



IMS Adapters Administrator's Guide

Version 6.0, November 2003

IONA, IONA Technologies, the IONA logo, Orbix, Orbix/E, Orbacus, Artix, Orchestrator, Mobile Orchestrator, Enterprise Integrator, Adaptive Runtime Technology, Transparent Enterprise Deployment, and Total Business Integration are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

CORBA is a trademark or registered trademark of the Object Management Group, Inc. in the United States and other countries. All other trademarks that appear herein are the property of their respective owners.

While the information in this publication is believed to be accurate, IONA Technologies PLC makes no warranty of any kind to this material including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IONA Technologies PLC shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

COPYRIGHT NOTICE

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of IONA Technologies PLC. No third party intellectual property right liability is assumed with respect to the use of the information contained herein. IONA Technologies PLC assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice.

Copyright © 1998, 2003 IONA Technologies PLC. All rights reserved.

All products or services mentioned in this manual are covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Updated: 06-Apr-2005

M 3 1 6 7

Contents

List of Figures	ix
List of Tables	xi
Preface	xiii
Part 1 Introduction	
Chapter 1 Introduction to CORBA and Orbix	3
Overview of CORBA	4
Why CORBA?	5
CORBA Objects	7
The ORB	9
CORBA Application Basics	10
Overview of Orbix	12
Simple Orbix Application	13
Broader Orbix Environment	16
Chapter 2 Introduction to the IMS Adapters	19
Overview of the IMS Server Adapter	21
Role of the IMS Server Adapter	22
IMS Server Adapter Processing of IDL Operations	25
The IMS Server Adapter imsrw Interface	26
Unsupported IDL Types	35
Overview of the Client Adapter	36

Part 2 Configuring the IMS Server Adapter and the Orbix Runtime inside IMS

Chapter 3	Introduction to IMS Server Adapter Configuration	43
	An IMS Server Adapter Sample Configuration	44
	Configuration Summary of Adapter Plug-Ins	48
Chapter 4	IMS Server Adapter Service Configuration Details	55
Chapter 5	Configuring the IMS Server Adapter OTMA Plug-In	63
	Setting Up OTMA for the IMS Server Adapter	64
	OTMA Plug-In Configuration Items	67
Chapter 6	Configuring the IMS Server Adapter APPC Plug-In	71
	Setting Up APPC for the IMS Server Adapter	72
	Defining LUs to APPC	73
	Defining an APPC Destination Name for the IMS LU	75
	Defining LUs to VTAM	77
	Additional RACF Customization Steps for APPC	80
	APPC Plug-In Configuration Items	83
Chapter 7	Configuring the IMS Server Adapter RRS Plug-In	85
	Introduction to RRS	86
	Setting up RRS for the IMS Server Adapter	87
	RRS Plug-In Configuration Items	94
Chapter 8	Configuring the IMS Server Adapter for Client Principals	95
	Activating Client Principal Support	97
	Setting up the Required Privileges	100
	Additional Requirements for IMS Protocol Plug-Ins	102
Chapter 9	Configuring the Orbix Runtime inside IMS	107
	Customizing the IMS JCL	108
	Customizing Orbix Event Logging	109

Chapter 10	IDL Compiler Configuration	111
Part 3 Configuring the Client Adapter and the Orbix Runtime inside IMS		
Chapter 11	Introduction to Client Adapter Configuration	117
	A Client Adapter Sample Configuration	118
	Configuration Summary of Client Adapter Plug-Ins	120
Chapter 12	Client Adapter General Configuration	123
Chapter 13	Configuring the Client Adapter AMTP_APPC Plug-in	125
	Setting Up APPC for the Client Adapter	126
	Defining LUs to APPC	127
	Defining an APPC Destination Name for the Client Adapter	130
	Defining LUs to VTAM	134
	Additional RACF Customization Steps for APPC	138
	LU-to-LU Security Verification	139
	Protecting LUs	141
	Enabling APPC/IMS	143
	AMTP_APPC Plug-In Configuration Items	144
Chapter 14	Configuring the Client Adapter mfu Plug-in	147
Chapter 15	Configuring the Orbix Runtime Inside IMS	149
	Customizing the IMS JCL	150
	Customizing Orbix Configuration	151
	Customizing Orbix Event Logging	153
	Customizing Orbix Maximum Segment Size	155
	Customizing Orbix APPC Function Timeout	157
	Customizing Orbix APPC Symbolic Destination	158
	Customizing Orbix APPC Local LU	160

Part 4 Securing and Using the IMS Server Adapter

Chapter 16	Securing the IMS Server Adapter	163
	Security Configuration Items	164
	Common Security Considerations	169
	APPC-Based Security Considerations	171
	OTMA-Based Security Considerations	177
	IMS Server Adapter Security Modes	179
	Choosing between OTMA and APPC Modes	182
	Setting up APPC and OTMA Modes	183
Chapter 17	Mapping IDL Interfaces to IMS	185
	The Mapping File	186
	Characteristics of the Mapping File	187
	Generating a Mapping File	189
	Using the IFR as a Source of Type Information	192
	Introduction to Using the IFR	193
	Registering IDL interfaces with the IFR	195
	Informing IMS Server Adapter of a New Interface in the IFR	198
	Using an IFR Signature Cache File	200
	Using type_info store as a Source of Type Information	202
	Introduction to Using a type_info Store	203
	Generating type_info Files using the IDL Compiler	205
	Informing IMS Server Adapter of a new type_info Store File	207
Chapter 18	Using the IMS Server Adapter	211
	Preparing the Server Adapter	213
	Starting the Server Adapter	217
	Stopping the IMS Server Adapter	219
	Running Multiple Server Adapters Simultaneously	220
	Using the MappingGateway Interface	222
	Locating IMS Server Adapter Objects Using itmfaloc	225
	WFI Support for IMS Transactions	228
	Conversational Support	229
	LTERM Propagation	233

Adding a Portable Interceptor to the IMS Server Adapter	234
Developing the Portable Interceptor	235
Compiling the Portable Interceptor	239
Loading the Portable Interceptor into the IMS Server Adapter	241
Gathering Accounting Information in the Server Adapter	244
Developing the Accounting DLL	245
Compiling the Accounting DLL	248
Loading the Accounting DLL into the Server Adapter	249
Exporting Object References at Runtime	250
Configuration Items for Exporting Object References	251
Exporting Object References to a File	253
Exporting Object References to Naming Service Context	254
Exporting Object References to Naming Service Object Group	256
Part 5 Securing and Using the Client Adapter	
Chapter 19 Securing the Client Adapter	263
Introduction to Client Security	264
APPC Security Considerations	265
LU Security	266
Conversation Security	269
Chapter 20 Using the Client Adapter	273
Starting the Client Adapter	274
Stopping the Client Adapter	276
Running Multiple Client Adapters Simultaneously	277
Load Balancing with Multiple Client Adapters	278
Running Two Client Adapters on the Same OS/390 Host	280
Chapter 21 Glossary of Acronyms	283
Index	287

CONTENTS

List of Figures

Figure 1: The Nature of Abstract CORBA Objects	7
Figure 2: Role of the ORB in the Basic CORBA Model	9
Figure 3: Invoking on a CORBA Object	11
Figure 4: Overview of a Simple Orbix Application	13
Figure 5: Graphical Overview of the Role of the IMS Server Adapter	23
Figure 6: Graphical Overview of the Role of the Client Adapter	38
Figure 7: Graphical Overview of a Load Balancing Scenario	278
Figure 8: Running Two Client Adapters on the Same OS/390 Host	281

LIST OF FIGURES

List of Tables

Table 1: Initial and Maximum Log Stream Sizes	89
Table 2: Client Principal Support and imsa Plug-In Configuration Items	98
Table 3: Event Logging Settings for the IMS Server Adapter	109
Table 4: Server Adapter Mapping Member Configuration Settings	113
Table 5: S390 Assembler Program Variables and Default Values	152
Table 6: Event Logging Settings for the IMS Server Adapter	153
Table 7: IMS LU and Client Adapter LU Required Keyword Definitions	266
Table 8: Glossary of Acronym Extensions	283

LIST OF TABLES

Preface

Orbix is a full implementation from IONA Technologies of the Common Object Request Broker Architecture (CORBA), as specified by the Object Management Group. Orbix complies with the following specifications:

- CORBA 2.3
- GIOP 1.2 (default), 1.1, and 1.0

Orbix Mainframe is IONA's implementation of the CORBA standard for the OS/390 platform. Orbix Mainframe documentation is periodically updated. New versions between releases are available at <http://www.iona.com/docs>.

Support

If you need help with this or any other IONA products, contact IONA at support@iona.com. Comments on IONA documentation can be sent to docs-support@iona.com.

Audience

This guide is intended for IMS system programmers who want to configure, secure, and use the IMS server adapter and client adapter that are supplied with Orbix Mainframe. It is assumed that the reader is familiar with the basic concepts of CORBA 2.3 and IMS administration.

Related Documentation

Orbix Mainframe documentation includes the following related guides:

- *CICS Adapters Administrator's Guide*
- *COBOL Programmer's Guide and Reference*
- *PL/I Programmer's Guide and Reference*
- *CORBA Programmer's Guide, C++*
- *CORBA Programmer's Reference, C++*
- *CORBA Administrator's Guide*

- *Mainframe Security Guide*
- *Mainframe Migration Guide*
- *Mainframe Management Guide*
- *Mainframe CORBA Concepts Guide*

The *Orbix IMS Adapter Programmer's Guide*, which is based on Orbix 2.3.x rather than Orbix Mainframe 6.x, is also a useful reference. For migration issues refer to the *Mainframe Migration Guide*.

For the latest version of all IONA product documentation, see the IONA web site at: <http://www.iona.com/support/docs>

Organization of This Guide

This guide is divided into the following parts:

Part 1, “Introduction”

This part introduces Common Object Request Broker Architecture (CORBA), and Orbix, IONA's implementation of CORBA. It also introduces IMS server adapter which is an Orbix server that can connect with IMS, as well as the client adapter which is an Orbix client that allows IMS transactions to act as clients to Orbix servers.

Part 2, “Configuring the IMS Server Adapter and the Orbix Runtime inside IMS”

This part describes how to configure the IMS server adapter and the Orbix runtime inside IMS.

Part 3, “Configuring the Client Adapter and the Orbix Runtime inside IMS”

This part describes how to configure the client adapter and the Orbix runtime inside IMS.

Part 4, “Securing and Using the IMS Server Adapter”

This part explains security considerations for the IMS server adapter, and how the server adapter can be used as a bridge between CORBA-based messages and IMS programs.

Part 5, “Securing and Using the Client Adapter”

This part explains security considerations for the client adapter, and how the client adapter can be used as a bridge between IMS programs and CORBA-based messages.

Glossary of Acronyms

This glossary provides an expansion for each of the acronyms used in this guide.

Index

This index is sorted letter-by-letter as opposed to word-by-word.

Additional Related Resources

The IONA knowledge base contains helpful articles, written by IONA experts, about the Orbix Mainframe, and other products. You can access the knowledge base at the following location:

http://www.iona.com/support/knowledge_base/

The IONA update center contains the latest releases, and patches for IONA products:

<http://www.iona.com/support/updates/>

Typographical Conventions

This guide uses the following typographical conventions:

Constant width	Constant width (courier font) in normal text represents portions of code, and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the <code>CORBA::Object</code> class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: <code>#include <stdio.h></code>
<i>Italic</i>	Italic words in normal text represent <i>emphasis</i> and <i>new terms</i> .
<i>Code italic</i>	Italic words or characters in code and commands represent variable values that you must supply; for example: <code>install-dir/etc/domains</code>
Code Bold	Code bold is used to highlight a piece of code within a particular code sample.

Keying Conventions

This guide might use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
\$	A dollar sign represents the OS/390 UNIX System Services command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
.	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Brackets enclose optional items in format and syntax descriptions.
{ }	Braces enclose a list from which you must choose an item in format and syntax descriptions.

Part 1

Introduction

In This part

This part contains the following chapters:

Introduction to CORBA and Orbix	page 3
Introduction to the IMS Adapters	page 19

Introduction to CORBA and Orbix

The Common Object Request Broker Architecture (CORBA) standard is specified by the Object Management Group (OMG) and provides the foundation for flexible and open systems. It underlies some of the Internet's most successful e-business sites, and some of the world's most complex and demanding enterprise information systems. Orbix is a full implementation of the CORBA standard from IONA Technologies. Orbix Mainframe is IONA's implementation of CORBA for the OS/390 platform. This chapter provides an introductory overview of both CORBA and Orbix.

In this chapter

This chapter discusses the following topics:

Overview of CORBA	page 4
Overview of Orbix	page 12

Overview of CORBA

Overview

The Common Object Request Broker Architecture (CORBA) provides the foundation for flexible and open systems. It underlies some of the Internet's most successful e-business sites and some of the world's most complex and demanding enterprise information systems. This section provides an overview of CORBA in terms of the enterprise information solutions that it provides and the basic principles on which it is based.

In this section

This section discusses the following topics:

Why CORBA?	page 5
CORBA Objects	page 7
The ORB	page 9
CORBA Application Basics	page 10

Why CORBA?

Overview

CORBA is a standard middleware architecture that can be used to develop and integrate a wide variety of distributed systems that use a variety of hardware, operating systems, and programming languages.

This subsection discusses the following topics:

- [Need for open systems](#)
 - [Need for high-performance systems](#)
 - [Open standard solution](#)
 - [Widely available solution](#)
-

Need for open systems

Today's enterprises need flexible, open information systems. Most enterprises must cope with a wide range of technologies, operating systems, hardware platforms, and programming languages that need to work together to make the enterprise function.

Need for high-performance systems

Orbix is a CORBA development platform for building high-performance systems. Its modular architecture supports the most demanding needs for scalability, performance, and deployment flexibility. The Orbix architecture is also language-independent, so you can implement Orbix applications in COBOL, PL/I, C++, or Java that interoperate, via the standard IIOP protocol, with applications built on any CORBA-compliant technology.

Open standard solution

CORBA is an open, standard solution for distributed object systems. You can use CORBA to describe your enterprise system in object-oriented terms, regardless of the platforms and technologies used to implement its different parts. CORBA objects communicate directly across a network, using standard protocols, regardless of the programming languages used to create objects or the operating systems and platforms on which the objects run.

Widely available solution

CORBA solutions are available for every common environment and are used to integrate applications written in C, C++, Java, Ada, Smalltalk, COBOL, and PL/I, COM, LISP, Python, and XML, running on embedded systems, PCs, UNIX hosts, and mainframes. CORBA objects running in these environments can cooperate seamlessly. Through COMet, IONA's dynamic

bridge between CORBA and COM, they can also interoperate with COM objects. CORBA offers an extensive infrastructure that supports all the features required by distributed business objects. This infrastructure includes important distributed services, such as transactions, messaging, and security.

CORBA Objects

In This Section

This subsection describes the most basic components of a CORBA system. It discusses the following topics:

- [Nature of abstract CORBA objects](#)
 - [Object references](#)
 - [IDL interfaces](#)
 - [Advantages of IDL](#)
-

Nature of abstract CORBA objects

A CORBA system provides distributed object capability between applications in a network. A *client* in a CORBA system is any program that invokes the services (or functions) of a CORBA object. A *server* in a CORBA system is any program that contains instances of *CORBA objects*.

CORBA objects are abstract objects in a CORBA system that provide distributed object capability between applications in a network. [Figure 1](#) shows that any part of a CORBA system can refer to the abstract CORBA object, but the object is only implemented in one place and time on some server within the system..

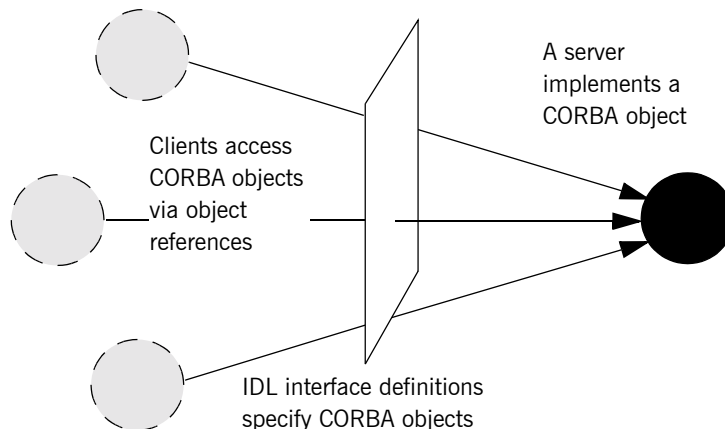


Figure 1: *The Nature of Abstract CORBA Objects*

Object references

An *object reference* is used to identify, locate, and address a CORBA object. Clients use an object reference to invoke requests on a CORBA object. CORBA objects can be implemented by servers in any supported programming language, such as COBOL, PL/I, C++, or Java. For integration with existing transactions in IMS, you can use the Orbix IMS server adapter to receive CORBA client requests and translate them to transaction invocations in IMS.

IDL interfaces

Although CORBA objects are implemented using standard programming languages, each CORBA object has a clearly-defined interface, specified in the *CORBA Interface Definition Language (IDL)*. The *interface definition* specifies which operations (member functions), data types, attributes, and exceptions are available to a client, without making any assumptions about an object's implementation. Not all IDL data types are supported by the IMS server adapter. Refer to [“Unsupported IDL Types” on page 35](#) for more information.

Advantages of IDL

With a few calls to an Object Request Broker's (ORB's) application programming interface (API), servers can make CORBA objects available to client programs in your network.

To call member functions on a CORBA object, a client programmer needs only to refer to the object's interface definition. Clients use their normal programming language syntax to call the member functions of a CORBA object. A client does not need to know which programming language implements the object, the object's location on the network, or the operating system in which the object exists.

Using an IDL interface to separate an object's use from its implementation has several advantages. For example, you can change the programming language in which an object is implemented without affecting the clients that access the object. You can also make existing objects available across a network.

The ORB

Overview

CORBA defines a standard architecture for object request brokers (ORBs). An ORB is a software component that mediates the transfer of messages from a program to an object located on a remote network host. The ORB hides the underlying complexity of network communications from the programmer.

This subsection discusses the following topics:

- [Role of an ORB](#)
- [Graphical overview](#)

Role of an ORB

An ORB lets you create standard software objects whose member functions can be invoked by *client* programs located anywhere in your network. A program that contains instances of CORBA objects is often known as a *server*. However, the same program can serve at different times as a client and a server. For example, a server program might itself invoke calls on other server programs, and so relate to them as a client.

Graphical overview

When a client invokes a member function on a CORBA object, the ORB intercepts the function call. As shown in [Figure 2](#), the ORB redirects the function call across the network to the target object. The ORB then collects results from the function call and returns these to the client.

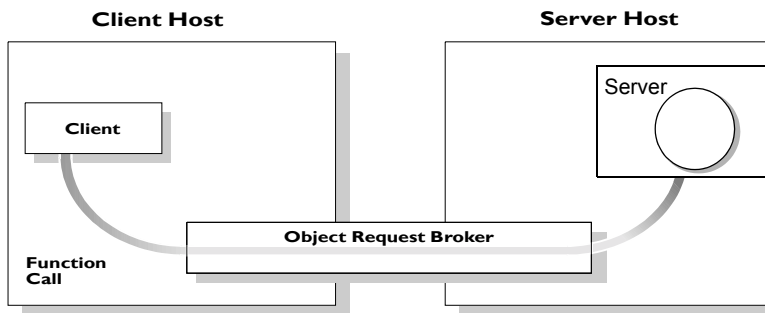


Figure 2: Role of the ORB in the Basic CORBA Model

CORBA Application Basics

In This Section

This subsection describes the basics of how CORBA applications work. It discusses the following topics:

- [Developing application interfaces](#)
 - [Client invocations on CORBA objects](#)
 - [IDL operation parameters](#)
 - [Parameter-passing mode qualifiers](#)
-

Developing application interfaces

The first step in developing a CORBA application is to define interfaces to objects in your system, in CORBA IDL. Then compile these interfaces with an IDL compiler. An IDL compiler can generate COBOL, PL/I, C++ or Java from IDL definitions. The generated code includes *client stub code* (excluding COBOL and PL/I), which you use to develop client programs; and *object skeleton code*, which you use to implement CORBA objects in server programs.

Note: With Orbix Mainframe, you can use the IDL compiler to generate only COBOL or PL/I server skeleton code from IDL definitions. The IDL compiler does not generate COBOL or PL/I client stub code.

Your installation of the IMS server adapter includes a server application that runs on OS/390 and acts as the CORBA gateway to the IMS system. Sample demonstrations are provided with the IMS server adapter installation program. These are located in the *orbixhlq.DEMOS.IMS* PDS. Samples of COBOL and PL/I IMS servers are provided. For more details about the COBOL demonstrations, refer to the section in the *COBOL Programmer's Guide and Reference* on developing an IMS server. For more details about the PL/I demonstrations, refer to the section in the *PL/I Programmer's Guide and Reference* on developing an IMS server.

Client invocations on CORBA objects

When a client wants to invoke operations on a CORBA object, it invokes on an object reference that it obtains from the server process. As shown in [Figure 3 on page 11](#), a client call is transferred through the client stub code to the ORB. The ORB then passes the function call through the object

skeleton code to the target object. Because the implemented object is not located in the client's address space, CORBA objects are represented in client code by *proxy objects*.

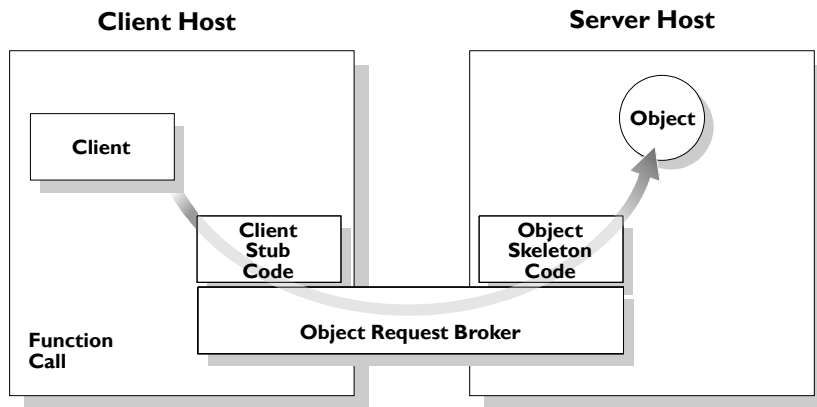


Figure 3: *Invoking on a CORBA Object*

IDL operation parameters

Each parameter specifies the direction in which its arguments are passed between client and object. Parameter-passing modes clarify operation definitions and allow the IDL compiler to accurately map operations to a target programming language. The Orbix IMS runtime uses parameter-passing modes to determine in which direction (or directions) it must marshal a parameter.

