode(String)	Comment out or remove
coim.ibm.ifs.gold.GtLogon.setCountryCode(String)	Comment out or remove

## Miscellaneous Problems

Compiling code that uses the com.ibm.ifs.gold.IfsTxn.execute() method may result in the following problem being reported:

```
unreported exception com.ibm.ifs.gold.IfsException; must be caught or declared to be thrown
```

If this happens you must place the call inside of a try/catch block. For example the following code:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    .
    .
    .
}
```

```

        txn.execute();
    .
    .
    .
}

```

Would be changed to:

```

public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    .
    .
    .
    try {
        txn.execute();
    }
    catch (IfsException e) {
        // Exception handling
    }
    .
    .
    .
}

```

---

## Appendix H. Building And Deploying CWS Web Applications In WebSphere 4

Building for and deploying applications on WebSphere 4.0.2 is considerably different than on WebSphere 2.0.3. The output of a build process is a Web Archive (WAR) file that contains all of the resources for a web application to function properly in the WebSphere 4.0.2 environment (more information about WAR files can be found below). WAR files are deployed (installed) into WebSphere 4.0.2, configured and then executed inside of a web container. A web container is a Java Virtual Machine and a set of APIs (Application Program Interfaces) that together provide an environment for the web application's Java code to run in. Web containers are part of the J2EE (Java 2 Enterprise Edition) specification that WebSphere 4.0.2 fully supports.

Below is a brief introduction to the J2EE environment and terminology, followed by a discussion of the base `ifs.war` file that is distributed with CWS, how the base `ifs.war` can be updated to include custom CWS applications, deployment and redeployment of the `ifs.war` file and information on how to compile Java code in the WebSphere 4.0.2 environment.

---

### J2EE Overview

WebSphere 4.0.2 is a fully compliant J2EE 1.2 environment, as a result, it differs from WebSphere 2.0.3 in many fundamental ways. One of the major distinctions is how applications are packaged and deployed. What follows is a brief description of the new J2EE packaging and deployment concepts and terms, how these new facilities are used to distribute the CWS portion of IFS and instructions on how the provided `ifs.war` file can be extended to include your custom web applications.

### J2EE Packaging and Deployment Concepts and Terms

Deployment of web and enterprise applications in the WebSphere 4.0.2 environment is more robust and flexible than in WebSphere 2.0.3, but also more complicated. No longer can you install stand alone servlets or Java Server Pages (JSPs), everything must belong to an *enterprise application*. Enterprise applications are distributed in Enterprise Archive (EAR) files and contain Web Archive (WAR) files, Enterprise Java Beans (EJBs), configuration information and all of the resources that the application needs to run properly. Enterprise applications are installed into an *application server* and configured to listen on a specified *virtual host*. New resources can be added to an application after deployment but it is important to note that the resource has to be added to an application, it cannot exist independently. EAR files and WAR files are the only two types of resources that can be deployed on WebSphere 4.0.2.

More details about the terms discussed in the previous paragraph are provided below.  
[http://java.sun.com/j2ee/tutorial/1\\_3-fcs/index.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html).

### Web Archive (WAR) Files

When a web client, such as a browser, communicates with a J2EE application, it does so through server-side web components. There are two types of web components Java Servlets and JSPs. Servlets are Java programming language classes that dynamically process requests and construct responses. JSP pages are text-based documents that are compiled into, and execute as, Servlets but allow for a more natural approach to creating static content. While Servlets and JSP pages can be used interchangeably, each has its strengths. Servlets are best suited to managing the control function of an application, such as creating and executing transactions, retrieving information from a database, performing data manipulation, error checking and handling, dispatching, and generation of binary content. JSPs are more appropriate for generating text-based markup such as HTML or XML. The two components work very nicely together to separate data and the view of said data. Servlets should be used to obtain or manipulate data which is then dispatched to JSPs for view generation (usually HTML based).

Web clients are packaged in WAR files. In addition to web components, a WAR file usually contains other resources including:

Server-side utility classes (database beans, shopping carts, and so on). Often, these classes conform to the JavaBeans component architecture.

Static web content (HTML, images, sound files, etc.)

- Dynamic content (Flash modules, client-side javascript source files, etc.)
- Client-side classes (applets and utility classes).
- Java property files
- Any other files that the web application needs to run such as Java Native Interface (JNI) libraries, tools, utilities, etc.

A WAR file has a specific directory structure. The top-level directory of a WAR file is considered to be the *document root* of the application. The document root is where JSPs, client-side classes and archives, and static web resources are stored.

The document root contains a subdirectory called WEB-INF, which contains the following files and directories.

- **web.xml** - the deployment descriptor for the web application. This file contains configuration information used by WebSphere including descriptions and Uniform Resource Identifiers (URI) for web components amongst other things.
- **ibm-web-bnd.xml** - a WebSphere proprietary file that describes JNDI (Java Naming and Directory Interface) binding information. (Note: this is currently not used by CWS).
- **ibm-web-ext.xml** - a WebSphere proprietary file that contains application server configuration information including Servlet reloading, Servlet reload interval and serving Servlets by classname.
- Tag library descriptor files for JSP tag libraries.
- **classes** - a directory that contains server side classes including Servlets, utility classes, properties files etc. The files that are in this directory are automatically added to the web application's Java classpath and should not be packaged in a JAR (Java Archive) file. It is important to note that if the Java classes are packaged, the correct directory structure must be maintained.
- **lib** - a directory that contains JAR files, usually JSP tag libraries and Java APIs, such as the JCWSAPI. JAR files, and all resources they contain, in this directory are automatically added to the web application's Java classpath.

You can create WAR files manually or by using the AAT (Application Assembly Tool) that is bundled with WebSphere 4.0.2. To manually create a WAR file you must arrange your files using the directory structure discussed above, create the **web.xml**, **ibm-web-bnd.xml** and **ibm-web-ext.xml** files and use the **jar** tool that is distributed with WebSphere to consolidate and compress the files. The resulting file must have a **.war** extension. An alternative way to create a WAR file is to use the AAT. The AAT is a graphical tool that greatly simplifies the process of creating WAR and EAR files. It provides wizards for rapid creation and automatically generates the necessary configuration files. More information about the AAT can be found in the WebSphere InfoCenter documentation that comes bundled with WebSphere or on-line at <http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.html> and in the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook*.

## Enterprise Archive (EAR) Files

An EAR file is a compressed collection of J2EE components. An EAR must contain at least one Web Module and optionally contain other J2EE components including EJB Modules and Client Modules. The CWS web application does not have any EJB or Client Modules and can be considered a standalone Web module. WebSphere 4.0.2 allows for Web modules to be installed on their own (internally they are converted to EAR files) which allows for CWS to be distributed as a WAR file instead of an EAR file.

## Application Server

WebSphere 4.0.2 Advanced Edition allows multiple application server instances to be created and managed by the same administrative server. To simplify the discussion an application server can be thought of as a JVM (Java Virtual Machine) and a set of APIs. Web applications can be configured to run inside of different application servers (JVMs) which allows for said applications to execute in different environments.

For example, one application server may host a self-registration CWS instance and a second a CSR instance. Both instances use different copies of the same codebase (JCWSAPI, libwsp.so, etc.) but use different webpages and have different traffic patterns. Furthermore, let's assume that the CWS instance contains many more JSPs and receives many more requests per minute than the CSR server. If this were the case we could increase the CWS JVM heap size to 256MB to better performance while leaving the CSR heap size to the default. Since the different CWS instances run in different application servers we can



customize each instances environment to best suit it s needs. Please note that application servers have many other features, the discussion of which is outside the scope of this document please refer to the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook* for more information.

## Virtual Hosts

Virtual hosts specify which hostnames and ports that WebSphere will listen for HTTP requests. For example, it s possible to configure a web application to only respond to requests made to hosts named *test.ibm.com* and [www.test.ibm.com](http://www.test.ibm.com) on ports *80* and *16000*. It is important to note that a virtual host specification is mapped to one or more web applications, not application servers. That is, an application configured with the virtual host specification defined above can be installed on any application server contained in a given WebSphere instance. Please refer to the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook* for more information about virtual hosts.

---

## Structure Of The ifs.war File

The structure of the ifs.war file packaged with CWS adheres to the J2EE specification outlined in the above section entitled *WAR Files*. The ifs.war file contains the minimum set of files needed for CWS to run in the WebSphere 4.0.2 environment. That is, the ifs.war file does not contain anything usable by a client browser such as test or example Servlets, JSPs or static HTML. In this form the ifs.war file is considered to be *deployable* but not *usable*. Before the ifs.war file is deployed into WebSphere 4.0.2 is must be updated with client CWS code/files. For the remainder of this document the unchanged ifs.war file will be referred to as the **base ifs.war** file.

What follows is a listing of the directories and files contained in the distributed base ifs.war. The directory structure of the base ifs.war file can be observed by extracting the contents of the file using the **jar** command included with WebSphere.

### <doc\_root>

The <doc\_root> directory is the base directory of the ifs.war file and contains the following:

Directories:

```
<doc_root>/META-INF
<doc_root>/WEB-INF
<doc_root>/bin
<doc_root>/docs
<doc_root>/ebiller_logos
<doc_root>/etc
<doc_root>/nls
<doc_root>/tools
```

Files:

None

### <doc\_root>/META-INF

Contains meta information about the files contained inside of the ifs.war file. This directory and it s contents are automatically created by the jar command when the ifs.war file is created.

Directories:

None

Files:

MANIFEST.MF

## <doc\_root>/WEB-INF

The WEB-INF directory contains application server configuration files and the special **classes** and **lib** directories as specified by the J2EE. See the *WAR Files* section above for more details about the directories and files contained in WEB-INF.

Directories:

```
<doc_root>/WEB-INF/classes
<doc_root>/WEB-INF/classes/com/ibm/ifs/resource
<doc_root>/WEB-INF/lib
```

Files:

```
web.xml
ibm-web.bnd.xmi
ibm-web.ext.xmi
```

## <doc\_root>/WEB-INF/classes

This directory contain Servlets, utility classes, as well as property files and any other resource, not contained in a JAR file, that needs to be in the web application s classpath.

Directories:

None

Files:

```
IfsSystem.properties
```

## <doc\_root>/WEB-INF/classes/com/ibm/ifs/resource

This is the default directory for the code to text property resource bundle files and the default property files.

If you change the default directory, create the target directory or ensure that the containing directory has write permission for the userid running the web application.

See the `wsp_ott_prb_path` keyword in the IFS configuration file.

Directories:

None

Files:

```
Ifs3tkErrorTextBundle.properties
IfsCwsErrorTextBundle.properties
IfsPendingTransactionStatus.properties
IfsSystemErrorResourceBundle.properties
readme.txt
```

## <doc\_root>/WEB-INF/lib

The contents of all of the JAR files in this directory are added to the web application s classpath. This directory should only contain JAR files.

Directories:

None

Files:

```
jcwsapi.jar
```

## **<doc\_root>/bin**

Contains CWS native library files and the back-end websrvr executable.

Directories:

None

Files:

```
libmtiso.a
libsmldso.a
libtteso.a
libwsp.so
websrvr
```

## **<doc\_root>/docs**

Contains data used by native code.

Directories:

<doc\_root>/docs/jcwsapi/data

Files:

195 .str files in <doc\_root>/docs/jcwsapi/data directory

## **<doc\_root>/ebiller\_logos**

This is the default directory for downloading the EBiller logos.

If you change the default directory, create the target directory or ensure that the containing directory has write permission for the userid running the web application.

See the wsp\_ebiller\_logos\_path keyword in the IFS configuration file.

Directories:

None

Files:

readme.txt

## **<doc\_root>/etc**

Contains configuration files, sample configuration files and other resources used by CWS.

Directories:

None

Files:

```
build.info
config.sample
config.sample.csr
config.toolkit
httpd.conf.sample
privkey.cbsf
pubkey.cbsf
```

## <doc\_root>/nls

*Contains NLS data for MTI*

Directories:

<doc\_root>/nls/lib

Files:

29 .nls files in <doc\_root>/nls/lib

## <doc\_root>/tools

Contains CWS tools and utilities. `startweb` and `stopweb` start and stop an instance of the IFS Common Web Server (CWS). `lsweb`

Directories:

None

Files:

cfgcomp  
cmdsrv  
comptree  
cwsctl  
generror  
generror.out  
golduser  
igweb  
mqweb  
msgclient  
parefile  
qcodes  
setmapiq  
startweb  
stopweb  
tracefmt  
updateWAR

---

## Extending The Base ifs.war File

As previously stated, the ifs.war file distributed with CWS is deployable but not usable. Transforming the base ifs.war file into a useable state consists of adding a custom CWS application. A custom CWS application can consist of Servlets, JSPs, HTML files, images and other static and dynamic content. Updating the base ifs.war file can be accomplished in two different ways: using the **updateWar** script, the AAT or by manual insertion of files.

### Using updateWAR

The recommended method of updating the base ifs.war file is through the use of the updateWAR script that is distributed with CWS. The **updateWAR** script can be integrated into an existing build environment as a means of fully automating the update process.

**updateWAR** is an AIX korn shell script that extracts files from an existing, specified WAR file, copies in new content and creates a new WAR file containing both the original and updated files. When you run updateWAR, be sure the jar command from Java 1.3.0 is in your PATH.

Below is the output of **updateWAR** when the help flag is specified:

*updateWAR - Updates an existing WAR file by adding new content.*

*updateWAR* adds new content to an existing WAR file by exploding the said WAR file and copying in new content from a specified `<NEW_CONTENT>` directory. The `<NEW_CONTENT>` directory must follow the structure for WAR files as defined in the J2EE specification and should only include files that are intended to be deployed.

For example, source code, test scripts, scratch directories, etc. should be removed from the `<NEW_CONTENT>` directory before running *updateWAR*. After the new content has been copied into the existing WAR file a new WAR file is created, this new WAR file contains all of the files that were in the existing WAR file and all of the files recursively contained in the `<<NEW_CONTENT>`

directory. Note that files from the `<NEW_CONTENT>` directory will overwrite files with the same name in the existing WAR file.

Please refer to the CWS User's Guide for more information about the structure of the `<NEW_CONTENT>` directory.

It should be noted that this is an example of how new content can be added to an existing WAR file. No updates are made to any of the configuration files such as `web.xml` or any of the IBM proprietary extension files (`ibm-web-bnd.xmi` or `ibm-web-ext.xmi`).

It is highly recommended that the Application Assembly Tool (AAT), that comes bundled with WebSphere 4, be used to make any changes to the configuration files and then to archive the updated files along with the rest of your code base. The updated configuration files should be placed in the `<NEW_CONTENT>` directory so that they will overwrite the default configuration files.

Usage: *updateWAR* [-h] [-v] -w `<WAR_FILE>` -c `<NEW_CONTENT>` [-n `NEW_WAR`]

Where:

-h displays help information.

-v causes jar to run in verbose mode.

`<WAR_FILE>` is the path to the base `ifs.war` file.

`<NEW_CONTENT>` is the path to the directory containing the content to add

`<NEW_WAR>` is the name to give the updated WAR file.

Note: jar version 1.3.0 (or higher) must be available.

The following steps/instructions illustrate the recommended approach for using the **updateWAR** script to update the base `ifs.war`.

1. On the build system, create a directory structure that mirrors the one described in the *Structure Of The ifs.war File* section above. That is, create a root directory that will contain a **WEB-INF** subdirectory and adhere to the J2EE specified structure described above. All of the files that will be copied into the updated `ifs.war` file will be placed here. This directory will be referenced as `<build_root>`
2. After compilation of the web application code (Java, etc.), copy all files that will be used by the web application into the appropriate location off of the `<build_root>` directory. Avoid placing extraneous files into the `<build_root>` directory, such as source code, test scripts, etc., because all files in the `<build_root>` directory will be deployed into the WebSphere 4.0.2 environment and may potentially be accessible to end users.
3. Use the AAT to make any changes to the `web.xml`, `ibm-ext-web.xmi` and `ibm-ext-bnd.xmi` configuration files and place them in the `<build_root>/WEB-INF` directory. If you plan on using the configuration files included in the base `ifs.war` file, or if there haven't been any changes to the files from the previous build, this step can be skipped.
4. Run the command:

```
updateWAR -w <path_to_base_ifs.war> -c <build_root> -n <new_war_name>
```

Where:

<path\_to\_base\_ifs.war> is the absolute path to the base ifs.war file  
<build\_root> is the absolute path to the base directory from step 1  
<new\_war\_name> is the path and name to give the updated WAR file

Example:

```
updateWar -w /ibm/ifs/cws/current/bin/ifs.war -c /usr/local/cws_base -n  
/usr/local/cws/current/newifs.war
```

The result is the WAR file /usr/local/cws/current/newifs.war that contains all of the base ifs.war files as well as the files from the /usr/local/cws\_base directory.

## Using The AAT (Application Assembly Tool)

The AAT is a graphical WAR and EAR file generation and manipulation tool that has the ability to automatically generate the **web.xml**, **ibm-ext.web.xmi** and **ibm-ext.bnd.xmi** configuration files. The AAT is a great tool to use to create and update WAR files as well as to generate configuration files. The AAT is an user interactive program and not suitable for automation.

Please refer to the WebSphere InfoCenter documentation for more information about using the AAT to update existing WAR files.

### Deploying Applications In WebSphere 4.0.2

Deploying a CWS web application on WebSphere 4.0.2 consists of installing the updated ifs.war file as well as a number of post-install related activities. Deploying a web application can be looked at from two different perspectives: deploying an application for the first time or redeploying an existing application. Chapter 3 provides step-by-step instructions that illustrate how to deploy a CWS web application for the first time, therefore, the following discussion will focus on redeploying existing web applications. **Note:** Chapter 19 of the *IBM RedBook-RedPiece SG24-6176-00 WebSphere Version 4.0 Advanced Edition Handbook* contains extensive information related to the deployment of applications on WebSphere 4.0.2. It is highly recommended that this material be read before continuing.

Often times new functionality is added to, or problems are fixed in code that is already deployed in a production or test environment. There are two different approaches to updating a deployed web application: the Uninstall/Reinstall approach and the Copy/Replace approach.

## Uninstall/Reinstall Approach

The Uninstall/Reinstall approach is the preferred approach for a production environment. It can be used in a test environment as well. The process consists of the following steps that must be performed everytime code is added or updated to the web application.

1. Uninstall the CWS web application.
2. Reinstall the CWS web application.
3. Perform post-install activities.

What's nice about this approach is that it can be completely automated through scripting and run on all production systems. That is, if the CWS web application is deployed on multiple physical systems, the same script can be run on each system to update the CWS web application resulting in consistent and reproducible results.

The following is a sample korn shell script that performs all three steps listed above on the fictional server *test.ibm.com*, and userid *testUser*. Please note that this is just one way in which the uninstall/reinstall approach can be implemented. For example, instead of using a korn shell script and wscp.sh in command line mode, you could write an equivalent *Tcl* script, using the *exec* command to execute operating system commands, and wscp.sh in interactive mode.

```

#!/bin/ksh

# Stop the application server
/usr/WebSphere/AppServer/bin/wscp.sh -c "ApplicationServer stop
/Node:test/ApplicationServer:testUser/"

# Backup CWS configuration file
cp /usr/WebSphere/AppServer/installedApps/testUser.ear/ifs.war/etc/config
/tmp/config.testUser

# Remove the web application
/usr/WebSphere/AppServer/bin/wscp.sh -c "EnterpriseApp remove
/EnterpriseApp:testUser/"

# Install the updated WAR file
/usr/WebSphere/AppServer/bin/wscp.sh -c "Module install /Node:test/
/updates/ifs.war -contextroot / -appname testUser -modvirtualhosts { {ifs.war
testUser} } -moduleappservers { {ifs.war /Node:test/ApplicationServer:testUser/}
}

# Copy in config file and update permissions
mv /tmp/config.testUser
/usr/WebSphere/AppServer/installedApps/testUser.ear/ifs.war/etc
chmod -R 755 /usr/WebSphere/AppServer/installedApps/testUser.ear/ifs.war/tools
chmod -R 755 /usr/WebSphere/AppServer/installedApps/testUser.ear/ifs.war/bin
chmod 777 /usr/WebSphere/AppServer/installedApps/testUser.ear/ifs.war/WEB-
INF/classes/com/ibm/ifs/resource
chmod 777
/usr/WebSphere/AppServer/installedApps/testUser.ear/ifs.war/ebiller_logos

# Start the application server
/usr/WebSphere/AppServer/bin/wscp.sh -c "ApplicationServer start
/Node:test/ApplicationServer:testUser/"

```

## Copy/Replace Approach

The copy/replace approach is faster than uninstall/reinstall but not as clean or safe and should only be used in a test or development environment. The copy/replace approach process consists of replacing old files with updated versions or copying new files into the web application. To accomplish this you have to stop the application, copy over the new files, then restart the application:

1. Stop the application server (see Appendix F.)

2. Copy over the updated class file:

```

# cp /updates/com/test/Sample.class
/usr/WebSphere/AppServer/installedApps/<userid>.ear/ifs.war/WEB-
INF/classes/com/test

```

If the *com.test.Sample* class was in a JAR file inside of the WEB-INF/lib directory, you will have to replace the entire jar file with the updated version for the changes to take effect.

3. Start the application server (see Appendix F.)

---

# Glossary of Terms and Abbreviations

This glossary includes terms and definitions from:

- The *Dictionary of Computing*, SC20-1699.
- The *Dictionary of Finance and Investment Terms*, by John Downes and Jordan Elliot Goodman, Barron's Financial Guide, 1991.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol (A) after the definition.

The *ANSI/EIA Standard - 440A: Fiber Optic Terminology*, copyright 1989 by the Electronics Industries Association (EIA). Copies can be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue N.W., Washington, D.C. 20006. Definitions are identified by the symbol (E) after the definition.

- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

The following cross-references are used in this glossary:

**Contrast with.** This refers to a term that has an opposed or substantively different meaning.

**See.** This refers the reader to multiple-word terms in which this term appears.

**See also.** This refers the reader to terms that have a related, but not synonymous, meaning.

**Synonym for.** This indicates that the term has the same meaning as a preferred term, which is defined in the glossary.

## A

**ACH.** Automated clearing house.

**Advanced Interactive Executive.** The IBM operating system for the Web server.

**AIF.** Application Integration Feature of FlowMark for MVS.

**AIX.** Advanced Interactive Executive.

**agent.** In Interactive Financial Services, the agent connects the transaction delivery manager to the financial institution. The agent converts the protocols used by the transaction delivery manager to those used by the financial institution. An agent can be one program or a collection of programs necessary to complete the conversion task. See also *back-end agent*.

**API.** Application program interface.

**Application Integration Feature.** Supports workflow capabilities for MVS/ESA.

**application program interface (API).** Software and standards used to support data integration between applications.

**architecture.** A definition of a system in terms of its components, how these components interact, and the interfaces used to assure that the components work together properly.

**ATM.** Automatic teller machine.

**audit trail.** Record that allows detailed tracking of transaction processes within the IFS system. Also, a way of tracking and verifying basic information about the status of trouble tickets.

**authentication.** The process for an financial institution to verify the identify of a customer attempting to log onto IFS or another financial institution system. Used for certifying individual or company identity.

**authorization.** Approved access levels once identity is verified.

**automated clearing house (ACH).** (1) A value-dated electronic funds transfer system governed by ACH rules and used in the U.S. for recurring payments. Transfers can be credit or debit, and settlement is for one or two business days after they are processed. Bank funds are provisional until the morning of the business day following the settlement day. The Reserve Bank may revoke the payments if the sending bank does not have sufficient funds in its account to fund them on the settlement date. (2) The actual organizations and computer facilities, operated by the Federal Reserve or others, that process the electronic transfers.

**automatic teller machine (ATM).** A self-service unit that allows the user, with suitable identification and account relation, to carry on financial transactions, such as a cash withdrawal.



**automation.** User-written operations programs that control repetitive and routine events on the IFS hardware and software components, for example monitoring conditions and workload.

## B

**back-end agent (BEA).** (1) A component of the Strip File Application that accepts incoming standard messages, forwards them to the online standard message processor (which gets a reply from the VSAM data sets created from strip files), and returns the reply to the requestor. (2) A deprecated term for (and see) *IFS interface module*.

**backup.** The process of saving copies of IFS data in case of a system failure. Operators back up data on a regular schedule, usually daily.

**bank holding company.** A company owning one or more financial institutions.

**BEA.** Back-end agent.

**bill originator (biller).** Entity that originates a bill; for example, a utility company, mortgage company, credit card company, or retail store.

**bill payment.** A transfer service that allows a customer to present a payment for a certain type of bill (such as an electricity bill or municipal rate) to the financial institution, which then processes the payment and reconciles it with the organization to which it is addressed.

**biller.** See *bill originator*.

**biller database.** A database of bill originators with relevant data about each bill originator for facilitating the bill payment process.

**browser.** A network client program that communicates with Web servers, used for retrieving and displaying documents from the World Wide Web.

## C

**CGI.** Common gateway interface.

**check.** See *paper check*.

**CICS.** Customer Information Control System.

**client.** An individual person using a Web browser or telephone to initiate a transaction, also called a customer or consumer. See also *end user*.

**client/server.** A concept that divides the processing for a unit of work between the end-user's work station (the client) and one or more other computing devices (servers) that provide service to multiple users through a network connection.

**Common Gateway Interface (CGI).** An interface that governs the interaction (communication) between the HTTP server and server gateway

programs, which act upon local resources such as databases. A specification to allow a Web server to communicate with other programs and systems. This feature of Web servers allows browsers to communicate over the Web with scripts installed on the server. CGI scripts can be written in any programming language that will run on the Web server.

**consumer.** An individual person using a Web browser or telephone to initiate a transaction, also called a client or customer. See also *end user*.

**core controller.** An IFS component responsible for routing and scheduling IFS transactions and ensuring transaction integrity. Also called IFS core controller. When working with other parts of a system, then referred to as transaction delivery manager.

**CRC.** Cyclic redundancy check.

**CSR.** Customer support representative.

**CSR/TSR workstation.** A tool used by Level 1 and Level 2 support representatives to view the IFS logs and standard messages. Also referred to as CTW.

**customer.** An individual person using a Web browser or telephone to initiate a transaction; also called a client or consumer. See also *end user*.

**CTW.** See *CSR/TSR workstation*.

**Customer Information Control System (CICS).** An IBM system used as a transaction monitor.

**customer profile.** A file that contains attributes associated with a user of the system, such as name, address, account numbers, and similar information.

**customer support representative (CSR).** A person who assists end users when they have a question or a problem; also called Level 1 support.

## D

**DATABASE 2 (DB2).** An IBM relational database management system.

**DB2.** DATABASE 2

**device-independent.** Using an open architecture and translating device messages into standard interface. Device-independence helps ensure the ability to add new devices without disrupting the existing system.

**dialog.** A graphical interface that permits a user to enter information and send it to the processor for a response.

**diary.** The mechanism used to track future financial transactions; for example, standing orders to pay mortgage on the first of each month; frequently used by bill pay and funds transfer applications.