Parameter-passing mode qualifiers

There are three parameter-passing mode qualifiers:

- in** This means that the parameter is initialized only by the client and is passed to the object.
- out** This means that the parameter is initialized only by the object and is passed to the client.
- inout** This means that the parameter is initialized by the client and passed to the server; the server can modify the value before returning it to the client.

Overview of Orbix

Overview

Orbix is IONA's implementation of the CORBA standard. This section provides an example of a simple Orbix application and an introduction to the broader Orbix environment.

In this section

This section discusses the following topics:

Simple Orbix Application	page 13
Broader Orbix Environment	page 16

Simple Orbix Application

Overview

A simple Orbix application might contain a client and a server along with one or more objects (see [Figure 4](#)). In this model, the client obtains information about the object it seeks, using *object references*. An object reference uniquely identifies a local or remote object instance.

This subsection discusses the following topics:

- [Graphical overview](#)
 - [Explanation of simple application](#)
 - [Portable object adapter](#)
 - [Limitations of a simple application](#)
-

Graphical overview

[Figure 4](#) provides a graphical overview of a simple Orbix application.

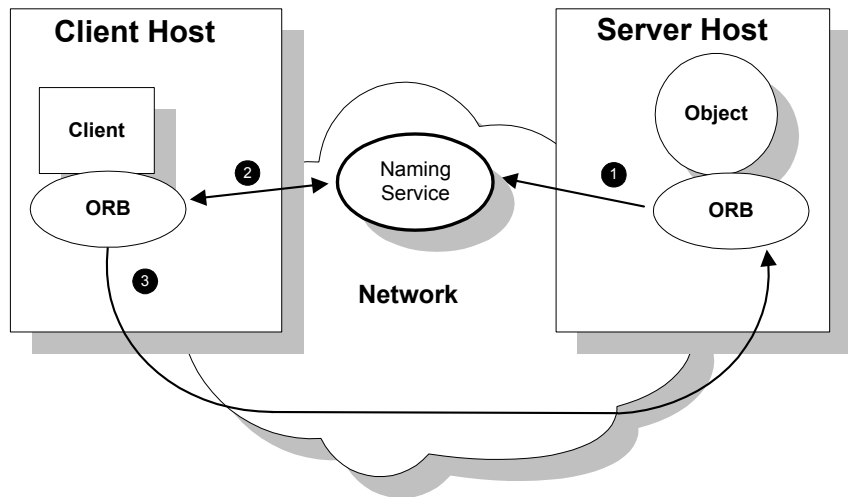


Figure 4: Overview of a Simple Orbix Application

Explanation of simple application [Figure 4 on page 13](#) shows how an ORB enables a client to invoke on a remote object:

Step	Action
1	When a server starts, it creates one or more objects and publishes their object references in a <i>naming service</i> . A naming service uses simple names to make object references accessible to prospective clients. Servers can also publish object references in a file or a URL.
2	The client program looks up the object reference by name in the naming service. The naming service returns the server's object reference.
3	The client ORB uses the object reference to pass a request to the server object.

Portable object adapter

For simplicity, [Figure 4 on page 13](#) omits details that all applications require. For example, Orbix applications use a Portable Object Adapter (POA), to manage access to server objects. A POA maps object references to their concrete implementations on the server. Given a client request for an object, a POA can invoke the referenced object locally.

The client request embeds the POA name and object ID taken from the published object reference. The server then uses the POA name to invoke the POA. The POA uses the object ID to invoke the desired object, if it exists on the server.

Refer to either the *COBOL Programmer's Guide and Reference* or the *PL/I Programmer's Guide and Reference* for details about the Orbix Mainframe POA.

Limitations of a simple application

This simple model uses a naming service to pass object references to clients. The naming service has some limitations and does not support all the needs of enterprise-level applications. For example, naming services are often not designed to handle frequent updates. They are designed to store relatively stable information that is not expected to change very often. If a process stops and restarts frequently, a new object reference must be published with each restart. In production environments where many

servers start and stop frequently, this can overwork a naming service. Enterprise applications also have other needs that are not met by this simple model—for example, on-demand activation, and centralized administration. These needs are met in a broader Orbix environment, as described in [“Broader Orbix Environment” on page 16](#).

Broader Orbix Environment

Overview

Along with the naming service, Orbix offers a number of features that are required by many distributed applications, for flexibility, scalability, and ease of use. This subsection provides an overview of these features. It discusses the following topics:

- [Location domains](#)
 - [Managing object availability](#)
 - [Configuration domains](#)
 - [Interface Repository](#)
-

Location domains

Location domains enable a server and its objects to move to a new process or host, and to be activated on demand. An Orbix location domain consists of two components—a locator daemon and a node daemon:

- locator daemon—This is a CORBA service that acts as the control center for the entire location domain. The locator daemon has two roles:
 - ◆ Manage the configuration information used to find, validate, and activate servers running in the location domain.
 - ◆ Act as the contact point for clients trying to invoke on servers in the domain.
- node daemon—This acts as the control point for a single host machine in the system. Every machine that runs an application server must run a node daemon. The node daemon starts, monitors, and manages application servers on its machine. The locator daemon relies on node daemons to start processes and tell it when new processes are available.

Managing object availability

A server makes itself available to clients by publishing Interoperable Object References (IORs). An IOR contains an object's identity and address. Refer to [“Sample configuration file” on page 215](#) for an example of an IOR.

When a client invokes on a object, Orbix locates the object as follows:

1. The ORB sends the invocation to the locator daemon.
 2. The locator daemon searches the implementation repository for the actual address of a server that runs this object.
 3. The locator daemon returns this address to the client.
 4. The client connects to the returned server address and directs this and all subsequent requests for this object to that address.
-

Configuration domains

Configuration domains allow you to organize ORBs into independently manageable groups. This brings scalability and ease of use to the largest environments.

Interface Repository

The *Interface Repository (IFR)* provides a source of type information, and allows clients to discover and use additional objects in the environment—even if clients do not know about these objects at compile time. Orbix Mainframe also supplies an alternative to using the IFR; refer to [“Using type_info store as a Source of Type Information” on page 202](#) for more information.

Introduction to the IMS Adapters

The Orbix Mainframe IMS server adapter provides a simple way to integrate distributed CORBA and EJB clients on various platforms with existing and new IMS transactions running on OS/390. It allows you to develop and deploy Orbix COBOL and Orbix PL/I servers in IMS, and to integrate these IMS servers with distributed CORBA clients running on various platforms. It also facilitates the integration of existing IMS transactions, not developed using Orbix, with distributed CORBA clients, without the need for code changes to these existing transactions. The IMS server adapter itself can execute in a native OS/390 or UNIX System Services address space.

The Orbix Mainframe client adapter provides a simple way for IMS transactions to act as clients of distributed CORBA servers on various platforms. It allows you to develop and deploy Orbix COBOL and Orbix PL/I clients in IMS. The client adapter itself can execute in a native OS/390 or UNIX System Services address space.

This chapter provides an introductory overview of both the IMS server adapter and the client adapter that are supplied with Orbix Mainframe.

In this chapter

This chapter discusses the following topics:

Overview of the IMS Server Adapter	page 21
Overview of the Client Adapter	page 36

Overview of the IMS Server Adapter

Overview

The IMS server adapter is an Orbix IONA Service that can be deployed in either a native OS/390 or UNIX System Services environment. Its function is to integrate distributed CORBA or EJB clients (or both) running on various platforms with existing or new IMS applications (or both) running on OS/390.

In This Section

This section discusses the following topics:

Role of the IMS Server Adapter	page 22
IMS Server Adapter Processing of IDL Operations	page 25
The IMS Server Adapter imsrw Interface	page 26
Unsupported IDL Types	page 35

Role of the IMS Server Adapter

Overview

The IMS server adapter acts as a bridge between CORBA/EJB clients and IMS servers. It allows you to set up a distributed system that combines the powerful online transaction processing capabilities of IMS with the consistent and well-defined structure of a CORBA environment.

This subsection discusses the following topics:

- [Characteristics of the IMS server adapter](#)
 - [IMS server adapter functions](#)
 - [Graphical overview](#)
 - [Graphical overview explanation](#)
-

Characteristics of the IMS server adapter

The IMS server adapter has the following characteristics:

- It is a fully dynamic bridge, because the interfaces that it provides to CORBA clients can be changed at runtime.
 - It is an Orbix server that is used to allow IMS transactions to process IDL-defined operations. Refer to [“IMS Server Adapter Processing of IDL Operations” on page 25](#) for more details.
 - It implements the `imsraw` IDL interface. Refer to [“The IMS Server Adapter `imsraw` Interface” on page 26](#) for more details.
-

IMS server adapter functions

The IMS server adapter performs the following functions:

1. It accepts an IDL request or a set of input message segments from the client.
2. It provides accepted IDL requests or input message segments to the IMS input message queue.
3. It runs the IMS transaction. If it is an IDL-based request, the server adapter marshals the operation parameters for the implementation server program in IMS, performing any necessary data conversion; otherwise, it simply runs the requested transaction with the supplied input message segments.
4. In the same way, it receives the results from IMS and returns them to the client.

Graphical overview

Figure 5 provides a graphical overview of the role of the IMS server adapter.

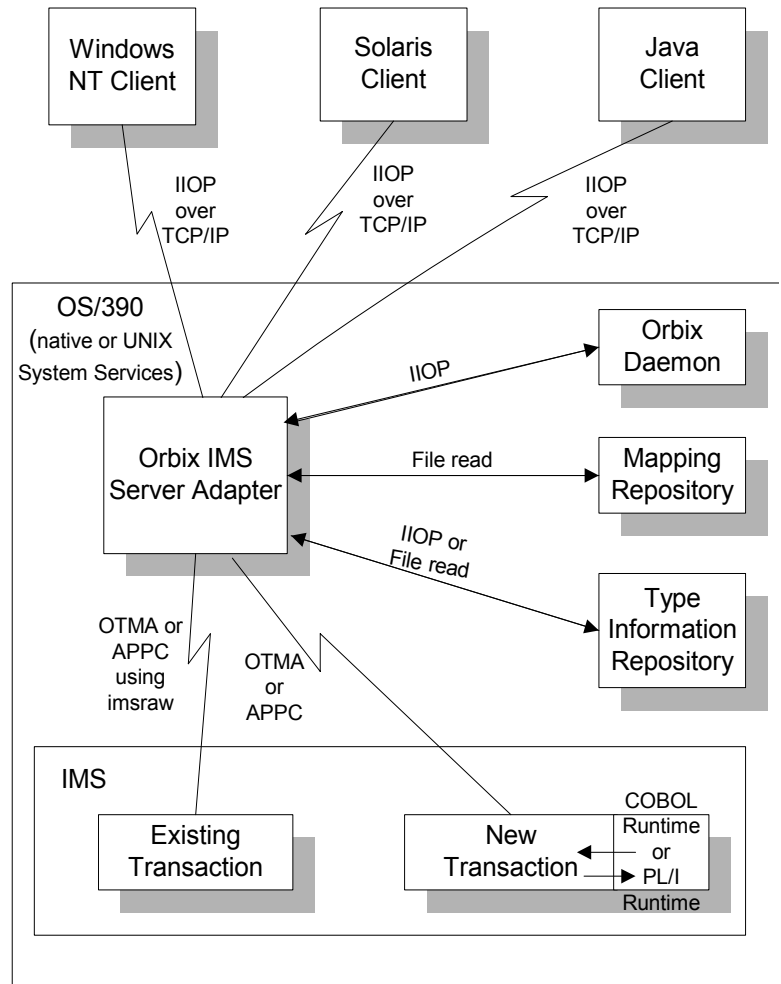


Figure 5: Graphical Overview of the Role of the IMS Server Adapter

Graphical overview explanation

[Figure 5 on page 23](#) provides an overview of the role of the IMS server adapter in integrating distributed CORBA or EJB clients (or both) on different platforms with IMS transactions running on OS/390. The CORBA or EJB clients can be written in languages such as C++ or Java.

The IMS server adapter communicates with IMS using either IBM's Open Transaction Manager Access (OTMA) or Advanced Program to Program Communications (APPC) protocol. As discussed, the IMS server adapter acts as a bridge between CORBA/EJB clients that can be running on various platforms and servers that are running in IMS.

IMS Server Adapter Processing of IDL Operations

Overview

The IMS server adapter is an Orbix server that allows IMS transactions to process IDL-defined operations. When the server adapter receives a request from a CORBA/EJB client, it looks up the appropriate IMS transaction name, based on the requested interface and operation name. The server adapter then marshals incoming data and submits the request to IMS with that transaction name. When the IMS transaction receives control via the normal IMS dispatching process, it uses the set of Orbix-provided services to read in the operation's parameters and marshal the return data, and returns the result to the client.

This subsection discusses the following topics:

- [List of required IDL interfaces](#)
- [IMS server adapter type information](#)

List of required IDL interfaces

The list of interfaces that the IMS server adapter needs to provide to its clients is provided to the server adapter in the form of a mapping file. Refer to [“The Mapping File” on page 186](#) for more details.

IMS server adapter type information

The IMS server adapter obtains IDL interface information (operation signatures) from either the IFR or from a `type_info` store, depending on the configuration values used. This enables the server adapter to unmarshal the data received from client programs and marshal the response back to the client. (Marshalling is the process whereby the communicated data is converted to a byte stream, so that it can be sent between the client and the server).

The exact manner in which information is loaded depends on the type information mechanism employed (that is, IFR or `type_info` store). Refer to [“Mapping IDL Interfaces to IMS” on page 185](#) for more information on these mechanisms.

The IMS Server Adapter `imsraw` Interface

Overview

This subsection provides an introductory overview of the `imsraw` IDL interface, which the IMS server adapter implements. It discusses the following topics:

- [What is the `imsraw` interface?](#)
- [Definition of the `imsraw` IDL](#)
- [Explanation of the `imsraw` IDL](#)
- [Demonstration of the `imsraw` interface](#)

What is the `imsraw` interface?

The IMS server adapter exposes a CORBA IDL interface, called `imsraw`, to its clients. The `imsraw` IDL interface defines operations to:

- Specify an IMS transaction name and set of input segments.
- Queue the transaction to IMS for dispatching.
- Receive the resulting output segments.

Note: If you used the previous versions of the IMS server adapter, the `imsraw` IDL interface has been modified to scope the `imsraw` interface inside a module called `IT_MFA_IMS`. However, to maintain backwards compatibility with older client applications, the IMS server adapter can be configured to expose the legacy unscoped `imsraw` API (see the *Mainframe Migration and Upgrade Guide* for more details). Also, as stated in the IDL of previous adapter versions, the `do_trans()` operation has been removed.

Definition of the `imsraw` IDL

The following shows the IDL definitions contained within the `imsraw` IDL interface:

Example 1: *The `imsraw` IDL Interface (Sheet 1 of 4)*

```

1 //IDL
2 #pragma prefix "iona.com"
3 module IT_MFA_IMS {
    interface imsraw {
        typedef string<8> tranName;
        typedef sequence<char> CharSegment;
        typedef sequence<CharSegment> CharSegments;
        typedef sequence<octet> ByteSegment;
    }
  }

```

Example 1: *The imsrw IDL Interface (Sheet 2 of 4)*

```

typedef sequence<ByteSegment> ByteSegments;

4  exception IMSunavailable { string reason; };
   exception unknownTransactionName {};
   exception segmentTooLarge {};
   exception userNotAuthorized { string reason; };
   exception transactionFailed { string reason; };
   exception internalError { string reason; };

   //
   // Methods for invoking IMS transactions.
   // The first uses CharSegments, so data is subject
   // to ASCII-EBCDIC conversion across platforms. The
   // second uses ByteSegments, so no conversion will be
   // done.
   //
5  CharSegments run_transaction(in tranName tran_name,
   in CharSegments din)
   raises(segmentTooLarge,
           IMSunavailable,
           unknownTransactionName,
           userNotAuthorized,
           transactionFailed,
           internalError);

5  ByteSegments run_transaction_binary(in tranName tran_name,
   in ByteSegments din)
   raises(segmentTooLarge,
           IMSunavailable,
           unknownTransactionName,
           userNotAuthorized,
           transactionFailed,
           internalError);

   //
   // Methods for invoking IMS transactions that do not
   // return a reply.
   // The first uses CharSegments, so data is subject
   // to ASCII-EBCDIC conversion across platforms. The
   // second uses ByteSegments so no conversion will be
   // done.

   // Methods run_transaction_no_reply() and
   // run_transaction_binary_no_reply() are only

```

Example 1: *The imsrw IDL Interface (Sheet 3 of 4)*

```

// supported in the IMS/APPC adapter.
//
6 CharSegments run_transaction_no_reply(in tranName tran_name,
    in CharSegments din)
    raises (segmentTooLarge,
           IMSunavailable,
           unknownTransactionName,
           userNotAuthorized,
           transactionFailed,
           internalError);

6 ByteSegments run_transaction_binary_no_reply(in tranName
    tran_name, in ByteSegments din)
    raises (segmentTooLarge,
           IMSunavailable,
           unknownTransactionName,
           transactionFailed,
           internalError);

7 readonly attribute unsigned long maxSegmentSize;
};

//
// Run conversational imsrw transactions
//
8 typedef sequence<octet> SessionHandle;

//
// Start the conversation in IMS/OTMA
// or IMS/APPC
//
9 void start_session(in tranName tran_name,
    out SessionHandle session)
    raises(internalError);

//
// Methods for invoking conversational IMS transactions.
// The first uses CharSegments, so data is subject
// to ASCII-EBCDIC conversion cross-platforms, the
// second uses ByteSegments so no conversion will be done.
//

10 CharSegments run_conv_transaction(
    in SessionHandle session,

```

Example 1: *The imsrw IDL Interface (Sheet 4 of 4)*

```

        in CharSegments din)
        raises(segmentTooLarge,
               IMSunavailable,
               unknownTransactionName,
               userNotAuthorized,
               transactionFailed,
               internalError);

10 ByteSegment run_conv_transaction_binary(
    in SessionHandle session,
    in ByteSegments din)
    raises(segmentTooLarge,
           unknownTransactionName,
           userNotAuthorized,
           transactionFailed,
           internalError);

// End the conversation in IMS/OTMA or IMS/APPC
//
11 void end_session(in SessionHandle session)
    raises(internalError);
};

```

Explanation of the imsrw IDL

The `imsrw` interface can be explained as follows:

- 1 This `pragma prefix` indicates that the IDL was developed by IONA.
- 2 The `imsrw` interface is within the `IT_MFA_IMS` module scope. The `IT_` prefix is a naming convention that is used to signify IDL modules developed by IONA Technologies. This helps to avoid naming clashes in the global scope.
- 3 It defines five data types:
 - ◆ `tranName`, which is a bounded string of up to eight characters.
 - ◆ `CharSegment`, which is a sequence of `char` types.
 - ◆ `CharSegments`, which is a sequence of `CharSegment` types.
 - ◆ `ByteSegment`, which is a sequence of `octet` types.
 - ◆ `ByteSegments`, which is a sequence of `ByteSegment` types.
- 4 It defines a series of exceptions that can be used to describe errors that might occur when running an IMS transaction. Any such errors are returned to the client, using this series of exceptions. This means that a client program can catch and handle any errors that might be used for diagnostic

purposes or for which a useful response is possible. See [“Exception information for APPC” on page 32](#) or [“Exception information for OTMA” on page 33](#) for more details of these exceptions.

- 5 It defines operations called `run_transaction()` and `run_transaction_binary()`. These operations are similar in that:
- ◆ They are both provided for passing message segments to a specified IMS transaction.
 - ◆ They both take two `in` parameters, called `tran_name` and `din`. The `tran_name` parameter specifies the IMS transaction that the client wants to invoke. The `din` parameter contains the message segments that the client wants to pass to the IMS transaction.

The two operations differ in the type of the `din` parameter and the return value. For example:

- ◆ The `din` parameter and return value for `run_transaction()` is of the `CharSegments` type. This means that the IMS server adapter performs ASCII-to-EBCDIC translations when it is sending the buffer that contains the message segments across different platforms.
 - ◆ The `din` parameter and return value for `run_transaction_binary()` is of the `ByteSegments` type. This means that the IMS server adapter passes the message segments intact to the IMS transaction, without translating them. The message segments are also passed intact from IMS back to the client via the IMS server adapter.
- 6 It defines operations called `run_transaction_no_reply()` and `run_transaction_binary_no_reply()`. These operations are similar in that:
- ◆ They are both provided for passing message segments to a specified IMS transaction.
 - ◆ They both take two `in` parameters, called `tran_name` and `din`. The `tran_name` parameter specifies the IMS transaction that the client wants to invoke. The `din` parameter contains the message segments that the client wants to pass to the IMS transaction.
 - ◆ They both return `void`. No reply data is expected from the IMS transaction.

- ◆ They both throw exceptions for some problems, such as if the specified transaction does not exist or if IMS is unavailable. Because the two operations do not receive a reply from IMS, they cannot report transaction results. The transaction might have completed without problems, it might be queued, or it might have ended abnormally (that is, abended).
- ◆ For the OTMA-based server adapter, IMS fast-path transactions, protected transactions, and conversational transactions cannot be used. Additionally, for the OTMA-based server adapter, a TPIPE (transaction pipe) must be configured, using the `plugins:ims_otma:xcf_tpipe_name` configuration item. See [“OTMA/IMS XCF TPIPE name” on page 68](#) for more details.

The two operations differ in the type of the `din` parameter. For example:

- ◆ The `din` parameter and return value for `run_transaction_no_reply()` is of the `CharSegments` type. This means that the IMS server adapter performs ASCII-to-EBCDIC translations when it is sending the buffer that contains the message segments across different platforms.
 - ◆ The `din` parameter for `run_transaction_binary_no_reply()` is of the `ByteSegments` type. This means that the IMS server adapter passes the message segments intact to the IMS transaction, without translating them.
- 7 The `readonly` attribute, `maxSegmentSize`, allows the client to retrieve the maximum segment length for which the IMS server adapter was configured when it was started. Because this is a `readonly` attribute, clients can read its value, but they cannot set it.
No changes are required to your IMS transaction.
 - 8 An IMS conversational transaction (that is, a program using a Save Program Area (SPA)) executes as a session. The `SessionHandle` data type is the handle used to indicate which calls belong to the same conversation.
 - 9 The `start_session` operation creates a conversation session with IMS and returns a handle to this conversation.

- 10 To navigate all the screens in the transaction, as many calls as necessary are made to the `run_conv_transaction()` or `run_conv_transaction_binary()` operation. One such call is necessary for each screen in the transaction, and each call is made with the handle returned by the `start_session` operation.
- The `run_conv_transaction()` and `run_conv_transaction_binary()` operations work in the same way as `run_transaction()` and `run_transaction_binary()` described in point 5. The only difference is that `run_conv_transaction()` and `run_conv_transaction_binary()` use a session name (rather than a transaction name) to indicate the conversational session being used.
- 11 When the conversation is finished, a call must be made to the `end_session()` operation, to free the session handle for the conversational transaction, and to release the resources associated with it in IMS and the Orbix IMS adapter.

Exception information for APPC

For APPC, the exception information that can be raised by the `imsraw` interface can be explained as follows:

- `reason`
The reason string is usually created from a call to `ATBEES3()`, with some other available information, such as the return code from the `ATBxxx` call, added where applicable. For failures that do not involve APPC, a reason string is generated by the adapter to describe the failure.
- `exception IMSunavailable { string reason; };`
An `IMSunavailable` exception is thrown when `ATBALC5()` fails with `k_badDestname`, `k_remoteLUnotActive`, or `k_remoteLUnotActive2`.
- `exception unknownTransactionName {};`
An `unknownTransactionName` exception is thrown when `ATBSEND()`, `ATBRCVW()`, or `ATBDEAL()` fails with `CM_TPN_NOT_RECOGNIZED`.
- `exception segmentTooLarge {};`
A `segmentTooLarge` exception is thrown if one of the input segments exceeds the maximum length specified for segments in the adapter configuration file.

- `exception userNotAuthorized { string reason; };`
A `userNotAuthorized` exception is thrown when `ATBSEND()`, `ATBRCVW()`, or `ATBDEAL()` fails with `CM_SECURITY_NOT_VALID`. It can also be thrown if the `plugins:imsa:use_client_principal` configuration item is set to "yes" but the principal received does not look like a valid RACF user ID.
- `exception transactionFailed { string reason; };`
A `transactionFailed` exception is thrown when `ATBSEND()` fails with `CM_PROGRAM_ERROR_NO_TRUNC`.
- `exception internalError { string reason; };`
An `internalError` exception is thrown for all other failures. Refer to the adapter event log output for more details on what caused a specific exception.

Exception information for OTMA

For OTMA, the exception information that can be raised by the `imsraw` interface can be explained as follows:

- `reason`
The reason string is usually created either from the error message that is returned by IMS over OTMA, or from OTMA return codes via the use of a look-up table for known return codes. For OTMA return codes that are not known to the adapter, the reason string contains the return and reason codes. For failures that do not involve OTMA, a reason string is generated by the adapter to describe the failure.
- `exception IMSunavailable { string reason; };`
An `IMSunavailable` exception is never thrown for OTMA, because the IMS server adapter cannot start in OTMA mode if IMS is not available.
- `exception unknownTransactionName {};`
An `unknownTransactionName` exception is thrown if an error message containing `DFS064` is returned from IMS along with return code 20. It can also be thrown if `otma_send_async()` returns with return code 8 and reason code 10.

- `exception segmentTooLarge {};`
 A `segmentTooLarge` exception is thrown if one of the input segments exceeds the maximum length specified for segments in the adapter configuration file. It can also be thrown for OTMA return code 8 with reason code 32.
- `exception userNotAuthorized { string reason; };`
 A `userNotAuthorized` exception is thrown if an error message containing `DFS1292E` is returned from IMS along with return code 20. It can also be thrown if the `plugins:imsa:use_client_principal` configuration item is set to "yes" but the principal received does not look like a valid RACF user ID.
- `exception transactionFailed { string reason; };`
 A `transactionFailed` exception is thrown for all OTMA failures relating to `otma_send_receive()` and `otma_send_async()`, with a return code 20, that are not covered by the other exceptions. The reason string is based on the error message returned by OTMA. It can also be thrown if a transaction is timed-out, or if RRS/OTS is used but the context switching for RRS fails.
- `exception internalError { string reason; };`
 An `internalError` exception is thrown for all other failures. Refer to the adapter event log output for more details on what caused a specific exception.

Demonstration of the `imsraw` interface

A C++ demonstration client for the `imsraw` interface is supplied with the other C++ demonstrations in your Orbix Mainframe installation. Follow the instructions in the supplied readme to run the client application.

Unsupported IDL Types

Overview

This subsection provides an overview of the IDL types that the IMS server adapter does not support.

Unsupported types

The following IDL types are not currently supported by the IMS server adapter:

- Object references.
- Value types, and other Pseudo-object types.
- wchar and wstring

Refer to the *COBOL Programmer's Guide and Reference* and the *PL/I Programmer's Guide and Reference* for details.

Overview of the Client Adapter

Overview

The Orbix Mainframe client adapter is an Orbix IONA Service that can be deployed in a native OS/390 or UNIX System Services environment. Its function is to allow IMS transactions to act as clients of CORBA servers running on various platforms.

The client adapter acts as a bridge between IMS client transactions and CORBA servers. The client adapter allows you to set up a distributed system that combines the powerful online transaction processing capabilities of IMS with the consistent and well-defined structure of a CORBA environment.

This section discusses the following topics:

- [Characteristics of the client adapter](#)
 - [Client adapter functions](#)
 - [Graphical overview](#)
 - [Graphical overview explanation](#)
-

Characteristics of the client adapter

The client adapter has the following characteristics:

- It is a mirror implementation of the IMS server adapter in that it adapts CORBA requests that originate in IMS, whereas the IMS server adapter adapts CORBA requests destined for IMS. [Figure 6 on page 38](#) provides an overview of the role of the client adapter in integrating IMS client transactions with distributed CORBA servers on different platforms.
- It implements the CORBA invocation facility via the Orbix Dynamic Invocation Interface (DII), and uses the IFR server to obtain type information. Refer to the *CORBA Programmer's Guide, C++* for more information on the DII.
- It provides an optional caching feature to improve performance. It can cache target object references and type information for operations.
- It is a multi-threaded application that can service multiple concurrent client requests.
- It can service multiple IMS regions.

Client adapter functions

The client adapter performs the following functions:

- It accepts a request from an IMS client transaction.
- It locates the target CORBA object and invokes the requested operation.
- It returns the CORBA object reply to the IMS client transaction.

Graphical overview

Figure 6 provides a graphical overview of the role of the client adapter.

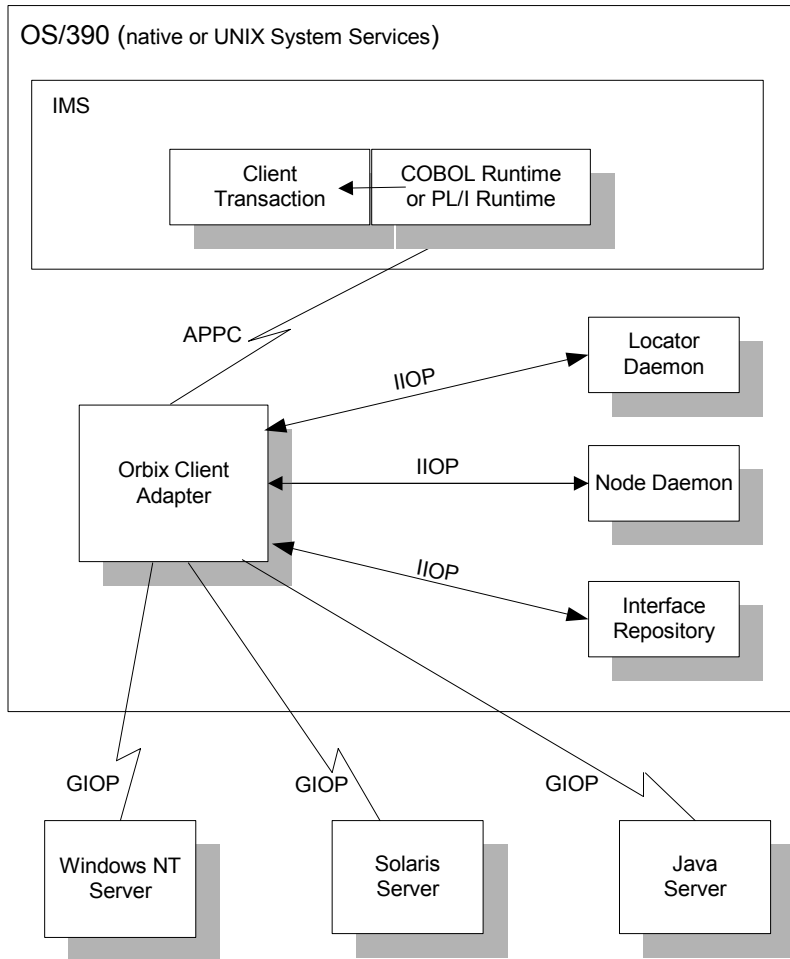


Figure 6: Graphical Overview of the Role of the Client Adapter

Graphical overview explanation

[Figure 6 on page 38](#) provides an overview of the role of the client adapter in integrating distributed CORBA servers on different platforms with IMS client transactions running on OS/390. The IMS client transactions can be written in COBOL or PL/I. The clients make a call to the COBOL or PL/I runtime that identifies both the target object and the operation to perform, and supplies `in`, `inout`, and `out` parameters. The COBOL or PL/I runtime uses the APPC protocol to communicate with the client adapter, and passes the client request to it. The client adapter locates the target server object and invokes the requested operation. The results are then returned back to the IMS client transaction.

Part 2

Configuring the IMS Server Adapter and the Orbix Runtime inside IMS

In this part

This part contains the following chapters:

Introduction to IMS Server Adapter Configuration	page 43
IMS Server Adapter Service Configuration Details	page 55
Configuring the IMS Server Adapter OTMA Plug-In	page 63
Configuring the IMS Server Adapter APPC Plug-In	page 71
Configuring the IMS Server Adapter RRS Plug-In	page 85
Configuring the IMS Server Adapter for Client Principals	page 95
Configuring the Orbix Runtime inside IMS	page 107
IDL Compiler Configuration	page 111

Introduction to IMS Server Adapter Configuration

This chapter provides information needed to configure the IMS server adapter and its components (plug-ins). It provides descriptions of all the configuration items involved in running the server adapter. It also provides details on configuring the various system components used by the server adapter. These components include IMS, OTMA, APPC/IMS, and RRMS.

In this chapter

This chapter discusses the following topics:

An IMS Server Adapter Sample Configuration	page 44
Configuration Summary of Adapter Plug-Ins	page 48

An IMS Server Adapter Sample Configuration

Overview

A sample configuration member is supplied with your Orbix Mainframe installation that provides an example of how you might configure and deploy the IMS server adapter on both native OS/390 and UNIX System Services.

This section discusses the following topics:

- [Location of configuration templates](#)
 - [Configuration scope](#)
 - [Configuration scope example](#)
-

Location of configuration templates

Sample configuration templates are supplied with your Orbix Mainframe installation in the following locations:

- Non-TLS template—`orbixhlq.CONFIG(FILETMPL)`
 - TLS template—`orbixhlq.CONFIG(TLSTMPL)`
-

Configuration scope

An ORBname of `iona_services.imsa` has been chosen for the IMS server adapter service. Therefore, the corresponding configuration items that are specific to the server adapter are scoped within an `iona_services.imsa` configuration scope.

Configuration scope example

The following is an example of the `iona_services.imsa` Configuration scope.

Example 2: *iona_services:imsa Configuration Scope Example (Sheet 1 of 3)*

```
iona_services
{
...
  imsa
  {
    event_log:filters = ["*=WARN+ERROR+FATAL", "IT_MFA=INFO_HI+WARN+ERROR+FATAL"];

    plugins:imsa:direct_persistence = "no";
    plugins:imsa:poa_prefix = "IT_MFA_IMS_";
  }
}
```

Example 2: *iona_services:imsa Configuration Scope Example (Sheet 2 of 3)*

```

#
# Settings for well-known addressing:
# (mandatory if direct_persistence is enabled)
#
# plugins:imsa:iop:port = "5005";
# plugins:imsa:iop:host = "%{LOCAL_HOSTNAME}";

# List of mappings of interface/operation -> IMS tran name
# PDS member or HFS filename may be specified
#
plugins:imsa:mapping_file = "DD:MFAMAPS";

# The adapter may be configured to use type_info files or
# to contact the IFR to attain type information dynamically
# during runtime.
#
# * To configure to use type_info files:
#   (note: source may be a PDS or HFS pathname)
#   plugins:imsa:repository_id   = "type_info";
#   plugins:imsa:type_info:source = "%{LOCAL_HFS_ROOT}/info.txt";
#
# * To configure to use the IFR:
#   plugins:imsa:repository_id   = "ifr";
#   plugins:imsa:ifr:cache       = "";
#
plugins:imsa:repository_id   = "ifr";
plugins:imsa:ifr:cache       = "";

# Activate this to display timing information on adapter requests
# plugins:imsa:display_timings = "yes";

# Activate this to display accounting info
# plugins:imsa:call_accounting_dll = "yes";

orb_plugins = ["iiop_profile", "giop", "iiop",
              "local_log_stream", "ots", "wto_announce"];

mf_subsystems = ["adapter"];

# choose IMS protocol plugin: ims_otma or ims_appc
#
initial_references:IT_imsraw:plugin = "ims_otma";
initial_references:IT_imsraw:plugin = "ims_appc";

```

Example 2: *iona_services:imsa Configuration Scope Example (Sheet 3 of 3)*

```

plugins:ims_otma:xcf_group_name = "IMSG";
plugins:ims_otma:xcf_adapter_member_name = "ORXIMSG";
plugins:ims_otma:xcf_ims_member_name = "IMS";
plugins:ims_otma:xcf_tpipe_prefix = "ORX1";
plugins:ims_otma:timeout = "30";
plugins:ims_otma:mq_length = "1024";
plugins:ims_otma:output_segment_num = "2";
plugins:ims_otma:xcf_tpipe_name = "ORXASYNC";

plugins:ims_appc:ims_destination_name = "ORBIXIMS";
plugins:ims_appc:appc_outbound_lu_name = "";
plugins:ims_appc:timeout = "30";
plugins:ims_appc:mq_length = "1024";

#
# For RRS/OTS support, add:
# plugins:rrs:rm_name = "TEST.IMSRAW.IONA.UA";
# initial_references:IT_RRS:plugin = "rrs";
#

#
# For client principal support, add/update:
# plugins:imsa:use_client_principal = "yes";
#
# And add the following if the client cannot send principals in a
# service context over GIOP 1.2 in a format recognised by the GIOP
# plugin
# policies:iiop:server_version_policy = "1.1";
#

# For the Adapter portable interceptor demo, update the
# orb_plugins setting as specified below, and add the
# the three settings that follow orb_plugins:
#
# orb_plugins = ["iiop_profile", "giop", "iiop",
#               "local_log_stream", "ots",
#               "demo_sec", "portable_interceptor" ];
# binding:server_binding_list = ["DemoPI"];
# plugins:demo_sec:shlib_name = "SECPI";
# plugins:demo_sec:shlib_version = "1";

...
};

```

Note: The configuration items shown in [Example 2](#) can be used to deploy an insecure server adapter. See [“Securing and Using the IMS Server Adapter” on page 161](#) for more details about the configuration items that are involved in deploying a server adapter in secure mode.

Configuring a domain

Refer to the *CORBA Administrator's Guide* for more details on how to configure an Orbix configuration domain.

Configuration Summary of Adapter Plug-Ins

Overview

Orbix configuration allows you to configure an application on a per-plug-in basis. This section provides a summary of the configuration items associated with plug-ins specific to the IMS server adapter.

This section discusses the following topics:

- [IMS server adapter plug-ins](#)
- [Summary of items for the imsa plug-in](#)
- [Summary of items for the ims_otma plug-in](#)
- [Summary of items for the ims_appc plug-in](#)
- [Summary of items for the rrs plug-in](#)
- [Summary of remaining configuration items](#)

Note: See [“Securing the IMS Server Adapter” on page 163](#) for more details about the items relating to the iS2 security plug-in.

IMS server adapter plug-ins

There are four plug-ins associated with the IMS server adapter:

- The `imsa` plug-in is the core IMS server adapter plug-in.
- The `ims_otma` plug-in is used specifically for communications with IMS over OTMA.
- The `ims_appc` plug-in is used specifically for communications with IMS over APPC.
- The `rrs` plug-in provides integration for the Object Transaction Service (OTS) and IMS commit processing. This plug-in is optional and can only be used if RRS is configured and RRS support in IMS is enabled. It can only be used with the `ims_otma` plug-in.

Note: Either the OTMA or APPC plug-in should be selected with the `initial_references:IT_imsraw:plugin` configuration variable.

Summary of items for the imsa plug-in

The following is a summary of the configuration items associated with the `imsa` plug-in. Refer to [“IMS Server Adapter Service Configuration Details” on page 55](#) for more details.

<code>iiop:port</code>	Specifies the TCP/IP port number that the IMS server adapter uses to listen for incoming requests. Valid values are in the range 1025–65535. This is an optional item.
<code>direct_persistence</code>	Specifies the persistence mode adopted by the IMS server adapter service. This is an optional item. <code>iiop:port</code> is required if this is specified as "yes".
<code>poa_prefix</code>	Specifies the POA prefix name. This is an optional item. The default value is <code>IT_MFA_</code> .
<code>iiop:host</code>	Specifies the host name that is contained in IORs exported by the IMS server adapter.
<code>mapping_file</code>	This file contains the mapping entries. Refer to “The Mapping File” on page 186 for more details. Optional.
<code>repository_id</code>	Specifies the type information source to use. This source supplies the IMS server adapter with operation signatures, as required. Valid values are <code>"ifr"</code> , <code>"type_info"</code> , and <code>"none"</code> . The default is <code>"ifr"</code> . Refer to “Type information mechanism” on page 59 for more information.
<code>ifr:cache</code>	This value is used if <code>repository_id</code> is set to <code>"ifr"</code> . The <code>ifr:cache</code> configuration item is optional, specifying the location of an (operation) signature cache file. This signature cache file contains a cache of operation signatures from a previous run of this server adapter. The default is no signature cache file (<code>"</code>).
<code>type_info:source</code>	This value is used if <code>repository_id</code> is set to <code>"type_info"</code> . The <code>type_info:source</code> variable denotes the location of a <code>type_info</code> store from which the server adapter can obtain operation signatures. Refer to “type_info store” on page 60 for more information.
<code>use_client_principal</code>	Indicates that the IMS server adapter should verify the client principal user ID with SAF before trying to start the target IMS transaction under that ID. The default is <code>no</code> .

<code>display_timings</code>	Specifies that the server adapter print timestamps showing how long it takes to process requests in IMS.
<code>call_accounting_dll</code>	If set to <code>yes</code> , this causes the accounting DLL to be loaded and accounting statistics to be displayed after each client request has been processed. The default is <code>no</code> . Refer to “Gathering Accounting Information in the Server Adapter” on page 244 for more details.
<code>capture_first_argument_in_dynany</code>	If set to <code>yes</code> , this passes the first argument of the request to the <code>IT_MFA_display_account_information()</code> function as a dynamic any. The default is <code>no</code> . Refer to “Gathering Accounting Information in the Server Adapter” on page 244 for more details.
<code>write_iors_to_file</code>	Specifies the file that should be used if you want the adapter to export object references to a file. If this configuration item is not included in the adapter’s configuration, no object references are exported to file. Refer to “Exporting Object References at Runtime” on page 250 for more details.
<code>write_iors_to_ns_context</code>	Specifies the Naming Service context that should be used if you want the adapter to export object references to a Naming Service context. If this configuration item is not included in the adapter’s configuration, no object references are exported to a Naming Service context. Refer to “Exporting Object References at Runtime” on page 250 for more details.
<code>publish_all_iors</code>	If set to <code>yes</code> , this instructs the adapter to export object references for the <code>MappingGateway</code> interface, the <code>imsraw</code> interface, and all interfaces specified in the adapter mapping file. If set to <code>no</code> , this instructs the adapter to export object references for the <code>MappingGateway</code> and <code>imsraw</code> interfaces only. The default is <code>no</code> . Refer to “Exporting Object References at Runtime” on page 250 for more details.

<code>remove_ns_iors_on_shutdown</code>	If set to <code>yes</code> , this instructs the adapter to unbind the object references from the Naming Service when shutting down normally. The default is <code>no</code> . Refer to “Exporting Object References at Runtime” on page 250 for more details.
<code>write_iors_to_ns_group_with_prefix</code>	Specifies the prefix that should be attached to each generated name indicating an interface, if you want the adapter to export object references to a Naming Service object group. This prefix is attached to the generated name, to specify the object group that is to be used. If this configuration item is not included in the adapter’s configuration, no object references are exported to any Naming Service object groups. Refer to “Exporting Object References at Runtime” on page 250 for more details.
<code>write_iors_to_ns_group_member_name</code>	Specifies the member name that the adapter should use in the object group. A unique member name must be specified for each adapter; otherwise, one adapter might end up replacing the object group members of another adapter. Refer to “Exporting Object References at Runtime” on page 250 for more details.

Summary of items for the `ims_otma` plug-in

The following is a summary of the configuration items associated with the `ims_otma` plug-in. Refer to [“OTMA Plug-In Configuration Items” on page 67](#) for more details.

<code>xcf_group_name</code>	Specifies the name of the Cross-Coupling Facility (XCF) group that you want the IMS server adapter to join. Default value is <code>IMSG</code> .
<code>xcf_adapter_member_name</code>	Specifies the member name automatically allocated to the IMS server adapter within the XCF group. Default value is <code>IONAIMS</code> .
<code>xcf_ims_member_name</code>	Specifies the IMS control region's member name in the XCF group. Default value is <code>IMS</code> .
<code>xcf_tpipe_prefix</code>	Specifies the 4-character prefix used for the name of the <code>TPIPE</code> opened between the IMS server adapter and the IMS region. Default value is <code>ORX1</code> .
<code>output_segment_num</code>	Specifies the number of initial output segments to be allocated by the IMS server adapter. The default value is 5.
<code>mq_length</code>	Specifies the maximum size, in bytes, of the data portion of a record on the IMS message queue. Default value is 32767.
<code>timeout</code>	Specifies the number of seconds that the IMS server adapter can wait for a response from IMS before cancelling the request. Default value is no timeout.
<code>use_sync_level_one</code>	Indicates whether OTMA calls are performed using OTMA Sync level 0 or Sync level 1. Default value is <code>true</code> .
<code>xcf_tpipe_name</code>	Specifies the <code>TPIPE</code> opened between the IMS server adapter and the IMS region for IMS transactions that do not return a reply to the client. Default value is <code>ORXASYNCR</code> .

Summary of items for the ims_appc plug-in

The following is a summary of the configuration items associated with the `ims_appc` plug-in. Refer to the [“APPC Plug-In Configuration Items” on page 83](#) for more details.