## E

**EAR (enterprise archive).** A file used for deploying web applications under an application server such as IBM WebSphere. The file is derived from the ZIP archive format but it contains a directory format and files contents as defined by Java/2 Enterprise Edition conventions. The EAR file contains all the contents of a WAR file plus a META-INF directory that describes how and where to deploy the WAR file.

**electronic commerce.** The end-to-end digital exchange of all information needed to conduct business, such as EDI transactions, electronic mail, audit trails, or graphics.

**element.** The structural building block of HTML documents.

**E-mail.** Electronic mail.

**encryption.** Using a secret key to encipher data so that it is unintelligible to anyone except another holder of the secret key. Processing that includes link-level encryption (secures all communications), network-level encryption (for example, IP tunneling between firewalls, IP encryption between two nodes), session-level or transport-level encryption (secures all TCP communications), and application-level encryption (such as HTTP, FTP, NNTP, Telnet).

**end user.** The user of IFS, also known as a customer of a financial institution. The end user uses FI services such as requesting account information, conducting financial transactions, or paying bills.

**enterprise archive (EAR).** A file used for deploying web applications under an application server such as IBM WebSphere. The file is derived from the ZIP archive format but it contains a directory format and files contents as defined by Java/2 Enterprise Edition conventions. The EAR file contains all the contents of a WAR file plus a META-INF directory that describes how and where to deploy the WAR file.

## F

**field.** An atomic data type, such as a character string or number, that is the basic unit of data representation within the message body. A given field may have zero or more field instances (or occurrences) associated with it. The instance concept allows the implementation of vector data types based on the atomic data types supported by the MTI.

**file transfer protocol (FTP).** An Internet client/server protocol for transferring files between computers.

**financial institution (FI).** An institution capable of providing all the banking services supported by Interactive Financial Services.

**financial institution profile.** A file that contains attributes of the financial institution including rules for transaction processing, logging, and formatting for the financial institution; and their customers, products and transactions. It establishes which IFS features will be used and the default processing options for customers. General information about the financial institution is also included. The financial institution profile allows the financial institution and IBM to identify user IDs with responsibility for customer profile maintenance, inquiry, and so on.

**firewall.** Software and possibly hardware that protects the system application data (for example, IFS) from hackers and other problems on the Internet and from other systems and entry points.

**FlowMark.** IBM family of products that supports workflow capabilities.

**form.** (1) A group of elements in an HTML document that generate graphical controls such as text boxes, radio buttons, and check boxes when the document is displayed by a browser. The user can enter data in a form and use the browser to submit it to a program on a server. (2) An interface that allows you to enter information into a database. A form also controls how that information is displayed in the database and printed. To create a document, you fill in a form in a database. A database can contain one or more forms.

**formats.** In the financial industry, a format is a specific arrangement of data used for acknowledgments, purchase orders, and invoice data. Some examples of formats are ACH formats (CTX; CTP; CCD+) and FEDI formats (820, 822, 823, 828, 835); and EDI formats.

**free-form messages.** IFS electronic mail messages that allow users to define the content and format to communicate with customer support representatives.

**FTP.** File transfer protocol.

## G

**gateway.** A connection between two networks that converts messages from one to be communicated to the other.

**Gold Standard Message.** A well-structured framework for defining messages used within the IFS application. The standard message is independent of operating systems, programming languages, and communications or messaging software.

## H

**helper.** A program launched or used by a browser to process files that the browser cannot handle on its own. Similar to a plug-in with the difference that it need not be integrated into the browser, but may be used by the browser for specific tasks such as image viewing. See also *plug-in*.

**home page.** The first page of a World Wide Web site that introduces the site and likely includes hypertext links to local resources.

**HotJava.** A World Wide Web browser produced by Sun Microsystems that uses a new programming language called Java. HotJava provides the ability to import code fragments, called applets, across the Internet and execute them.

**HTML.** HyperText Markup Language.

**HTTP.** HyperText Transfer Protocol.

**HyperText Markup Language (HTML).** A structured language for documents that are published on the Web.

**HyperText Transfer Protocol (HTTP).** An Internet, client/server protocol designed expressly for rapid distribution of hypertext documents.

## I

**IBM Global Services.** The world-wide IBM solution provider that provides network solutions and a global information infrastructure.

**IBM Messaging Queuing Series (MQSeries).** Products that manage transactions and enable programs to talk to each other across a network of unlike component (processors, operating systems, subsystems, and communication protocols) using a simple and consistent application programming interface.

**IFS.** Interactive Financial Services.

**IFS Alliance.** A user group of financial institutions that use the IFS platform to provide home banking online financial services in the Asia-Pacific region.

**IFS core controller.** See *core controller*.

**IFS data gateway (IGATE).** MQSeries communications between IFS on the IBM host processor and the CSR/TSR workstation. It validates the CSR and TSR logon requests.

**IFS infrastructure.** Part of the end-to-end IT solution (from the IFS user to the financial service application) that is owned or operated on-behalf-of the financial institution by IBM.

**IFS interface module.** A module, provided by the financial institution, that interfaces with the IFS transaction delivery manager. For each request sent to the financial institution, the IFS interface module converts the standard message to the financial institution s protocol and reconverts the

reply back to the standard message format for IFS.