<code>ims_destination_name</code>	Specifies the APPC LU (Logical Unit) name for the IMS region to which the IMS server adapter connects. Default value is <code>ORBIXIMS</code> .
<code>appc_outbound_lu_name</code>	Specifies the IMS server adapter’s APPC LU name. The default value is <code>none</code> , which means that the system base LU is used.
<code>timeout</code>	Specifies the number of minutes that the IMS server adapter can wait for a response from IMS before cancelling the request. Default value is no timeout.
<code>mq_length</code>	Specifies the maximum size, in bytes, of the data portion of a record on the IMS message queue. Default value is <code>500</code> .

Summary of items for the rrs plug-in

The following is a summary of the configuration items associated with the `rrs` plug-in. Refer to [“RRS Plug-In Configuration Items” on page 94](#) for more details.

<code>rm_name</code>	The resource manager name that the IMS server adapter uses to register with RRS. Ensure that this variable is not specified in the configuration scope of the server adapter, if you do not want the RRS plug-in loaded.
<code>initial_references:IT_RRS:plugin</code>	Indicates to the IMS server adapter that it is the plug-in to load to enable communication with RRS. This is required if the <code>rrs</code> plug-in is used.

Summary of remaining configuration items

The following is a summary of the remaining configuration items. Refer to [“IMS Server Adapter Service Configuration Details” on page 55](#) and the *CORBA Administrator's Guide* for more details.

<code>thread_pool:initial_threads</code>	Specifies the initial number of threads that are created in the thread pool to send requests to IMS. This item is optional. The default value is 5.
<code>thread_pool:high_water_mark</code>	Specifies the maximum number of threads created in the IMS server adapter thread pool to send requests to IMS. This item is optional. Default value is -1.
<code>event_log:filters</code>	Specifies the types of events that the IMS server adapter logs.
<code>orb_plugins</code>	The list of standard ORB plug-ins the IMS server adapter should load.
<code>initial_references:IT_MFA:reference</code>	The IOR used by <code>itadmin</code> to contact the IMS server adapter—added to configuration after the server adapter has been run in prepare mode.

IMS Server Adapter Service Configuration Details

This chapter provides details of the configuration items for the IMS Server Adapter's application service plug-in. These items are used to specify parameters such as TCP/IP transport details, the level of Orbix event logging, and mapping information for mapping IDL operations to IMS transactions.

Overview

This chapter discusses the following topics:

- [Persistence mode](#)
- [Well known addressing](#)
- [IT_imsraw initial reference](#)
- [Orbix event logging](#)
- [ORB plug-ins list](#)
- [POA prefix](#)
- [Displaying transaction processing times](#)

- [Mapping file](#)
 - [Type information mechanism](#)
 - [IFR signature cache file](#)
 - [type_info store](#)
-

Persistence mode

The related configuration item is `plugins:imsa:direct_persistence`. It specifies the persistence mode policy adopted by the IMS server adapter. If you want the server adapter to run as a standalone service, set this to `yes`. If you set this to `no`, the server adapter contacts and registers with the locator service.

Well known addressing

Configuration items for well known addressing can be specified on the IIOp and secure IIOp plug-ins that are loaded by the IMS server adapter. For example, you can use `plugins:imsa:iioport` to specify a fixed TCP/IP port that the IMS server adapter uses to listen for insecure incoming CORBA requests. If the adapter is running with direct persistence enabled, the specified port number is published in the IORs generated by the adapter in prepare mode, and in any IORs returned by the `MappingGateway` interface.

Refer to [“Using the MappingGateway Interface” on page 222](#) for more details. If the adapter is running in indirect persistent mode, the locator’s addressing information is published in the IORs; however, in this case, the adapter still listens on the specified port.

The specified port number cannot be less than 1025, because the TCP/IP port numbers up to and including 1024 are reserved for TCP/IP services. Therefore, ensure that you do not use a port that is allocated to some other TCP/IP service on the machine. The server adapter checks to see if the port is available before it attempts to use it.

IT_imsraw initial reference

The related configuration item is `initial_references:IT_imsraw:plugin`. The `imsa` plug-in uses this configuration item to establish the name of the IMS transport-level plug-in to be loaded. To load the IMS OTMA plug-in, set this item to `ims_otma`. To load the IMS APPC plug-in, set this item to `ims_appc`.

This plug-in is used by the IMS server adapter service to communicate with IMS—it is therefore required for processing both the `imsraw` interface and mapped IDL interface requests. This item is required.

Orbix event logging

The related configuration item is `event_log:filters`. It is used in Orbix configuration to specify the level of event logging. To obtain events specific to the IMS server adapter, the `IT_MFA` event logging subsystem can be added to this list item. For example:

```
event_log:filters = [ "*"=WARN+ERROR+FATAL",  
  "IT_MFA=INFO_HI+INFO_MED+WARN+ERROR+FATAL" ];
```

This then logs all `IT_MFA` events (except for `INFO_LOW` — low priority informational events), and any warning, error, and fatal events from all other subsystems (for example, `IT_CORE`, `IT_GIOP`, and so on). The level of detail that is provided for `IT_MFA` events can therefore be controlled by setting the relevant logging levels. Refer to the *CORBA Administrator's Guide* for more details.

The following is a categorization of the informational events associated with the `IT_MFA` subsystem.

<code>INFO_HI</code>	configuration settings and IMS server adapter startup and shutdown messages
<code>INFO_MED</code>	mapping gateway actions and IMS OTMA/APPC calls, including return codes
<code>INFO_LOW</code>	IMS segment data streams and RRS actions

ORB plug-ins list

The related configuration item is `orb_plugins`. It specifies the ORB-level plug-ins that should be loaded in your application at `ORB_init()` time. On OS/390, you can add the WTO announce plug-in support to any Orbix application by updating this list in the relevant configuration scope. For example, in the `iona_services.imsa` scope:

```
orb_plugins = ["iiop_profile", "giop", "iiop",  
  "local_log_stream", "ots", "wto_announce"];
```

This causes the IMS server adapter to load the WTO announce plug-in and issues a WTO message when the server adapter ORB starts up and shuts down. Messages have the following format:

```
+ORX2001I ORB iona_services.imsa STARTED (HOSTNAME:<process id>)  
+ORX2002I ORB iona_services.imsa ENDED (HOSTNAME: <process id>)
```

On UNIX System Services, `<process id>` is a pid. On native OS/390, `<process id>` is a job name and an `A=xxxx` job identifier.

Additionally, add the OTS plug-in if RRS support is needed.

POA prefix

The related configuration item is `plugins:imsa:poa_prefix`. It specifies the prefix to be assigned to the POA name used by the IMS server adapter. The default value is `IT_MFA_`. This POA name is embedded in the object key of the IOR that is published by the server adapter in `prepare` mode, and obtained with `resolve` from the Mapping Gateway interface. The POA name is not significant in a server that runs in direct persistent mode; however, it can be useful for the purposes of keeping track of IORs in an environment where multiple IMS server adapters are being deployed.

Displaying transaction processing times

The related configuration item is `plugins:imsa:display_timings`. This configuration item is `no` by default. If you set this configuration item to `yes`, the server adapter produces output similar to the following:

```
Thread: 5, Operation: get, Processing completed: Fri Apr 12
09:59:22 2002
1: +0 ms, 2: +4 ms, 3: +99 ms, 4:+105 ms
```

Each item of output contains two lines. The first line shows the thread that dispatched the request, the name of the operation, and the date and time when the request was completed. The second line shows the timestamps at each of the four measurement points, in milliseconds. All timestamps are relative to the first measurement point. Therefore, the first measurement point always shows zero milliseconds.

The four measurement points taken are:

1. After the dispatching handler thread gets the request from the server adapter's pending request work queue.
2. Before sending the request to IMS.
3. After receiving the response from IMS.
4. Before sending the response back to the client, using IIOp.

The times measured do not include any time that the request has waited for a server adapter processing thread to become available. If you therefore have five threads in the server adapter, and send six requests at exactly the

same moment, the times displayed for the sixth request do not include the time it waited in the server adapter input queue for a thread to become available.

The first measurement point is taken before the data is marshalled from the IIOp request buffer, and is exactly the same point in the source code for each version of the server adapter.

The second and third measurement points are only approximately the same point in the source code for each version of the server adapter IMS transport (OTMA or APPC) plug-ins.

The fourth point is taken after the data has been marshalled back into the IIOp request buffer, but before it is transmitted to the client. It is also exactly the same point in the source code for each version of the server adapter.

No information is displayed for threads with IDs greater than 99. A mutex lock is taken around the two lines of printed output, so that they do not become garbled. The use of `plugins:imsa:display_timings` can cause a small decrease in the performance of server adapters, as opposed to when the server adapters are running without `plugins:imsa:display_timings`.

Mapping file

The related configuration item is `plugins:imsa:mapping_file`. You can use this to specify either a native OS/390 dataset name or a fully qualified pathname to an OS/390 UNIX System Services file. The contents of the specified file represent the mappings between IDL operations that the adapter supports and target IMS transaction names. The mapping file is read by the adapter when it starts. Refer to [“The Mapping File” on page 186](#) for more details.

Type information mechanism

The related configuration item is `plugins:imsa:repository_id`. It specifies the repository used by the IMS server adapter to store operation signatures. Two repositories are supported: IFR ("`ifr`") and `type_info` store ("`type_info`"). The default is `ifr`. Refer to [“Using type_info store as a Source of Type Information” on page 202](#) for more information on the role of type information. You can also set this item to `"none"`, to indicate that the adapter should only support `imsraw` and not attempt to read type information from anywhere.

IFR signature cache file

If the IMS server adapter is configured to use the IFR as the type information repository (a store of operation signatures), an IFR signature cache file can be used to improve performance. The related configuration item is `plugins:imsa:ifr:cache`. Refer to [“Using an IFR Signature Cache File” on page 200](#) for more information on how IFR signature cache files work.

The filename specification for the signature cache file can take one of several forms:

- The following example reads the mappings from a file in the OS/390 UNIX System Services hierarchical file system (HFS):

```
plugins:imsa:ifr:cache = "/home/user/sigcache.txt";
```

- The following example shows the syntax to indicate that the mappings should be read from a PDS:

```
plugins:imsa:ifr:cache = "//HLQ.DEMOS.IFRCACHE";
```

The data set is created with the default attributes used by the LE runtime. Depending on the number of interfaces and the complexity of the types used, this might not be large enough. In this case, the IMS server adapter saves as many cache entries as possible and then issues error messages. If this occurs, you should preallocate a larger data set with the same attributes, and use this name the next time you start the server adapter.

Note: Do not use members of partitioned data sets as a signature cache file.

type_info store

If the IMS server adapter is configured to use a `type_info` store as the type information repository (a store of operation signatures), the location of the store must be supplied. The related configuration item is `plugins:imsa:type_info:source`.

The `plugins:imsa:type_info:source` variable can be set to one of the following:

- An HFS file (OS/390 UNIX System Services)
Specifies a file to use as a `type_info` source. Operation signatures are read from this file during start-up. If a refresh is requested (via `itadmin mfa refresh` for example), this file is re-read. For example:

```
plugins:imsa:type_info:source = "/home/bob/type_info.txt";
```

- An HFS directory (OS/390 UNIX System Services)
Specifies a directory to use as a `type_info` source. Operation signatures are read from all files in this directory during start-up. If a refresh is requested, all files in the directory are browsed until the relevant operation signature(s) are found. For example:

```
plugins:imsa:type_info:source = "/home/bob/typeinfo_store";
```

- A PDS member (native OS/390)
Specifies a PDS member (batch) to use as a `type_info` source. Operation signatures are read from this member during start-up. If a refresh is requested, this member is re-read. For example:

```
plugins:imsa:type_info:source = "//MY1.TYPEINFO(MYINFS)";
```

- A PDS (native OS/390)
Specifies a dataset to use as a `type_info` source. Operation signatures are read from all member in this dataset during start-up. If a refresh is requested, all member in the dataset are browsed until the relevant operation signature(s) are found. For example:

```
plugins:imsa:type_info:source = "//MY1.TYPEINFO";
```

For PDS names, you can use a DD name, as long as this is defined to the IMS server adapter start JCL, `obrixhlq.ORBIX60.JCL(IMSA)`

Note: The use of HFS directories or a PDS is preferable to the use of flat files, because these methods are better suited to the dynamic addition or removal of interface information, and they can also address IDL versioning.

Configuring the IMS Server Adapter OTMA Plug-In

This chapter describes how to configure the IMS server adapter to use OTMA to communicate with IMS.

In this chapter

This chapter discusses the following topics:

Setting Up OTMA for the IMS Server Adapter	page 64
OTMA Plug-In Configuration Items	page 67

Setting Up OTMA for the IMS Server Adapter

Overview

This section describes the steps to set up OTMA for the IMS server adapter. It discusses the following topics:

- [Prerequisites to enabling OTMA for IMS](#)
 - [Activating OTMA for IMS](#)
 - [Further reading](#)
-

Prerequisites to enabling OTMA for IMS

To use the OTMA version of the IMS server adapter, OTMA and the OTMA C/I must be enabled for IMS. APARs provide all the binaries needed for the OTMA C/I to function on your IMS system for IMS version 6, and it comes with the base IMS install for IMS version 7. Also, ensure that all the latest OTMA and OTMA C/I APARs have been applied to your IMS system.

Activating OTMA for IMS

OTMA is activated by providing the following three parameters (PARM1) for the IMS proclib `DFSPBxxx` member (which starts the IMS control region):

OTMA=Y The `OTMA` parameter indicates whether OTMA should be activated at start-up. If you specify `N` for this parameter, but still provide the other two parameters, you can start OTMA in IMS with the following command:

```
/START OTMA
```

GRNAME=IMSG The `GRNAME` parameter provides the name of the XCF group that IMS creates or joins (or both). The name `IMSG` is provided as an example. You need to provide the relevant name for your site to the IMS server adapter, using the `xcf_group_name` configuration item.

OTMANM=IMS The `OTMANM` parameter specifies the name that IMS has within this XCF group. The name `IMS` is provided as an example. The member name for your site must be provided to the IMS server adapter, using the `xcf_ims_member_name` configuration item. You need to decide on names for the XCF group, IMS in the group, the server adapter member name(s), and the four-character `TPIPE` prefix to set up the RACF security needed by OTMA.

You can find the procedure to activate the OTMA C/I in the IBM publication *OTMA C/I, SC26-8743*. This guide lists the following steps:

Example 3: Steps to Activate the OTMA C/I

One of the OTMA C/I modules, DFSYSVC0, needs to be loaded and registered to the SVC services by an authorized address space running on the same OS/390 image as the application programs that will be accessing it.

OTMA Callable Services provides a stand-alone program, DFSYSVIO, that must be run after MVS IPL to initialize the OTMA C/I.

You must add an entry in the MVS program properties table (PPT) for the OTMA Callable Services initialization program. The steps for doing this are:

1. Edit the SCHEDxx member of the SYS1.PARMLIB data set.
2. Add the following entry to the SCHEDxx member:

```
PPT PGMNAME(DFSYSVIO)
      CANCEL
      KEY(7)
      SWAP
      NOPRIV
      DSI
      PASS
      SYST
      AFF(NONE)
      NOPREF
```

3. Take one of the following actions to make the SCHEDxx changes effective:

```
Re-IPL the MVS system.
or
Issue the "MVS SET SCH=" command.
```

RELATED READING: For additional reading about updating the program properties table, see *MVS/ESA Initialization and Tuning Reference*.

A sample JCL proc for running DFSYSVIO is as follows:

```
//OTMAINIT PROC RGN=300K,SOUT=A
//IEFPROC EXEC PGM=DFSYSVIO,
//          REGION=&RGN
//STEPLIB DD DISP=SHR,UNIT=SYSDA,
//          DSN=IMSVS.RESLIB
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
```

Ensure that the `OTMAINIT` procedure is re-executed after every IPL. If you get a SF92 abend when the OTMA-enabled IMS server adapter starts, this usually means the `OTMAINIT` job was not executed since the last IPL.

Further reading

Refer to the IBM publication *OTMA C/I, SC26-8743* to set up the RACF security for the resource `IMSXCF.OTMACI`.

You can find detailed documentation to activate OTMA for IMS in the IBM publication *Open Transaction Manager Access Guide and Reference, SC26-8743*.

Refer to the section on security in the IBM publication *OTMA reference, SC26-8743* for details on security-related questions.

OTMA Plug-In Configuration Items

In this section

This section discusses the following topics:

- [OTMA/IMS XCF group name](#)
 - [OTMA/IMS XCF IMS server adapter member name](#)
 - [OTMA/IMS XCF IMS member name](#)
 - [OTMA/IMS XCF TPIPE prefix name](#)
 - [OTMA/IMS XCF TPIPE name](#)
 - [OTMA/IMS transaction request timeout](#)
 - [Number of output segments allocated at startup](#)
 - [IMS message queue length](#)
 - [OTMA/IMS sync level one](#)
-

OTMA/IMS XCF group name

The related configuration item is `plugins:ims_otma:xcf_group_name`. It specifies the name of the XCF group that the IMS server adapter is to join. This must be the same as the value for the `GRNAME` parameter that is specified to the IMS control region when it is being started. This is the name displayed under the `GROUP` heading when the `/DIS OTMA` command is entered in IMS. If you do not specify a value for the XCF group name, the default is `IMSG`. [Example 4](#) illustrates how the IMS console might appear when you enter the `/DIS OTMA` command:

Example 4: Example of Output from the /DIS OTMA Command

```

/DIS OTMA

GROUP/MEMBER      XCF-STATUS      USER-STATUS      SECURITY
IMSG
- IMS              ACTIVE          SERVER            FULL
- IONAIMS         NOT DEFINED     DISCONNECTED

```

OTMA/IMS XCF IMS server adapter member name

The related configuration item is `plugins:ims_otma:xcf_adapter_member_name`. It specifies the member name that the IMS server adapter has in the XCF group. If you do not specify a member name, the default name is `IONAIMS`. In the preceding [Example 4](#),

the status of the server adapter in the XCF group is displayed as being disconnected. The name of the IMS server adapter is displayed on this list only if it has previously connected to the group.

OTMA/IMS XCF IMS member name

The related configuration item is `plugins:ims_otma:xcf_ims_member_name`. It specifies the IMS control region's member name in the XCF group. The IMS server adapter directs all requests to this member name. In the preceding [Example 4 on page 67](#) the member name is `IMS`. This means that when you enter the `/DIS OTMA` command in IMS, the member name is shown as the value in the `GROUP/MEMBER` column that corresponds to the value of `SERVER` in the `USER-STATUS` column. If you do not specify a value for the IMS member name, the default is `IMS`. The IMS member name specified must match the value for the `OTMANM` parameter relating to the IMS control region being connected to.

OTMA/IMS XCF TPIPE prefix name

The related configuration item is `plugins:ims_otma:xcf_tpipe_prefix`. It specifies the four-character prefix that is used for the name of the TPIPE that is opened between the IMS server adapter and the IMS region. The OTMA C/I generates the rest of the name internally. If you do not specify a value for the TPIPE prefix name, the default is `ORX1`.

OTMA/IMS XCF TPIPE name

The related configuration item is `plugins:ims_otma:xcf_tpipe_name`. It specifies the TPIPE name that is opened between the IMS server adapter and the IMS region. It is used for client requests to IMS transactions that do not return a reply message to the client. These are transactions initiated via calls to the `run_transaction_no_reply` and `run_transaction_binary_no_reply` operations in the `imsraw` interface. The value for this configuration item must be different from the value for the `plugins:ims_otma:xcf_tpipe_prefix` configuration item. It can be up to eight characters in length. If you do not specify a value for the TPIPE name, the default is `ORXASYN`.

OTMA/IMS transaction request timeout

The related configuration item is `plugins:ims_otma:timeout`. It specifies the number of seconds that the IMS server adapter waits for a response from IMS before cancelling the request, and prevents the server adapter from having to wait indefinitely for a response from IMS if the transaction has stopped for some reason. The default is no timeout.

Number of output segments allocated at startup

The related configuration item is `plugins:ims_otma:output_segment_num`. OTMA obtains more output segments in the IMS server adapter as it needs them. If IMS needs to resize the output area, it issues a User 119 abend to cancel the transaction, so the server adapter can re-issue the transaction with a larger output area. You can prevent this abend by allocating enough output segments when starting the server adapter for the size of the data that you expect back from IMS.

If a request asks for more output segments than are available, the increased number of output segments is also available to future requests on that thread.

IMS message queue length

The related configuration item is `plugins:ims_otma:mq_length`. The IMS server adapter forwards a request to IMS by placing data in segments onto the IMS message queue. This setting specifies how big each segment can be. If a data segment does not fit into a single IMS message queue dataset segment, IMS allows the segment to be spanned across multiple message queue records.

The best choice of IMS message queue length is usually at or just below 32KB, which is the limit for segment lengths. There are two distinct advantages in sending up to 32KB in each data segment:

- Sending the maximum limit in each data segment results in the least amount of wasted space in the IMS message queue. For big requests it means that each IMS message queue record is filled completely, except for the last one used for each segment. This is preferable than trying to match the message queue record length, because this value can be difficult to match exactly, resulting in a small amount of space being wasted in each record.
- Sending a couple of big segments is faster than sending a lot of small segments, because the communication overhead per segment is reduced in OTMA.

Setting a big value for `plugins:ims_otma:mq_length` does not cause any extra overhead for small requests, because the IMS server adapter only uses what it needs up to this maximum. For a small request, therefore, only the small message is transmitted between the adapter and IMS.

OTMA/IMS sync level one

The related configuration item is `plugins:ims_otma:use_sync_level_one`. It indicates whether OTMA calls are performed using OTMA Sync level 0 or Sync level 1. If Sync level 0 is used, the response times might be improved, but OTMA timeouts are ignored; also, if IMS returns more output to the adapter than the adapter expects, the extra output is lost. Refer to the IBM *IMS OTMA Guide* for more details on OTMA Sync levels. The use of Sync level 0 is desirable if shared message queues are used for IMS, to avoid the creation of cascaded RRS units of recovery. The default is `true`, to use Sync level 1.

Configuring the IMS Server Adapter APPC Plug-In

This chapter describes how to configure the IMS server adapter to use APPC to communicate with IMS.

In this chapter

This chapter discusses the following topics:

Setting Up APPC for the IMS Server Adapter	page 72
Additional RACF Customization Steps for APPC	page 80
APPC Plug-In Configuration Items	page 83

Setting Up APPC for the IMS Server Adapter

Prerequisites to using APPC

Before you can run an Orbix IMS application in your region, you must perform a number of additional steps to enable the required APPC functionality on your OS/390 system. Depending on your installation, one or all of these tasks might already have been completed.

Further reading

For more information on setting up APPC/MVS, refer to the IBM publication *MVS Planning: APPC/MVS Management, GC28-107*.

In addition, you can find specific information on the use of APPC by IMS in the chapter on “Administering APPC/IMS and LU 6.2 Devices” in *IMS/ESA Administration Guide: Transaction Manager, SC26-8104*.

In this section

This section discusses the following topics:

Defining LUs to APPC	page 73
Defining an APPC Destination Name for the IMS LU	page 75
Defining LUs to VTAM	page 77
Additional RACF Customization Steps for APPC	page 80

Defining LUs to APPC

Overview

An LU (Logical Unit) name identifies each side of an APPC conversation. It is defined to APPC/MVS in the APPCPM_{xxx} member of SYS1.PARMLIB. You must define at least two LU names to use the IMS server adapter: one for the IMS server adapter, and one for IMS.

This section discusses the following topics:

- [Associating an IMS LU with a specific IMS region](#)
 - [LU names and outbound-only communication](#)
 - [Specifying the APPC-side information dataset name](#)
 - [Using other IMS-on-APPC functions](#)
 - [Running multiple server adapters](#)
-

Associating an IMS LU with a specific IMS region

The IMS LU definition is associated with a specific IMS region by specifying the name of that region (IMSID from the IMSCTRL macro in the IMS system generation) as the transaction scheduler for the LU. For example:

```
LUADD ACBNAME(IMSLU01)
BASE
SCHED(IMS1)
```

LU names and outbound-only communication

The LU name to be used by the IMS server adapter is only used for outbound communication. It can therefore be specified as follows:

```
LUADD ACBNAME(ORXLU01)
NOSCHED
```

Specifying the APPC-side information dataset name

The only other requirement in SYS1.PARMLIB(APPCPM_{xxx}) is the specification of the name of the VSAM data set where APPC-side information can be found—for example, SIDEINFO DATASET(SYS1.APPCSI).

This data set is used to define APPC destination names. If your installation does not already have one, see SYS1.SAMPLIB(ATBSIVSM) for sample JCL to create one.

Using other IMS-on-APPC functions

Although this is all that is required for Orbix, other keywords might be needed if your system is using other IMS-on-APPC functions, such as initiating outbound conversations from within IMS.

Running multiple server adapters

If you want to run multiple server adapters, you might want to set up separate LUs for each one.

Defining an APPC Destination Name for the IMS LU

Overview

The IMS server adapter connects to an IMS region through an APPC destination name rather than directly through the IMS LU name. The APPC destination name is used to establish various default characteristics for the APPC conversation being initiated; including the name of the partner LU, the transaction program name, and a logon mode name.

This section discusses the following topics:

- [Storage of the APPC destination name](#)
- [Example of the APPC-side information dataset](#)
- [Explanation of example data set](#)

Storage of the APPC destination name

All this information is stored in the APPC-side information data set. This data set is updated using the `ATBSDFMU` APPC/MVS utility program.

Example of the APPC-side information dataset

The following is an example of an APPC-side information data set:

Example 5: *Example of an APPC-Side Information Data Set*

```
1 //SIADDEXEC PGM=ATBSDFMU
2 //SYSPRINT DD SYSOUT=*
3 //SYSSDLIB DD DSN=SYS1.APPCSI,DISP=SHR
4 //SYSSDOUT DD SYSOUT=*
//SYSIN DD DATA
SIADD
DESTNAME(ORBIXIMS)
TPNAME(DFSAPPC)
MODENAME(APPCHOST)
PARTNER_LU(IMSLU01)
/*
```

Explanation of example data set

The example APPC-side information data set can be explained as follows:

- 1 For the purposes of the IMS server adapter, `DESTNAME` is used to name the string that is to be passed to the server adapter when it is started.
- 2 The `TPNAME` specification is used to name an IMS transaction to run. However, this is overridden by the server adapter for each conversation. Therefore, its value here is not important.
- 3 The `MODENAME` parameter is used to name an entry in the VTAM logon mode table. This specifies other characteristics that are to be used in the conversation. See the `SYS1.SAMPLIB(ATBLMODE)` data set for a definition of the `APPCHOST` logon mode, and the `SYS1.SAMPLIB(ATBLJOB)` data set for the JCL to install it.
- 4 `PARTNER_LU` must specify the previously defined IMS LU.

Defining LUs to VTAM

Overview

APPC/MVS expects its LUs to be defined as VTAM resources, so that they can access a SNA network. This subsection discusses the following topics:

- [VTAM requirements for LUs](#)
- [Using SYS1.SAMPLIB\(ATBAPPL\)](#)
- [APPC definition parameter security requirements](#)

VTAM requirements for LUs

Although the IMS server adapter is usually run on the same system as the IMS region with which it communicates (that is, an LU=LOCAL conversation), VTAM application program definition (APPL) macros must still be coded for each LU. See SYS1.SAMPLIB(ATBAPPL) for a sample APPL definition of an APPC LU.

Using SYS1.SAMPLIB(ATBAPPL)

The following definitions for the IMS and IMS server adapter LUs use the SYS1.SAMPLIB(ATBAPPL) definition, with some changes (which are highlighted):

Example 6: *Example of APPL Definitions for IMS and IMS Server Adapter LUs (Sheet 1 of 2)*

```

1  IMSLU01 APPL ACBNAME=IMSLU01,      C
   APPC=YES,                          C
   SECACPT=CONV,                       C
3  VERIFY=OPTIONAL,                   C
   AUTOSES=0,                          C
   DDRAINL=NALLOW,                    C
   DLOGMOD=APPCHOST,                   C
   DMINWNL=5,                          C
   DMINWNR=5,                          C
   DRESPL=NALLOW,                     C
   DSESLIM=10,                         C
   LMDENT=19,                          C
   MODETAB=LOGMODES,                  C
   PARSESS=YES,                       C
   SRBEXIST=YES,                      C
   VPACING=1
1  ORXLU01 APPLACBNAME=ORXLU01,      C
   APPC=YES,                          C
   SECACPT=CONV,                       C

```

Example 6: Example of APPL Definitions for IMS and IMS Server Adapter LUs (Sheet 2 of 2)

```

3  VERIFY=OPTIONAL,      C
   AUTOSES=0,           C
   DDRAINL=NALLOW,     C
   DLOGMOD=APPCHOST,    C
   DMINWNL=5,           C
   DMINWNR=5,           C
   DRESPL=NALLOW,      C
   DSESLIM=10,          C
   LMDENT=19,           C
   MODETAB=LOGMODES,   C
   PARSESS=YES,         C
   SRBEXIST=YES,        C
   PACING=1

```

APPC definition parameter security requirements

The following requirements exist:

- 1 Both the `ACBNAME=` parameter and the `APPL` statement label should match the LU name defined to APPC.
- 2 The `SECACPT=` and `VERIFY=` parameters specify which authentication and access checks are made when initiating conversations between the two LUs. Because both sides of an APPC conversation must agree on the level of conversation security to use, it is important that both LU definitions specify the same values for these two parameters.

`SECACPT=CONV` indicates that a partner LU must provide user and password information to authenticate itself before being allowed access to resources on the local system. This protects your IMS region from unauthorized access by users on other systems in your SNA network.

- 3 `VERIFY=OPTIONAL` indicates that the password requirement can be bypassed if LU-LU session-level verification can be performed. This allows the server adapter to get access (via the session keys in the APPC-LU profiles described in [“APPCLU profile contents and operation” on page 81](#)) to the IMS region without having to know the passwords of all its clients.

If there is no possibility of unauthorized access from other systems in your SNA network, you might prefer to code `SECACPT=ALREADYV` and `VERIFY=NONE` to indicate that partner LUs do not need to be authenticated. This is safe for `LU=LOCAL` conversations because user information is provided directly by `APPC/MVS`. Therefore, there is no opportunity for the programmers of the

partner LU to fabricate his identity. Refer to [“Securing the IMS Server Adapter” on page 163](#) for more details about APPC conversation security and session-level verification.

Additional RACF Customization Steps for APPC

Overview

There are a number of RACF definitions related to APPC that you might need to add or change to run the IMS server adapter. Refer to [“Securing the IMS Server Adapter” on page 163](#) for more details about how the server adapter fits into a secure system environment.

Much of the information provided in this section can be found in the sections relating to LU Security and Conversation Security in the Setting up Network Security chapter in the IBM publication *MVS Planning: APPC/MVS Management, GC28-1807*.

This section discusses the following topics:

- [Partner LUs and user ID requirements](#)
- [Bypassing partner LU user ID and password requirements](#)
- [APPCLU profile contents and operation](#)
- [Accessing APPCLU profiles](#)
- [Controlling access to APPCLU profiles](#)
- [Enabling APPC/IMS](#)

Partner LUs and user ID requirements

If you have defined the IMS LU to VTAM as having `SECACPT=CONV` and `VERIFY=OPTIONAL`, partner LUs that initiate a conversation must provide a user ID and password to authenticate themselves to the IMS LU. This ensures that the IMS transactions being submitted over the conversation can run under that user ID. The IMS server adapter does not have passwords for all its clients, so it cannot meet this requirement directly. If you are running OS/390 V1R3 or later, this option is enforced.

Bypassing partner LU user ID and password requirements

You can bypass this requirement by defining two RACF `APPCLU` profiles with a shared *session key* that essentially acts as a password replacement for conversations between the LUs named in the profiles.

Each `APPCLU` profile name has the form:

`'networkid.local-lu-name.partner-lu-name'` and contains information to be used by APPC/MVS on one side of a conversation between the two

named LUs. This means each side of a conversation has its own specific profile. For example, if LU `ORXLU01` initiates a conversation with LU `IMSLU01`, APPC/MVS on the initiating (outbound) side examines the `'networkid.ORXLU01.IMSLU01'` profile, and APPC/MVS on the receiving (inbound) side examines the `'networkid.IMSLU01.ORXLU01'` profile.

APPCLU profile contents and operation

Each profile contains a session key, which is a string of letters or numbers, and a `CONVSEC` setting. When a conversation is initiated between these two LUs, APPC/MVS on the outbound side passes the session key found in its profile to APPC/MVS on the inbound side. If APPC/MVS on the inbound side finds that the received session key matches the session key in its own profile, it overrides the `VTAM SECACPT=` setting with the `CONVSEC` setting from its profile. Thus, to allow the IMS server adapter to authenticate itself to IMS without passwords, the following definitions might be used:

```
RDEFINE APPCLU P390.ORXLU01.IMSLU01
UACC(NONE) SESSION(SESSION(137811C0) CONVSEC(ALREADYV))
RDEFINE APPCLU P390.IMSLU1.ORXLU01
UACC(NONE) SESSION(SESSION(137811C0) CONVSEC(ALREADYV))

SETROPTS CLASSACT(APPCLU)
```

Accessing APPCLU profiles

It is not necessary to permit the IMS server adapter or IMS region to have user IDs for the `APPCLU` profiles. However, access to the profiles should be tightly controlled to ensure that only appropriate users can read or change the session keys.

If you have set up the `APPCLU` profiles that allow a conversation between two specific LU names to bypass password-checking, you should limit the users that can initiate or receive conversations using those LU names.

Controlling access to APPCLU profiles

You can control access to `APPCLU` profiles by creating RACF `APPCPORT` profiles for each LU name and by permitting only certain users access to those profiles. For example:

```
RDEFINE APPCPORT IMSLU01 UACC(NONE)
PERMIT IMSLU01 CLASS(APPCPORT) ID(IMS1) ACCESS(READ)

RDEFINE APPCPORT ORXLU01 UACC(NONE)
PERMIT ORXLU01 CLASS(APPCPORT) ID(Adapter) ACCESS(READ)
```

```
SETROPTS CLASSACT(APPCPORT) RACLIST(APPCPORT)
```

By having an `ORXLU01` profile, you are restricting the users that can take advantage of the session-level verification provided by the `APPCLU` profiles. By having an `IMSLU01` profile, you are preventing users from being able to masquerade as an IMS region.

You might also want to be able to completely disallow a connection to the IMS LU on a per-user basis. For example, if a user initiates an APPC conversation with the IMS LU (either by using the IMS server adapter or a custom APPC program) from an LU for which no `APPCLU` profiles exist, and `SECACPT=CONV` is coded on the VTAM ACB for the IMS LU, users cannot be authenticated unless they provide a password. However, this does not prevent the conversation from being initiated; it simply means the transaction runs under no user. (This is known as a `security_none` conversation.)

If you want to prevent such connections, you can create a RACF `APPL` profile for the IMS LU name, and only grant access to specific users. For example:

```
RDEFINE APPL IMSLU01 UACC(NONE)
PERMIT IMSLU01 CLASS(APPL) ID(Adapter) ACCESS(READ)

SETROPTS CLASSACT(APPL) RACLIST(APPL)
```

Enabling APPC/IMS

To enable APPC/IMS, specify `APPC=Y` in the start-up parameters, or enter the following command on a running system:

```
/START APPC
```

To enable APPC/IMS security, specify `APPCSE=F` at start-up, or enter the following command:

```
/SECURE APPC FULL
```

APPC Plug-In Configuration Items

Overview

This section provides a detailed description of the APPC plug-in configuration items. It discusses the following topics:

- [IMS APPC destination LU name](#)
 - [Server Adapter outbound LU name](#)
 - [APPC/IMS transaction request timeout](#)
 - [IMS message queue length](#)
-

IMS APPC destination LU name

The related configuration item is `plugins:ims_appc:ims_destination_name`. This specifies the APPC LU name for the IMS region to which the IMS server adapter connects. All incoming client requests are forwarded into the specific IMS region that is associated with this destination name. The default value is `ORBIXIMS`.

The specified APPC destination name is verified only when the server adapter first attempts to issue a request to the specified IMS region. This means that the IMS region does not have to be available when you start the APPC-based adapter.

Server Adapter outbound LU name

The related configuration item is `plugins:ims_appc:appc_outbound_lu_name`. This specifies the APPC LU name that the server adapter uses to initiate communication with IMS. This is useful when security considerations prohibit APPC connections between the system base LU and IMS. Refer to [“APPC-Based Security Considerations” on page 171](#) for more details. Refer to [“Defining LUs to APPC” on page 73](#) for an example where the LU name is created as `ORXLU01`.

APPC/IMS transaction request timeout

The related configuration item is `plugins:ims_appc:timeout`. It specifies the number of minutes that the IMS server adapter waits for a response from IMS before cancelling the request. It prevents the server adapter from having to wait indefinitely for a response from IMS if the transaction has stopped for some reason. The default is no timeout.

IMS message queue length

The related configuration item is `plugins:ims_appc:mq_length`. The IMS server adapter forwards a request to IMS by placing data in segments onto the IMS message queue. This setting specifies how big each segment can be. If a data segment does not fit into a single IMS message queue dataset segment, IMS allows the segment to be spanned across multiple message queue records.

The best choice of IMS message queue length is usually at or just below 32K, which is the limit for segment lengths. There are two distinct advantages in sending up to 32K in each data segment:

- Sending the maximum limit in each data segment results in the least amount of wasted space in the IMS message queue. For big requests it means that each IMS message queue record is filled completely, except for the last one used for each segment. This is preferable than trying to match the message queue record length, because this value can be difficult to match exactly, resulting in a small amount of space being wasted in each record.
- Sending a couple of big segments is faster than sending a lot of small segments, because the communication overhead per segment is reduced in APPC.

Setting a big value for `plugins:ims_appc:mq_length` does not cause any extra overhead for small requests, because the IMS server adapter only uses what it needs up to this maximum. For a small request, therefore, only the small message is transmitted between the adapter and IMS.

Configuring the IMS Server Adapter RRS Plug-In

The RRS plug-in provides integration facilities between the CORBA OTS service in the IMS server adapter and the commit/rollback processing of IMS. This chapter provides an introduction to RRS functionality, shows you how to set up RRS for the IMS server adapter, and provides details of the RRS plug-in configuration items.

In this chapter

This chapter discusses the following topics:

Introduction to RRS	page 86
Setting up RRS for the IMS Server Adapter	page 87
RRS Plug-In Configuration Items	page 94

Introduction to RRS

RRS plug-in functionality

This plug-in can only be used in conjunction with the OTMA transport plug-in. The RRS plug-in only becomes involved in the request if the client sends the request with a transaction context. The server adapter therefore supports both transactional and non-transactional requests when the RRS plug-in is enabled. The transactional performance overheads only affect transactional requests. With RRS support, the server adapter only commits or rolls back transactions in IMS when the client program issues the commit or rollback call for a transactional request.

This section discusses the following topics:

- [IORs and transaction support](#)
 - [Further reading](#)
-

IORs and transaction support

IORs for IDL interfaces that support transactional processing have an extra component to indicate to the client that transactional support is available in the server (the server adapter in this case). Ensure that you obtain new IORs from the IMS server adapter, using prepare and resolve, and so on, after you have enabled the RRS plug-in. This is because transactional communication between the client program and the server adapter only works with these new IORs with the transaction support component.

Further reading

For further information, refer to the IBM publication *OS/390 MVS Setting up a Sysplex*, GC28-1779.

Further information about System Logger is available in the IBM publication *OS/390 MVS Setting up a Sysplex*, GC28-1779.

Setting up RRS for the IMS Server Adapter

In this section

This section describes what you need to do to use the RRS plug-in with the IMS server adapter. It discusses the following topics:

- [IPL your OS/390 system in Sysplex mode](#)
 - [Defining the required log streams](#)
 - [Managing log streams](#)
 - [Starting RRS](#)
 - [Stopping RRS](#)
 - [Restarting IMS when RRS is available on the system](#)
-

IPL your OS/390 system in Sysplex mode

RRS requires the use of a sysplex couple data set, which means that your OS/390 system must be configured as part of a single-system or multi-system sysplex.

The following steps are required.

Step	Action
1	Change the PLEXCFG parameter in SYS1.PARMLIB(IEASYSXX) to PLEXCFG=MONOPLEX for a single-system sysplex or PLEXCFG=MULTISYSTEM for a multi-system sysplex. PLEXCFG=ANY is also valid.
2	Specify COUPLExx in SYS1.PARMLIB(IEASYSxx) to identify the COUPLExx parmlib member that describes the sysplex environment.

Step	Action
3	<p>Use the XCF couple dataset format utility (IXCL1DSU) to create and format all sysplex couple data sets before IPLing a system that is to use them. The following JCL can be used:</p> <pre>//STEP1 EXEC PGM=IXCL1DSU //STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR //SYSPRINT DD SYSOUT=A //SYSIN DD * DEFINEDS SYSPLEX(IONAPLEX) DSN(SYS1.XCF.CDS01) VOLSER(S27VL1) MAXSYSTEM(8) CATALOG DATA TYPE(SYSPLEX) ITEM NAME(GROUP) NUMBER(50) ITEM NAME(MEMBER) NUMBER(120) ITEM NAME(GRS) NUMBER(1) DEFINEDS SYSPLEX(IONAPLEX) DSN(SYS1.XCF.CDS02) VOLSER(S27VL2) MAXSYSTEM(8) CATALOG DATA TYPE(SYSPLEX) ITEM NAME(GROUP) NUMBER(50) ITEM NAME(MEMBER) NUMBER(120) ITEM NAME(GRS) NUMBER(1) /*</pre>
4	<p>Create a COUPLE_{xxx} member in SYS1.PARMLIB that includes the couple data sets you have just defined; for example:</p> <pre>COUPLE SYSPLEX(IONAPLEX) PCOUPLE(SYS1.XCF.CDS01) ACOUPLE(SYS1.XCF.CDS02)</pre>
5	<p>IPL your system for the above changes to take effect.</p>

Defining the required log streams

There are two types of log streams:

- Coupling facility log streams.
- DASD-only log streams.

The main difference between the two types of log streams is the storage medium used to hold interim log data. In a coupling facility log stream, interim storage for log data is contained in coupling facility list structures. In

a DASD-only log stream, interim storage for log data is contained in local storage buffers on the system. For the purposes of this demonstration, DASD-only log streams are used.

Prerequisites to running the log streams

RRS requires five log streams to be defined to System Logger. The IBM publication *OS/390 MVS Programming: Resource Recovery, GC28-1739* lists the following initial and recommended sizes for the log streams:

Table 1: *Initial and Maximum Log Stream Sizes*

Log Stream	Initial Size	Maximum Size
RM.Data	1 MB	1 MB
MAIN.UR	5 MB	50 MB
DELAYED.UR	5 MB	50 MB
RESTART	1 MB	5 MB
ARCHIVE	5 MB	50 MB

The initial sizes listed should be sufficient to run the demonstration, but the log streams should be set up with the maximum sizes, if possible, to facilitate future use of RRS on the system. This is because production-level applications require the maximum sizes listed. Also, the `ARCHIVE` stream is not required, but setting it up could help to trace any problems with RRS later on.

Managing log streams

Log streams are managed based on the policy information that is placed in the LOGR couple data set. To do this perform the following steps.