**IGATE.** IFS data gateway.

**implementation manager.** An IBM staff member who is responsible for delivering IBM s commitments and tasks related to a financial institution s implementation plan for IFS.

**integrity.** Processing to guarantee that a transmission arrives in exactly the same form in which it was sent.

**Interactive Financial Services (IFS).** A consortium of IBM and banking partners that provide a network-centered, end-to-end, online banking solution between consumer financial institutions and their customers.

**International Organization for Standardization (ISO).** The world body that establishes global technology standards.

**intranet.** A private network within the World Wide Web network.

**ISO.** International Organization for Standardization.

## J

**JAR (Java archive).** A file used for running Java programs using a Java Virtual Machine. The file is derived from the ZIP archive format but it contains a directory format and files contents as defined by Java conventions. The jar file contains all of the classes, native libraries, properties, and resources to run a Java application.

**J2EE (Java/2 Enterprise Edition).** A platform for running Java web applications. The platform contains all of the programs and packages of the Java/2 Standard Edition, plus packages for writing and deploying servlets, JSPs and web applications.

**JSPs (Java Server Pages).** A programming language with markup tags for writing web pages. JSPs are best used to display dynamic web content, as the JSPs are translated to servlets and web pages by the web server when a user with a browser requests a web page.

**Java.** An object-oriented programming language developed at Sun Microsystems. Used extensively within HotJava, the WWW browser. In big-picture terms, Java is like C++ without pointers and with a few add-ons (to ensure greater security). Java includes executable code that can be downloaded from the server to the client to be executed.

**Java Database Connection (JDBC).** An API developed by Sun and implemented by many database vendors. The API provides a standard set of interfaces for Java programs to talk to databases such as IBM DB/2..

## L

**LAN.** Local area network.

**Level 1.** A customer support representative (CSR) who is the first point of contact for a customer.

**Level 2.** A technical support representative (TSR) who provides technical assistance for Level 1 CSRs when they need specific help with an IFS problem.

**LIG.** Local interface gateway.

**link.** In hypertext documents, the connection from one document to another.

**Local area network (LAN).** A computer network located within a limited geographical area.

**local interface gateway (LIG).** A machine that provides access to the IBM Global Network.

**log.** For IFS, a collection of standard messages placed in a database to record IFS events or for accounting or data collection purposes.

**LU 6.2.** A type of logical unit that supports general communications between programs in a distributed processing environment. LU 6.2 is characterized by (a) a peer relationship between session partners, (b) efficient use of a session for multiple transactions, (c) comprehensive end-to-end processing, and (d) a generic application program interface (API) consisting of structured verbs that are mapped into a product implementation.

## M

**MAPI.** Message API.

**mapping.** A process of utilizing user-defined data maps to translate business documentation from in-house data format to an EDI standard, or from EDI to an in-house format.

**Message API (MAPI).** A command language that converts messages into AIF format, so they can be processed by a script.

**message translation interface (MTI).** An interface used for constructing, populating, converting, and unpacking a message.

**messaging.** Communication between programs by sending data in messages rather than by calling each other directly.

**messaging and queuing (MQ).** IBM's MQSeries products that enable programs to talk to each other across a network of unlike components (processors, operating systems, subsystems, and communication protocols) using a simple and consistent application programming interface.

**messaging and queuing interface (MQI).**

Queuing interface for message transport.

**META-INF.** A directory within a JAR, WAR, or EAR file. This directory is reserved for the manifest.mf file which describes the archive directory structure and structure version.

**MIME.** Multipurpose Internet mail extensions.

**modem.** A device that enables computer data to be exchanged over telephone lines.

**MQSeries.** IBM Messaging Queuing Series.

**MTI.** Message translation interface.

**multipurpose Internet mail extensions (MIME).** E-mail allowing for sound, video, image, and text.

**MVS/ESA.** IBM's Multiple Virtual Storage/Enterprise Systems Architecture.

## N

**Netscape Communicator.** An Internet browser program that end users can use to access IFS.

**network.** An arrangement of nodes and connecting branches used for communications.

**network services.** See *IFS network services*.

**Notify.** The portion of the IBM Global Network services that Level 1 and Level 2 support personnel use to communicate and track customer problems with trouble tickets.

## O

**on-behalf-of (OBO).** Tasks a financial institution customer support representative (CSR) does at the request of customers. An example of an on-behalf-of customer task would be to add a payee for a customer using a voice response system for bill payment.

## P

**packet.** Small amount of data that is packaged for transmission over a link. See TCP/IP.

**paper check.** Non-same-day, paper form of debit transfer wherein bill originators collect from payers. Funds made available by banks to depositors of checks are provisional and may be reversed if the payer's account does not have sufficient funds to pay the check when it is received by the payer's bank.

**password.** A combination of characters the user types when logging on, designed to be secret to prevent unauthorized access of the user's data.

**payment request.** Process in which, after submitting a list of accounts to be eligible for payment and a list of payees to make payments to, the customer uses an access device to request the bank to pay a biller a specified amount.

**payee.** A person, company, or organization that receives payments. The bill payment application recognizes three types of payees: biller, free-form, and interbank.

**pay/no pay decision.** Determination made by the consumer's financial institution whether to release a payment instruction into the IFS system for delivery to a biller. Once a pay decision has been made, the consumer financial institution cannot recall the transaction, even if the consumer is unable to fund the payment instruction.

**PIN.** Personal identification number. See *password*.

**ping.** An Internet protocol for requesting an echo/response from another element in the network; a way of ascertaining whether another network element is up and functional.

**plug-in.** A dynamic code module, native to a specific platform on which the Netscape client runs, the primary goal of which is to provide seamless new datatype support for Netscape users.

**point-to-point protocol (PPP).** A protocol that provides router-to-router and host-to-network connections over both synchronous and asynchronous circuits. PPP supports Internet Protocol (IP) and other protocols.

**port.** A communication channel.

**PPP.** Point-to-point protocol.

**prenote.** (1) A zero dollar transaction sent through ACH or other proprietary services (MasterCard, etc.). (2) A hardcopy letter sent to the biller. The prenote is used to validate that the biller or biller financial institution will be able to accept and process payment instructions supplied by the consumer in advance of the first actual payment.

**privacy.** Condition in which a message is understood only by appropriate parties.

**private key.** A personalized value in the use of an encrypting algorithm similar to the use of a combination number for a vault, for secure electronic payments.

**profile.** IFS data that describes characteristics of an end user, an financial institution, or a service.

**properties file.** A file of option settings that are read by a Java program at runtime. The file contains comments and key/value pairs that determine which code paths a Java program executes.

**protocol.** An agreed convention for computer-to-computer communications. For example, TCP/IP defines how messages are passed on the Internet. FTP protocol, which is built on top of TCP/IP, defines how FTP messages are sent and received.

**proxy.** A substitute for a person (can be software).

**public key.** A value in the use of an encrypting algorithm similar to the use of a combination number for a vault. This key is used for secure electronic payments.

## Q

**queuing.** A function that stores messages in queues so that programs can run independently of each other, at different speeds and times, in different locations, and without having a logical connection between them.

## R

**RACF.** Resource Access Control Facility.

**Resource Access Control Facility (RACF).** An MVS/ESA security program that protects system resources.

**RISC.** Reduced instruction set computer.

**RISC System/6000.** An IBM processor based on the RISC technology; also known as an RS/6000.