Step	Action
1	<p>Create and format the LOGR couple data set. The following JCL can be used:</p> <pre>//STEP1 EXEC PGM=IXCL1DSU //STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR //SYSPRINT DD SYSOUT=* //SYSIN DD * DEFINEDS SYSPLEX(IONAPLEX) DSN(SYS1.SLC.FDSS1) VOLSER(S27VL1) DATA TYPE(LOGR) ITEM NAME(LSR) NUMBER(100) ITEM NAME(LSTRR) NUMBER(50) ITEM NAME(DSEXTENT) NUMBER(20) DEFINEDS SYSPLEX(IONAPLEX) DSN(SYS1.SLC.FDSS2) VOLSER(S27VL2) DATA TYPE(LOGR) ITEM NAME(LSR) NUMBER(100) ITEM NAME(LSTRR) NUMBER(50) ITEM NAME(DSEXTENT) NUMBER(20) /*</pre>
2	<p>Update the SYS1.PARMLIB(COUPLExx) member to include the LOGR data sets you have just defined. For example:</p> <pre>DATA TYPE(LOGR) PCOUPLE(SYS1.SLC.FDSS1) ACOUPLE(SYS1.SLC.FDSS2)</pre>

Step	Action
3	<p>Make the LOGR couple data sets available. You can use either of the following ways to make the LOGR datasets available to the system:</p> <ul style="list-style-type: none"><li data-bbox="638 395 1135 453">• IPL the system to activate the newly defined specifications in the COUPLxx member.<li data-bbox="638 465 1247 522">• Issue the following SETXCF operator commands to bring the LOGR data sets online without an IPL: <pre data-bbox="687 545 1247 565">SETXCF COUPLE,TYPE=LOGR,PCOUPLE=(SYS1.SLC.FDSS1)</pre><pre data-bbox="687 586 1247 607">SETXCF COUPLE,TYPE=LOGR,ACOUPLE=(SYS1.SLC.FDSS2)</pre>

Step	Action
4	<p>Define the log streams, using the IXCMIAPU utility provided in SYS1.MIGLIB. The following JCL can be used:</p> <pre> //STEP1 EXEC PGM=IXCMIAPU //STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR //SYSPRINT DD SYSOUT=* //SYSIN DD * DATA TYPE(LOGR) REPORT(YES) DEFINE LOGSTREAM NAME(ATR.IONAPLEX.ARCHIVE) HLQ(IXGLOGR) MODEL(NO) LS_SIZE(1024) LOWOFFLOAD(0) HIGHOFFLOAD(80) RETPD(15) AUTODELETE(YES) DASDONLY(YES) DEFINE LOGSTREAM NAME(ATR.IONAPLEX.RM.DATA) HLQ(IXGLOGR) MODEL(NO) LS_SIZE(1024) LOWOFFLOAD(0) HIGHOFFLOAD(80) RETPD(15) AUTODELETE(YES) DASDONLY(YES) DEFINE LOGSTREAM NAME(ATR.IONAPLEX.MAIN.UR) HLQ(IXGLOGR) MODEL(NO) LS_SIZE(1024) LOWOFFLOAD(0) HIGHOFFLOAD(80) RETPD(15) AUTODELETE(YES) DASDONLY(YES) DEFINE LOGSTREAM NAME(ATR.IONAPLEX.DELAYED.UR) HLQ(IXGLOGR) MODEL(NO) LS_SIZE(1024) LOWOFFLOAD(0) HIGHOFFLOAD(80) RETPD(15) AUTODELETE(YES) DASDONLY(YES) DEFINE LOGSTREAM NAME(ATR.IONAPLEX.RESTART) HLQ(IXGLOGR) MODEL(NO) LS_SIZE(1024) LOWOFFLOAD(0) HIGHOFFLOAD(80) RETPD(15) AUTODELETE(YES) DASDONLY(YES) /* </pre>

Starting RRS

Perform the following steps to start RRS:

Step	Action
1	Update the <code>IEFSSNxx</code> member of <code>SYS1.PARMLIB</code> to add RRS as an OS/390 subsystem as follows: <pre>SUBSYS SUBNAME(RRS)</pre> An IPL is required to activate this change. Dynamic subsystem definition is not supported by RRS, so you cannot use the <code>SETSSI ADD,SUBNAME=RRS</code> command to define RRS.
2	Copy <code>SYS1.SAMPLIB(ATRRRS)</code> to <code>SYS1.PROCLIB(RRS)</code>
3	Start RRS by issuing the following operator command: <pre>S RRS</pre>

Stopping RRS

To stop RRS, issue the following command:

```
SETRRS CANCEL
```

Restarting IMS when RRS is available on the system

Restart the IMS control region. The following message must appear in the IMS control region output to indicate that IMS has attached to RRS:

```
DFS0653I PROTECTED CONVERSATION PROCESSING WITH RRS/MVS ENABLED
```

For recent versions of IMS, such as IMS v8 (with very up-to-date maintenance) and IMS v9, you might also need to specify `RRS=Y` as a start-up parameter to the IMS control region, before RRS can be activated in IMS.

RRS Plug-In Configuration Items

In this section

This section provides a detailed description of the RRS plug-in configuration items. It discusses the following topics:

- [Server adapter resource manager name](#)
 - [Initial reference name for RRS plug-in](#)
-

Server adapter resource manager name

The related configuration item is `plugins:rrs:rm-name`. It specifies the resource manager name that the IMS server adapter uses to register with RRS. The server adapter registers with RRS as a communications resource manager, because it only forwards transactional requests and does not itself manage incoming data on a transactional basis (that is, it supports only communication and is not a database). Each server adapter should have its own resource manager name that it uses to register with RRS. The resource manager name should also be in a dot-separated format; for example, as follows: `TEST.IMSADAP1.IONA.UA`

According to the rules of RRS on the naming of resource managers, the resource manager name for the server adapter must be suffixed with `.UA`. This indicates to RRS that the server adapter might run without APF authorization and that it does not use any of the RRS services that require APF authorization. The second last item in the name should be the company name that provides this resource manager. Depending on the naming schemes in your company, this should either be `IONA` or the name of your company. Using `IONA` is usually the best option, to ensure that the resource manager names do not conflict with resource managers provided by other companies. The rest of the name should be specified in such a way that it is unique for each server adapter.

The presence of this configuration item triggers the server adapter to attempt to load RRS.

Initial reference name for RRS plug-in

The related configuration item is `initial_references:IT_RRS:plugin`. It specifies that the RRS plug-in should be used for RRS services in the server adapter. This should always be set to `RRS` and is a required item if RRS is used.

Configuring the IMS Server Adapter for Client Principals

The IMS server adapter can be configured to read the client principal from incoming GIOP 1.0 and 1.1 requests. It can also be configured to read the principal from a service context for GIOP 1.2. If the server adapter reads the principal from the GIOP request, it passes it into IMS for mapped requests. The server adapter can also run the transaction in IMS under the user principal obtained from the client. This chapter explains how to configure the server adapter to use client principals.

In this chapter

This chapter discusses the following topics:

Activating Client Principal Support	page 97
Setting up the Required Privileges	page 100

[Additional Requirements for IMS Protocol Plug-Ins](#)

[page 102](#)

Note: See [“Securing and Using the IMS Server Adapter”](#) on [page 161](#) for more details about the use of client principals when running the server adapter in secure mode.

Activating Client Principal Support

Overview

For IDL mapped requests, the server adapter marshals the principal data into IMS, making it available to the Orbix server inside IMS. The server adapter can also be configured to run the transaction in IMS under this client's user ID for both `imsraw` requests and mapped requests.

This section discusses the following topics:

- [Using CORBA::Principal](#)
- [Configuring the imsa plug-in](#)

Using CORBA::Principal

`CORBA::Principal` has been deprecated by the OMG in GIOP 1.2 and higher. Hence the principal can only be made available to the server adapter via GIOP 1.0 or 1.1 client requests. However, GIOP 1.2 can still be used. In this case, the client must pass the principal string in a service context and the server adapter must be configured to read the principal from this service context.

Configuring the imsa plug-in

To configure `client_principal` support, the following items within the server adapter's configuration scope must be reviewed.

Table 2: *Client Principal Support and imsa Plug-In Configuration Items (Sheet 1 of 2)*

Configuration Item	Description
<p><code>plugins:imsa:use_client_principal</code></p>	<p>When this item is set to "true", the principal is to be obtained from GIOP, truncated to eight characters and converted to uppercase. The IMS server adapter then also runs the transaction under the user ID. If no principal is available or it is invalid, the transaction fails.</p> <p>Setting this item to "true", therefore, instructs the IMS server adapter to use OS/390 services, to assume the identity of the client when communicating with IMS. This results in IMS and either APPC or OTMA making their security checks against that user ID. If this option is not specified, the security checks are made against the user ID of the server adapter itself. The use of this option requires that the server adapter has special privileges set up. See "Securing the IMS Server Adapter" on page 163 for more details about using this configuration item. When this item is set to "false", the transaction runs under the server adapter's user ID.</p> <p>When this item is set to "true" or "false", the principal is still obtained from GIOP and passed as is (apart from being converted from ASCII to EBCDIC) to the transaction inside IMS, if <code>imsraw</code> is not being used. If the client principal is not available from GIOP, it is not passed as part of the request to IMS, but the transaction is still executed.</p> <p>The default is "false".</p>
<p><code>policies:iiop:server_version_policy</code></p>	<p>This must be set to 1.0 or 1.1, unless the intention is to pass the principal in a service context. The server adapter then only publishes IORs with a version number (1.0 or 1.1) that supports transmission of the <code>CORBA::principal</code> if 1.0 or 1.1 has been selected. If the intention is to use a service context to transmit the principal to the server adapter, this item must be set to 1.2 (the default).</p>
<p><code>policies:giop:interop_policy:enable_principal_service_context</code></p>	<p>For GIOP 1.2, if this item is set to "true", it instructs the server adapter to look for the principal string in a service context. The default value is "false".</p>

Table 2: *Client Principal Support and imsa Plug-In Configuration Items (Sheet 2 of 2)*

Configuration Item	Description
<code>policies:giop:interop_policy:principal_service_context_id</code>	This item specifies the service context ID from which the IMS server adapter attempts to read the principal string if <code>policies:giop:interop_policy:enable_principal_service_context</code> is set to <code>true</code> . The default service context ID where the server adapter looks for the principal string is <code>0x49545F44</code> .

Setting up the Required Privileges

Overview

If the IMS server adapter is to be run using the `use_client_principal` configuration item in the APPC or OTMA plug-ins, the user ID under which the server adapter runs might need to be granted special privileges to enable thread-level security environments. The requirements vary, depending on whether the FACILITY RACF class profile `BPX.SERVER` is defined on your system.

This section discusses the following topics:

- [Requirements when BPX.SERVER is defined](#)
 - [Requirements when BPX.SERVER is not defined](#)
 - [Impersonating users](#)
-

Requirements when BPX.SERVER is defined

If `BPX.SERVER` is defined, the user ID does not need to have a `UID` of 0, but it must have `READ` access to the `BPX.SERVER` profile. In addition, the server adapter executable must reside in an OS/390 load library that is PADS-defined. (PADS is the acronym for Program Access to Data Sets.)

Requirements when BPX.SERVER is not defined

If `BPX.SERVER` is not defined, this user ID must have a `UID` of 0 assigned to it in the `OMVS` segment of its RACF user profile.

Impersonating users

Additionally, because the IMS server adapter is processing requests for users without having their passwords, you must activate the `SURROGAT` RACF class and define profiles in it that allow the server adapter's user ID to *impersonate* particular users. You can do this by establishing a profile for each potential client user. For example:

```
RDEFINE SURROGAT BPX.SRV.client1 UACC(NONE)
PERMIT BPX.SRV.client1 CLASS(SURROGAT) ID(Adapter) ACCESS(READ)
RDEFINE SURROGAT BPX.SRV.client2 UACC(NONE)
PERMIT BPX.SRV.client2 CLASS(SURROGAT) ID(Adapter) ACCESS(READ)
```

Alternatively, you might want to use a generic profile that allows the IMS server adapter to *impersonate* any client user. For example:

```
RDEFINE SURROGAT BPX.SRV.* UACC(NONE)
PERMIT BPX.SRV.* CLASS(SURROGAT) ID(Adapter) ACCESS(READ)
```

Access to such profiles should be very tightly controlled.

Additional Requirements for IMS Protocol Plug-Ins

Overview

When running authorized and using the `use_client_principal` configuration item in the APPC or OTMA plug-in, the IMS server adapter changes the ID of the thread processing the request to that of the client principal. It then makes the request under the new ID; so, in this case, the request should start the IMS transaction with an ACEE for the client ID.

This section discusses the following topics:

- [Switching threads](#)
- [Making the IMS server adapter program-controlled](#)
- [Making the IMS OTMA server adapter APF-authorized](#)
- [Address space not program-controlled](#)
- [OTMA adapter address space not authorized](#)
- [Further reading](#)

Switching threads

The IMS server adapter uses the `pthread_security_np()` call on the thread that is processing the client request, to switch that thread to run under the requested user ID (client principal). For OTMA, it then issues the `otma_alloc()` call, passing this ID to allocate the session with IMS. For APPC, it issues the APPC calls now that the thread is running under this user ID. For this to work, an OTMA or APPC server adapter must be program-controlled. Additionally, an OTMA server adapter must be APF-authorized.

Making the IMS server adapter program-controlled

To make the IMS server adapter program-controlled, you need to consider the following issues:

Step	Action
1	<p>If the server adapter user ID does not have READ access to the <code>BPX.SERVER</code> RACF resource, in the <code>FACILITY</code> class, you get the <code>EPERM</code> errors when the server adapter is trying to switch identities on the thread. The server adapter user ID also needs access to the <code>BPX.SRV.userid</code> resource in the RACF <code>SURROGAT</code> class where <code>userid</code> is the client principal in question. If the user ID under which the server adapter runs is well controlled, you could possibly give it read access to the <code>BPX.SRV.*</code> resource, to enable the server adapter to handle requests from any client principal.</p>
2	<p>When deploying in UNIX System Services, the IMS server adapter must run in its own address space. You must ensure that the <code>_BPX_SHAREAS</code> variable is not set in the server adapter's environment. The <code>itimsa</code> shell script supplied by IONA handles this, by unsetting this variable before running the server adapter program.</p>
3	<p>When deploying in UNIX System Services, you must ensure that any UNIX System Services files that are involved in running the server adapter have the appropriate extended attributes set. Your systems programmer might execute the <code>extattr</code> command, as follows, to make these files program-controlled:</p> <pre data-bbox="635 1116 1089 1164">\$ cd \$IT_PRODUCT_DIR \$ extattr +p shlib/* asp/6.0/bin/itimsa</pre> <p>The command <code>ls -E</code> can be used to display the extended file attributes in the UNIX System Services shell.</p>

Making the IMS OTMA server adapter APF-authorized

In addition to running program-controlled, if the server adapter is communicating with IMS over OTMA, the address space must be running APF-authorized. This means that all load modules (executables) used by an IMS OTMA server adapter must reside in an APF-authorized location. To ensure that an IMS OTMA server adapter is running APF-authorized:

1. The following load libraries must be APF-authorized:

- ◆ `orbixhlq.LPA`
- ◆ `orbixhlq.RUN`

This is required regardless of whether the IMS OTMA server adapter is deployed in a native OS/390 or UNIX System Services environment.

Note: When running in native OS/390, all libraries in the STEPLIB must be APF-authorized.

2. When deploying in UNIX System Services, you must ensure that any additional UNIX System Services files involved in running the adapter have the appropriate extended attributes set. Your systems programmer might execute the `extattr` command, as follows, to make these files APF-authorized:

```
$ cd $IT_PRODUCT_DIR
$ extattr +a shlib/* asp/6.0/bin/itimsa
```

The command `ls -E` can be used to display the extended file attributes in the UNIX System Services shell.

Address space not program-controlled

If, at this point, the address space is still not program-controlled, the server adapter throws an exception back to the client and logs an error message to indicate that it could not switch to that user ID, and therefore it is not going to attempt to start the transaction in IMS.

OTMA adapter address space not authorized

If, at this point, the address space for the OTMA-based server adapter is still not authorized, OTMA ignores the supplied ID and uses the primary address space ID without notifying the server adapter that it has done so. This therefore explains why, if the server adapter address space is not fully authorized, you might see a message from the server adapter saying it is making the request in the client's ID, but the request arrives in IMS with the server adapter's ID. In this case, verify that you have completed all the above steps.

Further reading

Refer to the IBM publications *z/OS V1R2.0 UNIX System Services Planning, GA22-7800-01* or *Planning: OpenEdition MVS, SC23-3015* for more information on enabling thread-level security for servers.

Configuring the Orbix Runtime inside IMS

This chapter provides information on configuring the Orbix runtime that is used by Orbix servers running in IMS.

In this chapter

This chapter discusses the following topics:

Customizing the IMS JCL	page 108
Customizing Orbix Event Logging	page 109

Customizing the IMS JCL

Overview

This section describes how to customize the IMS JCL used to run Orbix servers inside IMS.

Customizing IMS JCL

To customize the IMS JCL perform the following steps:

Step	Action
1	The following library should be added to the IMS message region's STEPLIB concatenation as follows: <pre>DD DSN=HLQ.ORBIX60.MFA.LOAD,DISP=SHR</pre>
2	Check if the following entries are already defined in the IMS message region's JCL. If not, they should be added, to ensure you receive all output from your IMS servers: <pre>SYSPRINT DD SYSOUT=* CEEDUMP DD SYSOUT=* CEEOUT DD SYSOUT=* SYSOUT DD SYSOUT=*</pre>
3	Recycle the message regions to pick up these libraries.

Customizing Orbix Event Logging

Overview

For the Orbix runtime in IMS, most of the configuration settings are fixed. However, the level of event logging performed by the runtime can be customized for the server adapter.

This section discusses the following topics:

- [Customizing the level of event logging](#)
 - [Event logging settings](#)
 - [ORXMFACTx DLL setting](#)
 - [Modifying the ORXMFACTx DLL setting](#)
-

Customizing the level of event logging

This is done by modifying the `ORXMFACTx` DLL. This DLL contains an S390 Assembler `CSECT` that supplies the event logging string to the runtime.

Event logging settings

The event logging settings are as follows:

Table 3: *Event Logging Settings for the IMS Server Adapter*

Value	Description
0	<code>LOG_NONE</code> — no logging in IMS is performed.
1	<code>LOG_ERROR</code> —only log errors.
2	<code>LOG_WARNING</code> —log warnings and errors.
3	<code>LOG_INFO_HIGH</code> —log high priority informational messages, warnings and errors.
4	<code>LOG_INFO_MED</code> —log medium priority informational messages, high priority informational messages, warnings and errors.
5	<code>LOG_INFO_LOW</code> —log low priority informational messages, medium priority informational messages, high priority informational messages, warnings and errors.
6	<code>LOG_INFO_ALL</code> —log all messages.

ORXMFACTx DLL setting

The `ORXMFACTx` DLL shipped with the IMS server adapter has a setting of 2 for event logging in IMS.

This setting can be modified to some other setting. For example, to help trace a problem with a transaction in IMS, it can be changed to 6.

Modifying the ORXMFACTx DLL setting

This is done using the `MFACTLINK` JCL member supplied in the main JCL library of the installed package. In this JCL, the `LOGLVL` variable can be modified to contain the new event logging value. It can then be run to create a new version of the `ORXMFACTx` DLL with this new value. Ensure that you make a backup copy of `ORXMFACTx`, before running this JCL member. After this re-link of the DLL, make it available to the IMS region in which you are testing, for the new setting to come into effect. After the testing is complete, consider copying back the original DLL, to revert to the normal logging levels.

IDL Compiler Configuration

This chapter describes Orbix IDL compiler settings for the mfa plug-in, which is used to generate IMS server adapter mapping files and type_info files.

In this chapter

This chapter discusses the following topics:

Overview	page 112
Configuration settings	page 112
Mandatory settings	page 112
User-defined settings	page 113
List of available settings	page 113

Overview

The `-mfa` plug-in allows the IDL compiler to generate IMS server adapter mapping members and IMS server adapter `type_info` files from IDL. The behavior of the Orbix IDL compiler is defined by the IDL compiler configuration file, `orbixhlq.CONFIG(IDL)`. This section details the default settings used and describes how these can be modified.

Note: IDL compiler configuration is separate from standard Orbix configuration and is contained in its own configuration member (`orbixhlq.CONFIG(IDL)`).

Configuration settings

The IMS server adapter mapping member configuration is listed under `MFAMappings` as follows:

```
MFAMappings
{
    Switch = "mfa";
    ShlibName = "ORXBMFA";
    ShlibMajorVersion = "6";
    IsDefault = "NO";
    PresetOptions = "";

    # Mapping & Type Info file suffix and ext. may be overridden
    # The default mapping file suffix is A
    # The default mapping file ext. is .map and none for OS/390
    # The default type info file suffix is B
    # The default type info file ext. is .inf and none for OS/390
    # MFAMappingExtension = "";
    # MFAMappingSuffix = "";
    # TypeinfoFileExtension = "";
    # TypeinfoFileSuffix = "";

};
```

Note: Settings listed with a `#` are considered to be comments and are not in effect.

Mandatory settings

The first three of the preceding settings are mandatory and must not be altered. They inform the Orbix IDL compiler how to recognize the server adapter mapping member switch, and what name the DLL plug-in is stored under.

User-defined settings

All but the first three settings are user-defined and can be changed. The reason for these user-defined settings is to allow you to change, if you want, default configuration values that are set during installation. To enable a user-defined setting, use the following format.

```
setting_name = "value";
```

List of available settings

Table 4 provides an overview and description of the available settings.

Table 4: *Server Adapter Mapping Member Configuration Settings*

Setting Name	Description	Default
IsDefault	Indicates whether the Orbix IDL compiler generates server adapter mapping members by default from IDL. If this is set to YES, you do not need to specify the <code>-mfa</code> switch when running the compiler.	NO
PresetOptions	The arguments that are passed by default as parameters to the Orbix IDL compiler for the purposes of generating server adapter mapping members.	
MFAMappingExtension	Extension for the server adapter mapping file (on UNIX System Services).	map
TypeinfoFileExtension	Extension for server adapter <code>type_info</code> files (on UNIX System Services).	inf
TypeinfoFileSuffix	Suffix for server adapter <code>type_info</code> files (on native OS/390 and UNIX System Services). If you do not supply a value for this, a default suffix of <code>B</code> is used.	B

Table 4: *Server Adapter Mapping Member Configuration Settings*

Setting Name	Description	Default
MFAMappingSuffix	Suffix for the server adapter mapping member on OS/390. If you do not specify a value for this, a default suffix of <code>A</code> is used.	<code>A</code>

Part 3

Configuring the Client Adapter and the Orbix Runtime inside IMS

In this part

This part contains the following chapters:

Introduction to Client Adapter Configuration	page 117
Client Adapter General Configuration	page 123
Configuring the Client Adapter AMTP_APPC Plug-in	page 125
Configuring the Client Adapter mfu Plug-in	page 147
Configuring the Orbix Runtime Inside IMS	page 149

Introduction to Client Adapter Configuration

This chapter provides information needed to configure the client adapter and its components (plug-ins). It provides descriptions of all the configuration items involved in running the client adapter. It also provides details on configuring the various system components used by the client adapter. These components include AMTP_APPC and MFU.

In this chapter

This chapter discusses the following topics:

A Client Adapter Sample Configuration	page 118
Configuration Summary of Client Adapter Plug-Ins	page 120

A Client Adapter Sample Configuration

Overview

A sample configuration member is supplied with your Orbix Mainframe installation that provides an example of how you might configure and deploy the client adapter on both native OS/390 and UNIX System Services.

This section discusses the following topics:

- [Location of configuration templates](#)
 - [Configuration scope](#)
 - [Configuration scope example](#)
 - [Configuring a domain](#)
-

Location of configuration templates

Sample configuration templates are supplied with your Orbix Mainframe installation in the following locations:

- Non-TLS template—`orbixhlq.CONFIG(FILETMPL)`
 - TLS template—`orbixhlq.CONFIG(TLSTMPL)`
-

Configuration scope

The client adapter uses an ORBname of `iona_services.mfu`. The items specific to the client adapter configuration are scoped in the `iona_services.mfu` configuration scope.

Configuration scope example

The following is an example of the `iona_services.mfu` configuration scope.

Example 7: An `iona_services.mfu` Configuration Scope Example

```

iona_services
{
    mfu
    {
        # Client Adapter logging
        #
        event_log:filters = ["*=WARN+ERROR+FATAL", "IT_MFU=INFO_HI+WARN+ERROR+FATAL"];

        # Client Adapter amtp_appc plugin
        #
        plugins:amtp_appc:symbolic_destination = "ORXCLNT1";
        plugins:amtp_appc:appc_function_wait = "5";
        plugins:amtp_appc:min_comm_threads = "5";
        plugins:amtp_appc:max_comm_threads = "10";

        # Client Adapter mfu plugin
        #
        initial_references:IT_AMTP_OutgoingRequestHandler:plugin = "mfu";
        plugins:mfu:target_cache_size = "10";
        plugins:mfu:interface_cache_size = "10";

        plugins:ots_lite:use_internal_orb = "true";
        plugins:ots_lite:orb_name= "iona_services.mfu.ots";

        orb_plugins = ["local_log_stream", "iiop_profile", "giop",
                      "iiop", "mfu", "amtp_appc", "wto_announce"];

        ots
        {
            orb_plugins = ["local_log_stream", "iiop_profile", "giop", "iiop"];
        };
    };
};

```

Configuring a domain

Refer to the *CORBA Administrator's Guide* for details on how to configure an Application Server Platform domain.

Configuration Summary of Client Adapter Plug-Ins

Overview

Orbix configuration allows you to configure an application on a per-plug-in basis. This section provides a summary of the configuration items associated with plug-ins specific to the client adapter.

This section discusses the following topics:

- [Client adapter plug-ins](#)
- [Summary of items for the amtp_appc plug-in](#)
- [Summary of items for the mfu plug-in](#)
- [Summary of remaining configuration items](#)

Client adapter plug-ins

The client adapter consists of two plug-ins:

- The `amtp_appc` plug-in, which is used to provide APPC transport between IMS client transactions and the client adapter.
- The `mfu` plug-in, which is used to locate target servers, invoke operations on the servers, and get the reply from the invoked operation.

Summary of items for the amtp_appc plug-in

The following is a summary of the configuration items associated with the `amtp_appc` plug-in. Refer to [“AMTP_APPC Plug-In Configuration Items” on page 144](#) for more details.

`symbolic_destination` Specifies the APPC/MVS symbolic destination name the client adapter uses for APPC services. The Orbix Runtime in IMS uses the symbolic destination to send IMS client transaction requests to the client adapter. The default value is “`ORXCLNT1`”.

`appc_function_wait` Specifies the number of minutes that the client adapter can wait for a response from an IMS client transaction before canceling the request. Valid values are in the range 0–1440. The default value is 5 minutes.

<code>min_comm_threads</code>	Specifies the minimum number of client adapter threads used to service requests from IMS client transactions. Each thread processes a request from an IMS client transaction. A valid value is greater than 0. The default value is 5 threads.
<code>max_comm_threads</code>	Specifies the maximum number of client adapter threads that can be used to service requests from IMS client transactions. If all client adapter threads are busy, and the client adapter receives another request, it dynamically starts more threads up to this maximum number. The default value is 10 threads.

Summary of items for the mfu plug-in

The following is a summary of the configuration items associated with the `mfu` plug-in. Refer to [“Configuring the Client Adapter mfu Plug-in” on page 147](#) for more details.

<code>target_cache_size</code>	Specifies the size of the target cache. A value of 0 indicates there is no caching of targets. This is an optional item. The default value is 0.
<code>interface_cache_size</code>	Specifies the size of the interface cache. A value of 0 indicates there is no caching of interface data type information obtained from the IFR. This is an optional item. The default value is 0.

Summary of remaining configuration items

The following is a summary of the remaining configuration items. Refer to [“Client Adapter General Configuration” on page 123](#) and the *CORBA Administrator's Guide* for more details.

<code>event_log:filters</code>	Specifies the types of events the client adapter logs.
<code>orb_plugins</code>	List of standard ORB plug-ins the client adapter should load.

Client Adapter General Configuration

This chapter provides details of the configuration items for the core client adapter. These details specify the level of Orbix Event logging and plug-ins to be loaded when the ORB is initializing.

In this chapter

This chapter discusses the following topics:

- [Orbix event logging](#)
 - [ORB plug-ins list](#)
 - [WTO message format](#)
-

Orbix event logging

The related configuration item is `event_log:filters`. It specifies the level of event logging. To obtain events specific to the client adapter, the `IT_MFU` event logging subsystem can be added to this list item. For example:

```
event_log:filters = ["*=WARN+ERROR+FATAL", "IT_MFU=INFO_HI+INFO_MED+WARN+ERROR+FATAL"];
```

This logs all `IT_MFU` events (except for `INFO_LOW` — low priority informational events), and any warning, error, and fatal events from all other subsystems (for example, `IT_CORE`, `IT_GIOP`, and so on). The level of detail provided for `IT_MFU` events can be controlled by setting the relevant logging levels. Refer to the *CORBA Administrator's Guide* for more details.

The following is a categorization of the informational events associated with the `IT_MFU` subsystem.

<code>INFO_HI</code>	Configuration settings and client adapter start-up and shutdown messages
<code>INFO_MED</code>	APPC informational messages
<code>INFO_LOW</code>	IMS segment data streams

ORB plug-ins list

The related configuration item is `orb_plugins`. It specifies the ORB-level plug-ins that should be loaded into your application at `ORB_init()` time. On OS/390, you can add the WTO announce plug-in support to any Orbix application by updating this list in the relevant configuration scope. For example, in the `iona_services.mfu` scope:

```
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
              "iiop", "mfu", "amtp_appc", "wto_announce"];
```

This causes the client adapter to load the WTO announce plug-in and issues a WTO message when the client adapter ORB starts up and shuts down.

WTO message format

Messages have the following format:

```
+ORX2001I ORB iona_services.mfu STARTED (HOSTNAME:<process id>)
+ORX2002I ORB iona_services.mfu ENDED (HOSTNAME: <process id>)
```

On OS/390 UNIX System Services, `<process id>` is a pid. On native OS/390, `<process id>` is a job name and an `A=xxxx` job identifier.

Configuring the Client Adapter AMTP_APPC Plug-in

The AMTP_APPC plug-in for the client adapter uses APPC to communicate with client transactions. This chapter describes how to configure APPC for IMS, and the client adapter AMTP_APPC plug-in configuration.

In this chapter

This chapter discusses the following topics:

Setting Up APPC for the Client Adapter	page 126
Additional RACF Customization Steps for APPC	page 138
AMTP_APPC Plug-In Configuration Items	page 144

Setting Up APPC for the Client Adapter

Prerequisites to using APPC

Before you can run the client adapter, you must first enable the required APPC functionality on your OS/390 system. Depending on your installation, one or all of these tasks might already have been completed.

Further reading

For more information on setting up APPC/MVS, refer to the IBM publication *MVS Planning: APPC/MVS Management, GC28-107*.

Additionally, you can find specific information on how IMS uses APPC in the chapter on “Administering APPC/IMS and LU 6.2 Devices” in the IBM publication *IMS/ESA Administration Guide: Transaction Manager, SC26-8104*.

In this section

This section discusses the following topics:

Defining LUs to APPC	page 127
Defining an APPC Destination Name for the Client Adapter	page 130
Defining LUs to VTAM	page 134

Defining LUs to APPC

Overview

A Logical Unit (LU) name identifies each side of an APPC conversation. It is defined to APPC/MVS in the `APPCPMxx` member of `SYS1.PARMLIB`. You must define at least two LU names to use the client adapter—one for the client adapter, and one for IMS.

This section discusses the following topics:

- [Associating an IMS LU with a specific IMS region](#)
- [Client adapter LU](#)
- [Specifying the APPC/MVS-side information dataset name](#)
- [Client adapter LU name and security](#)
- [Running multiple client adapters](#)

Associating an IMS LU with a specific IMS region

The IMS LU definition is associated with a specific IMS region by specifying the name of that region (`IMSID` from the `IMSCTRL` macro in the IMS system generation) as the transaction scheduler for the LU. For example:

```
LUADD ACBNAME(IMSLU01)
BASE
SCHED(IMS1)
```

Note: If you are using the IMS APPC plug-in for the IMS server adapter, this step might already have been performed. Refer to [“Associating an IMS LU with a specific IMS region” on page 73](#) for more details.

Refer to [“Customizing Orbix APPC Local LU” on page 160](#) for information on where the IMS LU must be configured for the Orbix runtime in IMS.

Client adapter LU

The client adapter LU is used by the client adapter to receive requests from IMS client transactions, and to return replies back to IMS client transactions. It can be defined as follows:

```
LUADD ACBNAME(ORXLUCA1)
NOSCHED
```

Specifying the APPC/MVS-side information dataset name

The APPC/MVS side information dataset contains APPC symbolic destination names. If your installation does not have a side information dataset, see `SYS1.SAMPLIB(ATBSIVSM)` for sample JCL to create one.

The name of the side information dataset must be defined in `SYS1.PARMLIB(APPCPMxxx)` (for example, `SIDEINFO DATASET(SYS1.APPCSI)`).

Note: If you are using the IMS APPC plug-in for the IMS server adapter, this step might have already been performed. Refer to [“Specifying the APPC-side information dataset name” on page 73](#) for more details.

Client adapter LU name and security

If you define a new LU for the client adapter (for example, `ORXLUCAL`), its LU name must be used as part of the `APPCLU` RACF profile name when securing LU conversations. Refer to [“APPCLU profiles” on page 139](#) for more information.

Running multiple client adapters

If you want to run multiple client adapters, you must first decide if you want the client adapters to share APPC/MVS allocation queues.

APPC/MVS allocation queues hold requests to start APPC conversations. As client transactions initiate requests to the client adapter, they are first placed in an APPC/MVS allocation queue. The requests designate which LU and Transaction Program Name (TPN) they are destined for. The client adapter registers with APPC/MVS and specifies the LU and TPN requests it expects to process. (Refer to [“Defining an APPC Destination Name for the Client Adapter” on page 130](#) for details of how to set up the LU and TPN name used by the client adapter.) APPC/MVS delivers the requests from the allocation queue to the client adapter.

You can choose to run multiple client adapters that specify the same LU and TPN. The client adapters all share the same APPC/MVS allocation queue. APPC/MVS chooses one of the client adapters to deliver the request to. This approach can be used as a form of load balancing where the load is spread over multiple client adapters. This approach also provides a measure of fault tolerance. If a client adapter is stopped or goes down, allocation requests from client transactions can still be processed by other client adapters.

You can alternatively choose to run multiple client adapters where each client adapter specifies a different LU and TPN. The client adapters all have their own APPC/MVS allocation queue. This approach is useful for setting up

a test client adapter along with a production client adapter. The Orbix runtime inside the test IMS region is configured to direct allocation requests to the test client adapter, while the Orbix runtime inside the production IMS region is configured to direct allocation requests to the production client adapter.

Defining an APPC Destination Name for the Client Adapter

Overview

An IMS client transaction connects to the client adapter through an APPC destination name rather than directly through the client adapter LU name. The APPC destination name is used to establish various default characteristics for the APPC conversation being initiated, including the name of the partner LU, the TPN, and a logon mode name.

This section discusses the following topics:

- [Storage of the APPC destination name](#)
- [Example of the APPC destination name](#)
- [Explanation of the APPC destination name](#)
- [Example of multiple APPC destination names](#)
- [Explanation of multiple APPC destination names](#)

Storage of the APPC destination name

The APPC destination name information is stored in the APPC-side information data set. This data set is updated using the ATBSDFMU APPC/MVS utility program.

Example of the APPC destination name

The following is an example of defining an APPC destination name.

Example 8: *Example of an APPC Destination Name*

```
//SIADDEXEC PGM=ATBSDFMU
//SYSPRINT DD SYSOUT=*
//SYSSDLIB DD DSN=SYS1.APPCSI,DISP=SHR
//SYSSDOUT DD SYSOUT=*
//SYSIN DD DATA
SIADD
1 DESTNAME(ORXCLNT1)
2 TPNNAME(ORXCLNT1)
3 MODENAME(APPCHOST)
4 PARTNER_LU(ORXLUCAL)
/*
```

Explanation of the APPC destination name

The example APPC destination name can be explained as follows:

- 1 The `DESTNAME` is a symbolic name that contains the `TPNAME`, `MODENAME`, and `PARTNER_LU`. It is used in two places:
 - The Orbix runtime inside IMS configuration specifies which destname the IMS region uses for APPC communication with the client adapter.
 - The `amtp_appc` plug-in configuration item `symbolic_destination`, which tells the client adapter which LU and TPN to use for APPC communication. The LU/TPN define the APPC/MVS allocation queue from which the client adapter receives allocation requests.
- 2 The `TPNAME` specification forms part of the APPC/MVS allocation queue designation. If you want to run a test client adapter along with a production client adapter, two symbolic destinations can be defined. They can each specify the same `MODENAME` and `PARTNER_LU`, but each can specify a different `TPNAME`. (Refer to [“Example of multiple APPC destination names” on page 132](#) for more information.)
- 3 The `MODENAME` parameter is used to name an entry in the VTAM logon mode table. This specifies other characteristics that are to be used in the conversation. See the `SYS1.SAMPLIB(ATBLMODE)` data set for a definition of the `APPCHOST` logon mode, and the `SYS1.SAMPLIB(ATBLJOB)` data set for the JCL to install it.
- 4 `PARTNER_LU` must specify the client adapter LU name.

Example of multiple APPC destination names

You might want to define multiple APPC destination names to allow multiple client adapters that do not share APPC/MVS allocation queues. A good example of this is to have a production client adapter processing requests from a production IMS region, and a test client adapter processing requests from a test IMS region.

Example 9: Example of Multiple APPC Destination Names

```

1 //SIADDEXEC PGM=ATBSDLFUM
//SYSPPRINT DD SYSOUT=*
//SYSSDLIB DD DSN=SYS1.APPCSI,DISP=SHR
//SYSSDOUT DD SYSOUT=*
//SYSIN DD DATA
2 SIADD
3 DESTNAME(ORXCLNT1)
4 TPNAME(ORXCLNT1)
5 MODENAME(APPCHOST)
PARTNER_LU(ORXLUCAL)
SIADD
2 DESTNAME(ORXTEST)
3 TPNAME(ORXTEST)
4 MODENAME(APPCHOST)
5 PARTNER_LU(ORXLUCAL)
/*

```

Explanation of multiple APPC destination names

The example multiple APPC destination names can be explained as follows:

- 1 The first `SIADD` statement defines the production destination, as explained in [“Explanation of the APPC destination name” on page 131](#).
- 2 A second `DESTNAME` is defined for the test destination. It defines a different name from the production `DESTNAME`. The production IMS region and production client adapter are configured to use the production `DESTNAME`. The test IMS region and test client adapter are configured to use the test `DESTNAME`.
- 3 The test `DESTNAME` defines a `TPNAME` that is different from the production `TPNAME`. This causes APPC/MVS to use separate allocation queues for the production and test client adapters.
- 4 The test `MODENAME` is the same as the production `MODENAME`.

- 5 The test `PARTNER_LU` is the same as the production `PARTNER_LU`. This means you can run multiple client adapters that do not share APPC/MVS allocation queues, yet still use the same LU name for each.

Defining LUs to VTAM

Overview

APPC/MVS expects its LUs to be defined as VTAM resources, so that they can access a SNA network.

This section discusses the following topics:

- [VTAM requirements for LUs](#)
- [Using SYS1.SAMPLIB\(ATBAPPL\)](#)
- [APPC definition parameter security requirements](#)

VTAM requirements for LUs

Although the client adapter is usually run on the same system as the IMS region with which it communicates (that is, an LU=LOCAL conversation), VTAM application program definition (APPL) macros must still be coded for each LU. See SYS1.SAMPLIB(ATBAPPL) for a sample APPL definition of an APPC LU.

Using SYS1.SAMPLIB(ATBAPPL)

The following definitions for the IMS and client adapter LUs use the SYS1.SAMPLIB(ATBAPPL) definition, with some changes (which are highlighted). If you are using the IMS APPC plug-in for the IMS server adapter, the IMS LU might already be defined. Refer to [“Using SYS1.SAMPLIB\(ATBAPPL\)” on page 77](#) for more information.

Example 10: *Example of APPL Definitions for Client Adapter LUs*

```

1  IMSLU01 APPL ACBNAME=IMSLU01,           C
2                                     APPC=YES,           C
3                                     SECACPT=CONV,        C
4                                     VERIFY=OPTIONAL,     C
5                                     AUTOSES=0,           C
6                                     DDRAINL=NALLOW,      C
7                                     DLOGMOD=APPCHOST,     C
8                                     DMINWNL=5,            C
9                                     DMINWNR=5,            C
10                                    DRESPL=NALLOW,        C
11                                    DSESLIM=10,            C
12                                    LMDENT=19,             C
13                                    MODETAB=LOGMODES,     C
14                                    PARSESS=YES,          C
15                                    SRBEXIST=YES,         C
16                                    VPACING=1

```

Example 10: *Example of APPL Definitions for Client Adapter LUs*

```

1 ORXLUCA1 APPL ACBNAME=ORXLCA1, C
2 APPC=YES, C
3 SECACPT=ALREADYV, C
  VERIFY=OPTIONAL, C
  AUTOSES=0, C
  DDRAINL=NALLOW, C
  DLOGMOD=APPCHOST, C
  DMINWNL=5, C
  DMINWNR=5, C
  DRESPL=NALLOW, C
  DSESLIM=10, C
  LMDENT=19, C
  MODETAB=LOGMODES, C
  PARSESS=YES, C
  SRBEXIST=YES, C
  VPACING=1

```

APPC definition parameter security requirements

The code for `APPL` definitions for client adapter LUs can be explained as follows:

- 1 Both the `ACBNAME=` parameter and the `APPL` statement label should match the LU name defined to `APPC`.
 The `VERIFY=` and `SECACPT=` parameters specify the security levels for each LU. Determining the correct values for these parameters depends on the environment in which IMS and the client adapter are running. A test environment might not require the same level of security that a production environment does.
- 2 `SECACPT=` specifies the greatest level of security information passed on a conversation allocation request from an IMS client transaction to the client adapter. If the LUs are secured using RACF `APPCLU` profiles, this level of security information can be overridden to the value set in the `APPCLU` profile. Refer to [“Additional RACF Customization Steps for APPC” on page 80](#) for more details.
 - `SECACPT=NONE`—If you do not require security, use `SECACPT=NONE` for both `IMSLU01` and `ORXLUCA1`.
 - `SECACPT=CONV`—If you require security, use `SECACPT=CONV` for `IMSLU01`. In this case, `ORXLUCA1` requires a different setting, as described in the next point.

- `SECACPT=ALREADYV`—If you require security, use `SECACPT=ALREADYV` for `ORXLUCAL`.

If you are using security, you can verify that the `SECACPT` setting is correct, by issuing the following command after `IMSLU01` has established sessions with `ORXLUCAL`:

```
D NET,CNOS, ID=ORXLUCAL, LUNAME=IMSLU01
```

The message `IST1005I` should appear as part of the display results. Ensure that `CONVSECL=ALREADYV` appears in the message. If not, you might have to modify `LU ORXLUCAL` in `APPC/MVS`. For more details of how to modify a local LU see the IBM publication *MVS Planning: APPC/MVS Management, GC28-107*.

- 3 `VERIFY=` specifies that VTAM should verify the identity of partner LUs that attempt to establish sessions with this LU. Generally each LU has the same value for `VERIFY=`, but there are valid cases where the values can be different.
 - `VERIFY=NONE`—VTAM should not verify partner LUs. Use this value if security is not required.
 - `VERIFY=OPTIONAL`—VTAM should verify those LUs that have session keys defined. The session keys are defined in the RACF `APPCLU` profile. Refer to the topic on “LU 6.2 Security” in the IBM publication *SNA Network Implementation Guide, SC31-8562* for more information on how VTAM verifies the partner LU. Use this value when security is desired.
 - `VERIFY=REQUIRED`—VTAM should verify every partner LU. This provides even tighter security control. The IMS LU can be defined with `VERIFY=OPTIONAL`, and the client adapter LU can be defined with `VERIFY=REQUIRED`. This provides two benefits:
 - ◆ Compatibility with the IMS server adapter if it is being used.
 - ◆ Only those LUs defined with a proper RACF `APPCLU` profile can connect to the client adapter.

If there is no possibility of unauthorized access from other systems in your SNA network, you might prefer to code `SECACPT=ALREADYV` and `VERIFY=NONE` to indicate that partner LUs do not need to be authenticated. This is safe for `LU=LOCAL` conversations because user information is provided directly by `APPC/MVS`. Therefore, there is no opportunity for the programmer of the

partner LU to fabricate his or her identity. Refer to [“Securing the Client Adapter” on page 263](#) for more details about APPC conversation security and session-level verification.

Additional RACF Customization Steps for APPC

Overview

There are a number of RACF definitions related to APPC that you might need to add or change to run the client adapter. Refer to [“Securing the Client Adapter” on page 263](#) for more details about how the client adapter fits into a secure system environment.

Much of the information provided in this section can be found in the sections relating to LU Security and Conversation Security in the IBM publication *MVS Planning: APPC/MVS Management, GC28-1807*.

In this section

This section discusses the following topics:

LU-to-LU Security Verification	page 139
Protecting LUs	page 141
Enabling APPC/IMS	page 143

LU-to-LU Security Verification

Overview

LU-LU security verification provides a means of controlling which LUs can establish sessions with a particular LU. RACF provides the `APPCLU` class for this purpose.

This section discusses the following topics:

- [APPCLU profiles](#)
- [APPCLU profile contents and operation](#)
- [Accessing APPCLU profiles](#)

APPCLU profiles

`APPCLU` profiles can be defined to control which LUs can establish sessions with a particular LU.

Each `APPCLU` profile name has the form:

`'networkid.local-lu-name.partner-lu-name'`.

Each profile contains information to be used by APPC/MVS on one side of a session between the two named LUs. This means each side of a session has its own specific profile. For example, if LU `IMSLU01` attempts to establish a session with LU `ORXLUCA1`, APPC/MVS on the initiating (outbound) side examines the `'networkid.IMSLU01.ORXLUCA1'` profile, and APPC/MVS on the receiving (inbound) side examines the `'networkid.ORXLUCA1.IMSLU01'` profile.

APPCLU profile contents and operation

Each `APPCLU` profile contains a session key, which is a string of letters or numbers, and a `CONVSEC` setting. When a session is initiated between two LUs, APPC/MVS on the initiating (outbound) side passes the session key found in its `APPCLU` profile to APPC/MVS on the receiving (inbound) side. If APPC/MVS on the inbound side finds that the received session key matches the session key in its own `APPCLU` profile, it overrides the VTAM `SECACPT=`

setting with the `CONVSEC` setting from its profile. Thus, to allow an IMS client transaction to authenticate itself to the client adapter, the following definitions might be used:

```
RDEFINE APPCLU P390.ORXLUCA1.IMSLU01
UACC(NONE) SESSION(SESSKEY(137811C0) CONVSEC(ALREADYV))

RDEFINE APPCLU P390.IMSLU01.ORXLUCA1
UACC(NONE) SESSION(SESSKEY(137811C0) CONVSEC(ALREADYV))

SETROPTS CLASSACT(APPCLU)
```

To refresh the profiles in VTAM, use the following VTAM commands:

```
F VTAM,PROFILES, ID=IMSLU01
F VTAM,PROFILES, ID=ORXLUCA1
```

Accessing APPCLU profiles

It is not necessary to permit the client adapter or IMS region to have user IDs for the `APPCLU` profiles. However, access to the profiles should be tightly controlled to ensure that only appropriate users can read or change the session keys.