**RSA encryption.** The Rivest-Shamir-Adleman algorithm, used by FIs to encrypt passwords in the strip files they send to the Core Banking with Strip Files Application.

## S

**script.** Programming code that defines a list of FI business rules and processes.

**secure HTTP.** An extension to HTTP that secures the form used in communications and performing transactions.

**secure sockets layer (SSL).** Secures the pipe.

**self-defining (field or structure).** A field or structure that is specified by the financial institution using appropriate MTI APIs and is sent as part of the message. The receiving program does not require external definition information in order to decode the message element.

**Self-registration.** Allows a customer to self-register for IFS services electronically. The customer must have the appropriate software installed on the PC to do this.

**serial line Internet protocol (SLIP).** A protocol that runs Internet Protocol (IP) over serial lines, such as telephone circuits or RS-232 cables connecting two systems.

**server.** A program running on a networked computer that responds to requests from clients (programs) communicating through a client/server protocol.

**service.** An IFS capability, such as core banking, bill pay, diary, transfers.

**service request.** Individual transaction type that supports a service, such as logon, logoff, balance inquiry, add payee.

**servlet.** A Java program that is run by a web application server when a user with a browser requests a web page. Servlets are the basis of dynamic, customizable web pages.

**SLIP.** Serial line Internet protocol.

**S/MIME.** Secured/Multimedia Interactive Mail Extension. An E-mail technology.

**SNA.** Systems Network Architecture.

**standard message.** See Gold Standard Message

**structure.** Used to represent complex data types consisting of any combination of fields or other structures. A given structure may have zero or more instances (or occurrences) associated with it.

**Systems Network Architecture (SNA).** An architecture that provides communications functions for all applications. It simplifies application programming by putting the terminal definition and support functions in a common, IBM-supplied program.

## T

**tag.** Marks the beginning (start-tag) or end (end-tag) of an element in an HTML file.

**TCP/IP.** Transmission Control Protocol/Internet Protocol.

**TDM.** Transaction delivery manager.

**technical support representative (TSR).** An Level 2 technical person who assists the Level 1 CSR.

**Transaction.** See *service request*.

**transaction delivery manager (TDM).** When the core controller works with other parts of the system as an intelligent router. It consists of an IBM MVS/ESA system with workflow software that handles the transactions required for a financial transaction.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** The basic (lowest-level) communication protocol of the Internet; it breaks messages up into small packets, and sends them independently to their destination, where they are assembled without error in transmission.

**Trumpet Winsock.** A Windows Sockets 1.1 compatible TCP/IP stack that provides a standard networking layer for many Windows networking applications to use. It allows users to open multiple sessions at once, including Netscape, FTP, Telnet, and others.

**TSR.** Technical support representative.

## U

**uniform resource locator (URL).** A pointer used in hypertext to address Internet resources on the World Wide Web.

**URL.** Uniform resource locator.

**user.** See *customer* or *end user*.

**utility.** Reference to financial applications, for example, bill payment.

## V

**Vendor.** An individual, company, or entity that has billed a bank customer and is expecting a payment. See *bill originator*.

**voice response system (VRU).** An automated device used to route, record, and act upon incoming calls from customers.

**VRU.** Voice response unit.

**VTAM (Virtual Telecommunications Access Method).** A program that provides communication facilities for IFS.

## W

**WAR (web archive).** A file used for deploying web applications under an application server such as IBM WebSphere. The file is derived from the ZIP archive format but it contains a directory format and files contents as defined by Java/2 Enterprise Edition conventions. The war file contains all of the contents of a servlet JAR file, plus information to how and where to load, run and describe a Java servlet.

**Web browser.** See *browser*.

**WEB-INF.** A directory within a JAR, WAR, or EAR file that typically contains deployment information. The deployment information describes the program and gives details such as how to name or where to run the program and often is written in XML format.

**Web pages.** The financial institution's HTML panels that end users use for IFS inquiries and transactions.

**Web server.** See *server*.

**Window.** A portion of the screen set aside for a particular task or function.

**Windows.** The operating system on an end user home computer that controls the computer and its software (which includes the IFS browser).

**Workflow.** Ability to flow work in an orderly process, ensuring that the right person receives the right information to perform the required task or make the required decision.

**Workflow management.** A standard set of business and automation processes that an FI uses to respond to customer s service requests in a consistent and controlled manner.

**World Wide Web (WWW).** A generic term for the collection of Web servers and browsers.

---

## Bibliography

---

### Interactive Financial Services Library

The following documents are part of the IFS library:

- *IFS Overview and Release Changes Guide*
- *IFS Planning Guide*
- *IFS Transaction Delivery Manager Guide*
- *IFS Customizable Web Server Guide*
- *IFS Messaging Implementation and Installation Guide*
- *IFS Service Provider Adapter Guide*
- *IFS Open Financial Exchange Translator Guide*
- *IFS National Language Support Guide*
- *IFS Message Modification Process Guide*
- *IFS Customer Service Guide*
- *MTI Programming Guide and Reference for C*
- *MTI Programming Guide and Reference for COBOL*
- *MTI Programming Guide and Reference for JAVA*
- *MTI Programming Guide and Reference for RPG*
- *IFS Gold Standard Message Transaction Specification*

*IFS Gold Standard Message Data Dictionary*

---

### Other Documents

The following documents may be of interest:

*HSBC JAVA CWS Performance White Paper 12/20/00*

---

### CTW HELP

The CTW s Help is online computer documentation designed to support your ongoing work. It consists of Help topics, or chunks of Help information, that are in many ways similar to sections of a print manual. Many of the topics are procedure topics, containing numbered steps preceded by some kind of conceptual information.

---

### Other Publications

You might want to refer to the following IBM publications for related information:

*MQSeries Application Programming Guide*, order number SC33-0807

*FlowMark for MVS/ESA Application Integration Feature: Writing Scripts and Agents*, order number GC28-1245

If you need to see publications for an outside service provider (such as Checkfree or Intuit), contact that company s representative. You can also access the service provider s Web site for more information.



---

## Reader's Comments – We'd Like to Hear from You

Interactive Financial Services  
Customizable Web Server (CWS) Guide

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

**Note:** Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: \_\_\_\_\_

What is your occupation?

How did you use this publication?

- ☐ As an introduction
- ☐ As a reference manual
- ☐ For another purpose (explain)

- ☐ As a text (student)
- ☐ As a text (instructor)

---

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number:              Comment:

Name              Address

Company or Organization

Phone No.

*Please mail completed form to:*

*IFS Publication Coordinator  
IBM Manassas  
10511 Battleview Parkway  
Manassas, VA 20109  
USA*



Printed in the United States of America  
on recycled paper containing 10%  
recovered post-consumer fiber.