Protecting LUs

Overview

Protecting LUs involves controlling the users that are permitted to use the IMS local LU that initiates requests to the client adapter LU, and controlling the users that are permitted to use the client adapter LU that receives requests from IMS.

This section discusses the following topics:

- [Controlling access to the IMS local LU](#)
- [Controlling access to the client adapter LU](#)

Controlling access to the IMS local LU

The IMS local LU initiates requests to allocate conversations with the client adapter. This LU is considered the APPC port of entry. It can be secured by controlling the users that are permitted to use the LU. The RACF `APPCPORT` class provides this security control. First, a profile is defined for the IMS local LU that permits no access. A `PERMIT` is then issued for each user that requires access to the IMS local LU. For example:

```
RDEFINE APPCPORT IMSLU01 UACC(NONE)
PERMIT IMSLU01 CLASS(APPCPORT) ID(USER1) ACCESS(READ)
PERMIT IMSLU01 CLASS(APPCPORT) ID(USER2) ACCESS(READ)
...
SETROPTS CLASSACT(APPCPORT) RACLIST(APPCPORT)
```

Note: To allow IMS to provide the user ID of the user that is running the transaction, rather than the user ID of the user that started the IMS control region, `IMS exit DFSBEX0` must be used. See the IBM publication *IMS/ESA Customization Guide, SC28-8732* for more details.

Controlling access to the client adapter LU

The client adapter LU receives requests initiated by the IMS local LU. The client adapter LU can be secured by controlling the users that are permitted to use this LU. The RACF `APPL` class provides this security control. First, a profile is defined for the client adapter LU that permits no access. A `PERMIT` is then issued for each user that requires access to the client adapter LU. For example:

```
RDEFINE APPL ORXLUCA1 UACC(NONE)
PERMIT ORXLUCA1 CLASS(APPL) ID(USER1) ACCESS(READ)
PERMIT ORXLUCA1 CLASS(APPL) ID(USER2) ACCESS(READ)

SETROPTS CLASSACT(APPL) RACLIST(APPL)
SETROPTS RACLIST(APPL) REFRESH
```

Note: To allow IMS to provide the user ID of the user that is running the transaction, rather than the user ID of the user that started the IMS control region, IMS exit DFSBSEX0 must be used. See the IBM publication *IMS/ESA Customization Guide, SC28-8732* for more details.

Enabling APPC/IMS

Overview

This section describes how to enable APPC/IMS. It discusses the following topics:

- [Enabling APPC/IMS](#)
- [Enabling APPC/IMS security](#)

Enabling APPC/IMS

To enable APPC/IMS, specify `APPC=Y` in the IMS JCL start-up parameters, or enter the following command on a running system:

```
/START APPC
```

Enabling APPC/IMS security

To enable APPC/IMS security, specify `APPCSE=F` in the IMS JCL start-up parameters, or enter the following command on a running system:

```
/SECURE APPC FULL
```

Note: If you are using the IMS APPC plug-in for the IMS server adapter, APPC/IMS might already be enabled.

AMTP_APPC Plug-In Configuration Items

Overview

This section discusses the following topics:

- [APPC destination](#)
 - [AMTP function timeout](#)
 - [APPC minimum communication threads](#)
 - [APPC maximum communication threads](#)
-

APPC destination

The related configuration item is

`plugins:amtp_appc:symbolic_destination`. This specifies the APPC/MVS symbolic destination name that identifies the LU, TPN, and `LOGMODE` the client adapter uses. The Orbix runtime in IMS is configured to use this destination. Refer to [“Customizing Orbix APPC Symbolic Destination” on page 158](#) for more information on configuring the destination in the Orbix runtime in IMS. IMS client transactions have their requests sent to the client adapter using this symbolic destination. The default value is `ORXCLNT1`.

The specified symbolic destination name is verified only when an IMS client transaction attempts to send a request to the client adapter. This means the IMS region does not have to be available when you start the client adapter. Refer to [“Example of the APPC destination name” on page 130](#) for details of how to define the symbolic destination to APPC/MVS.

AMTP function timeout

The related configuration item is `plugins:amtp_appc:function_wait`. It specifies the number of minutes the client adapter waits for a response from the IMS client transaction before canceling the request. It prevents the client adapter from having to wait indefinitely for a response from the IMS client transaction if the transaction has stopped for some reason. The default is 5 minutes.

APPC minimum communication threads

The related configuration item is `plugins:amtp_appc:min_comm_threads`. It specifies the minimum number of client adapter threads that are used to service IMS client transaction requests. Each thread services a single client transaction request. Multiple threads allow for multiple concurrent client requests to be processed. The default is 5 threads.

APPC maximum communication threads

The related configuration item is `plugins:amtp_appc:max_comm_threads`. It specifies the maximum number of client adapter threads that can be used to service IMS client transaction requests. If all client adapter threads are busy, and another request arrives, further threads are started dynamically up to this maximum number. The default is 10 threads.

Configuring the Client Adapter mfu Plug-in

The mfu plug-in for the client adapter receives IMS client transaction requests from the `amtp_appc` plug-in, locates target objects, invokes operations, and returns results to the `amtp_appc` plug-in. This chapter describes how to configure the mfu plug-in.

In this chapter

This chapter discusses the following topics:

- [Target cache size](#)
 - [Interface cache size](#)
-

Target cache size

The related configuration item is `plugins:mfu:target_cache_size`. This specifies the size of the target cache used by the mfu plug-in. An IMS client transaction uses a `corbaloc` to specify the target object against which it wants to issue an operation. (Refer to the *CORBA Programmer's Guide, C++* for more information about using `corbaloc`). The mfu plug-in makes a call to `string_to_object()` to convert the `corbaloc` string to an object reference. The locator service is used to get the object reference. Once the mfu plug-in has the object reference, it can be cached if the target cache

size is greater than 0. The next time an IMS client transaction directs a request to the same target, it is found in the cache rather than having to use the locator service.

When the target cache becomes full, an IMS client transaction request to a new target causes the oldest target in the cache to be removed.

Target cache information is not maintained after the client adapter is stopped. The next run of the client adapter starts with an empty target cache.

This configuration item is optional. The default value is 0.

Interface cache size

The related configuration item is `plugins:mfu:interface_cache_size`. This specifies the size of the interface cache used by the mfu plug-in. When the mfu plug-in processes an IMS client transaction directed to an operation defined in a particular interface, it locates the interface using the IFR service. After the interface has been located, it can be cached if the interface cache size is greater than 0. The next time an IMS client transaction directs a request to an operation in that same interface, it is found in the cache rather than having to use the IFR service.

After the interface cache becomes full, IMS client transaction requests to operations in interfaces that are not cached cause the oldest interface in the cache to be removed.

Interface cache information is not maintained after the client adapter is stopped. The next run of the client adapter starts with an empty interface cache.

This configuration item is optional. The default value is 0.

Note: If the client adapter has cached an interface, and a dynamic update for this interface is made to the IFR, the client adapter does not recognize the change until it is restarted. If you want to make dynamic interface updates to the IFR, this can be avoided by setting `plugins:mfu:interface_cache_size` to 0. However, this negates the performance benefit of caching interfaces and cause the client adapter to use the IFR service to locate the interface on every client transaction.

Configuring the Orbix Runtime Inside IMS

This chapter provides information on configuring the Orbix runtime that is used by Orbix clients running in IMS.

In this chapter

This chapter discusses the following topics:

Customizing the IMS JCL	page 150
Customizing Orbix Event Logging	page 153
Customizing Orbix Maximum Segment Size	page 155
Customizing Orbix APPC Function Timeout	page 157
Customizing Orbix APPC Symbolic Destination	page 158
Customizing Orbix APPC Local LU	page 160

Customizing the IMS JCL

Overview

The IMS JCL must be updated to add the Orbix IMS runtime library to the IMS message region's `STEPLIB`.

Note: If you are using the IMS server adapter, this might have already been performed.

Customizing IMS JCL

To customize IMS JCL perform the following steps.

Step	Action
1	The following library should be added to the IMS message region's <code>STEPLIB</code> concatenation as follows: <code>DD DSN=HLQ.ORBIX60.MFA.LOAD,DISP=SHR</code>
2	Check if the following entries are already defined in the IMS message region's JCL. If not, they should be added to ensure you receive all output from your IMS servers. <ul style="list-style-type: none"> • <code>SYSPRINT DD SYSOUT=*</code> • <code>CEEDUMP DD SYSOUT=*</code> • <code>CEEOUT DD SYSOUT=*</code> • <code>SYSOUT DD SYSOUT=*</code>
3	Recycle the message regions to pick up these libraries.

Customizing Orbix Configuration

Overview

The Orbix configuration inside IMS is DLL-based. (DLL is the acronym for Dynamic Link Library.) The Orbix runtime inside IMS does not access a file for configuration information, but instead gets configuration information from a DLL. The DLL resides in the Orbix IMS runtime library that was added to the IMS message region's STEPLIB. The ORXMFACTx member is the configuration DLL.

This section discusses the following topics:

- [How the configuration is changed](#)
- [Steps to change the configuration](#)
- [S390 Assembler program variables](#)

How the configuration is changed

Changing the configuration involves updating the configuration DLL. The DLL is updated by assembling and linking an S390 Assembler program that defines the configuration settings. See *orbixhlq.JCL(MFACLINK)* for sample JCL to update the DLL. The sample JCL runs the Assembler and re-links the configuration in the DLL. The JCL contains the S390 Assembler program that defines the configuration settings.

Steps to change the configuration

Perform the following steps to update the configuration DLL:

Step	Action
1	Make a backup of your current configuration DLL. The configuration DLL is in <i>orbixhlq.MFALOAD(ORXMFACTx)</i> .
2	Make the appropriate changes to the <i>orbixhlq.JCL(MFACLINK)</i> JCL, as outlined in the JCL comments.
3	Change the S390 Assembler program to define the new configuration settings.
4	Submit the JCL.

Step	Action
5	Make the updated DLL available to your IMS region for the configuration changes to take effect.

S390 Assembler program variables

The following table lists the S390 Assembler program variables that can be changed in order to change the configuration

Table 5: *S390 Assembler Program Variables and Default Values*

Assembler Variable	Description	Default Value
LOGLVL	Event logging level	2
MAXSEG	Maximum APPC segment size	32760
TIMEOUT	APPC function time out	5
SYMBDST	APPC symbolic destination	ORXCLNT1
LOCALLU	APPC LU IMS uses to communicate with the client adapter	IMSLU01

Customizing Orbix Event Logging

Customizing the level of event logging

The level of logging performed by the Orbix IMS runtime can be configured.

Note: If you are using the IMS server adapter, this might have already been performed.

This section discusses the following topics:

- [Customizing the level of event logging](#)
 - [Event logging settings](#)
 - [ORXMFACTx DLL setting](#)
 - [Modifying the ORXMFACTx DLL setting](#)
-

Customizing the level of event logging

This is done by modifying the `ORXMFACTx` DLL. This DLL contains an S390 Assembler `CSECT` that supplies the event logging string to the runtime.

Event logging settings

The event logging settings are as follows:

Table 6: *Event Logging Settings for the IMS Server Adapter*

Value	Description
0	<code>LOG_NONE</code> —no logging in IMS is performed.
1	<code>LOG_ERROR</code> —only log errors.
2	<code>LOG_WARNING</code> —log warnings and errors.
3	<code>LOG_INFO_HIGH</code> —log high priority informational messages, warnings and errors.
4	<code>LOG_INFO_MED</code> —log medium priority informational messages, high priority informational messages, warnings and errors.
5	<code>LOG_INFO_LOW</code> —log low priority informational messages, medium priority informational messages, high priority informational messages, warnings and errors.

Table 6: *Event Logging Settings for the IMS Server Adapter*

Value	Description
6	LOG_INFO_ALL—log all messages.

ORXMFACTx DLL setting

The ORXMFACTx DLL shipped with the IMS server adapter has a setting of 2 for event logging in IMS. This means that all warning and error messages are displayed, but none of the informational messages are displayed.

Modifying the ORXMFACTx DLL setting

The ORXMFACTx DLL setting can be modified to some other value. For example, to help trace a problem with a transaction in IMS, it can be changed to 6.

Customizing Orbix Maximum Segment Size

Overview

The Orbix runtime inside IMS sends client transaction data to the client adapter in a stream of segments. The maximum size of these segments can be customized.

This section discusses the following topics:

- [Customizing the maximum segment size](#)
 - [ORXMFACTx DLL setting](#)
 - [Modifying the ORXMFACTx DLL setting](#)
 - [Maximum segment size constraints](#)
-

Customizing the maximum segment size

Customizing the maximum segment size is done by modifying the `ORXMFACTx` DLL. This DLL contains an S390 Assembler `CSECT` that supplies the maximum segment size to the Orbix runtime.

ORXMFACTx DLL setting

The `ORXMFACTx` DLL shipped with the client adapter has a setting of `32760` for the maximum segment size. (This is 32K rounded down to be a multiple of eight.)

Modifying the ORXMFACTx DLL setting

The Orbix runtime in IMS builds up APPC segments of this size. For APPC, multiple segments of this size are used to transmit data. The 32K APPC limit for a single segment applies, but all the segments together can be more than 32K. Depending on your network definitions, these segments can be further broken up into smaller segments by VTAM and *chained* when they are transmitted.

The `ORXMFACTx` DLL setting can be modified to be some other value, if, for example, your installation has restrictions on the size of APPC buffers. For example, it might be changed to `4096` to meet an installation requirement.

Maximum segment size constraints

When choosing a value for the maximum segment size consider the following:

- The value must be a multiple of 8
- The minimum value is 32
- The maximum value is 32760
- The default value is 32760

Customizing Orbix APPC Function Timeout

Overview

The Orbix runtime in IMS uses APPC when communicating with the client adapter. It issues an `APPC send` to send data to the client adapter, and issues an `APPC receive` to receive data from the client adapter. The amount of time the Orbix runtime in IMS allows the `APPC receive` to wait before it times out can be customized.

This section discusses the following topics:

- [Customizing the APPC function timeout](#)
 - [ORXMFACTx DLL setting](#)
 - [Modifying the ORXMFACTx DLL setting](#)
 - [APPC function timeout restrictions](#)
-

Customizing the APPC function timeout

Customizing the APPC function timeout is done by modifying the `ORXMFACTx` DLL. This DLL contains an S390 Assembler `CSECT` that supplies the APPC function timeout to the Orbix runtime.

ORXMFACTx DLL setting

The `ORXMFACTx` DLL shipped with the client adapter has a setting of 5 for the APPC function timeout. (This value represents 5 minutes.)

Modifying the ORXMFACTx DLL setting

The `ORXMFACTx` DLL setting can be modified to some other value. If your installation considers any response that takes longer than 2 minutes to be a problem, the APPC function timeout can be changed to: 2.

APPC function timeout restrictions

When choosing a value for the APPC function timeout consider the following:

- The value must be in the range 0 to 1440.
- A value of 0 means no timeout.
- The value designates a timeout value in minutes.
- The default value is 5 minutes.

Customizing Orbix APPC Symbolic Destination

Overview

The Orbix runtime in IMS uses APPC when communicating with the client adapter. It issues an `APPC allocate` to initiate an APPC conversation with the client adapter. The `APPC allocate` must identify the client adapter as the target of the `allocate` request. An APPC symbolic destination is used to identify the client adapter. The symbolic destination can be customized.

This section discusses the following topics:

- [Customizing the APPC symbolic destination](#)
- [ORXMFACTx DLL setting](#)
- [Modifying the ORXMFACTx DLL setting](#)
- [ACCP symbolic destination restrictions](#)

Customizing the APPC symbolic destination

Customizing the APPC symbolic destination is done by modifying the `ORXMFACTx` DLL. This DLL contains an S390 Assembler `CSECT` that supplies the APPC symbolic destination to the Orbix runtime.

ORXMFACTx DLL setting

The `ORXMFACTx` DLL shipped with the client adapter has a setting of `ORXCLNT1` for the APPC symbolic destination.

Modifying the ORXMFACTx DLL setting

The `ORXMFACTx` DLL setting can be modified to some other value. If your installation has naming standards for symbolic destinations, it can be changed to, for example, `PRODCADP`.

ACCP symbolic destination restrictions

When choosing a value for the APPC symbolic destination consider the following:

- The default value is `ORXCLNT1`.
- The value must match the client adapter's `amtp_appc` plug-in `plugins:amtp_appc:symbolic_destination` configuration item setting. Refer to [“APPC destination” on page 144](#) for more information on the `amtp_appc` plug-in configuration setting.

- Refer to [“Defining an APPC Destination Name for the Client Adapter” on page 130](#) for more information on how to define a symbolic destination to APPC/MVS.

Customizing Orbix APPC Local LU

Overview

The Orbix runtime in IMS uses APPC when communicating with the client adapter. It issues an `APPC allocate` to initiate an APPC conversation with the client adapter. The `APPC allocate` must identify the local LU it uses to communicate with the client adapter's LU. The local LU can be customized.

This section discusses the following topics:

- [Customizing the APPC local LU](#)
- [ORXMFACx DLL setting](#)
- [Modifying the ORXMFACx DLL setting](#)
- [APPC local LU restrictions](#)

Customizing the APPC local LU

This is done by modifying the `ORXMFACx` DLL. This DLL contains an S390 Assembler `CSECT` that supplies the APPC local LU to the Orbix runtime.

ORXMFACx DLL setting

The `ORXMFACx` DLL shipped with the client adapter has the following setting for the APPC local LU: `IMSLU01`

Modifying the ORXMFACx DLL setting

This setting can be modified to be some other setting. If your installation has a different name for the Orbix runtime in IMS local LU, then it can, for example, be changed to: `OURIMSLU`.

APPC local LU restrictions

Note the following when choosing a value for the APPC local LU:

- The default value is `IMSLU01`.
- Refer to [“Associating an IMS LU with a specific IMS region” on page 73](#) for more information on where the IMS local LU is defined. This is the value that must be defined for the APPC local LU.

Part 4

Securing and Using the IMS Server Adapter

In this part

This part contains the following chapters:

Securing the IMS Server Adapter	page 163
Using the IMS Server Adapter	page 211

Securing the IMS Server Adapter

This chapter provides details of security considerations involved in using the IMS server adapter. It provides a review of general Orbix security implications and the relevant IMS, APPC, and OTMA security mechanisms. It describes the two security modes that the server adapter supports, with particular emphasis on how each mode affects the existing IMS security mechanisms.

In this chapter

The following topics are discussed in this chapter:

Security Configuration Items	page 164
Common Security Considerations	page 169
APPC-Based Security Considerations	page 171
OTMA-Based Security Considerations	page 177
IMS Server Adapter Security Modes	page 179
Choosing between OTMA and APPC Modes	page 182
Setting up APPC and OTMA Modes	page 183

Security Configuration Items

Overview

This section provides an example and details of how to configure the IMS server adapter to run with Transport Layer Security (TLS) enabled. The sample configuration includes an `is2` sub-scope that highlights the configuration items required to integrate with the IONA Security Framework (iSF) and, in particular, enable CSIV2-based authentication using the off-host iS2 service. The `is2` sub-scope also includes configuration items that allow you to deploy a fully standalone IMS adapter service.

Sample configuration

[Example 11](#) provides an overview of the configuration items used to enable security with the server adapter.

Example 11: *Sample Security Configuration for IMS Server Adapter (Sheet 1 of 3)*

```
binding:client_binding_list = [ "OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
  "OTS+POA_Coloc", "POA_Coloc", "OTS+GIOP+IIOP_TLS", "GIOP+IIOP_TLS", "OTS+GIOP+IIOP",
  "GIOP+IIOP" ];

iona_services
{
  plugins:iiop_tls:racf_keyring = "ORBXRING";

  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id =      "security_label";

  principal_sponsor:auth_method_data = ["label=iona_services"];

  # Semi-secure service policies
  policies:target_secure_invocation_policy:requires = ["NoProtection"];
  policies:target_secure_invocation_policy:supports = ["Confidentiality", "Integrity",
    "DetectReplay", "DetectMisordering", "EstablishTrustInTarget", "NoProtection"];
  policies:client_secure_invocation_policy:requires = ["NoProtection"];
  policies:client_secure_invocation_policy:supports = ["Confidentiality", "Integrity",
    "DetectReplay", "DetectMisordering", "EstablishTrustInTarget", "NoProtection"];
  ...
  imsa
  {
    ...
    orb_plugins = ["iiop_profile", "giop", "iiop", "iiop_tls",
```

Example 11: *Sample Security Configuration for IMS Server Adapter (Sheet 2 of 3)*

```

"local_log_stream", "ots", "wto_announce"];

...
is2
{
    # enable is2 authentication
    #

orb_plugins = ["iiop_profile", "giop", "iiop",
               "iiop_tls", "local_log_stream", "ots", "gsp"];

event_log:filters = ["IT_CSI=*", "IT_GSP=*", "IT_IIOP_TLS=*",
                    "IT_MFA=INFO_HI+WARN+ERROR+FATAL"];

binding:client_binding_list = ["OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
                               "OTS+POA_Coloc", "POA_Coloc", "CSI+OTS+GIOP+IIOP_TLS", "OTS+GIOP+IIOP_TLS",
                               "CSI+GIOP+IIOP_TLS", "GIOP+IIOP_TLS", "CSI+OTS+GIOP+IIOP", "OTS+GIOP+IIOP",
                               "CSI+GIOP+IIOP", "GIOP+IIOP"];

binding:server_binding_list =
    ["CSI+GSP+OTS", "CSI+GSP", "CSI+OTS", "CSI"];

# configure a standalone IMS Adapter Service that does not use the IFR
plugins:imsa:direct_persistence = "yes";
plugins:imsa:iiop_tls:port = "5105";
plugins:imsa:iiop:port = "5005";
plugins:imsa:repository_id = "type_info";

plugins:imsa:use_client_principal = "yes";
plugins:imsa:check_security_credentials = "yes";

initial_references:IT_SecurityService:reference = "IOR:000...";
policies:csi:auth_over_transport:target_supports =
    ["EstablishTrustInClient"];

# allow non-CSIV2 based requests to proceed for
# demonstrational purposes. Insert this config item
# to enforce CSIV2 authentication:
#
#     policies:csi:auth_over_transport:target_requires =
#         ["EstablishTrustInClient"];

policies:csi:auth_over_transport:server_domain_name =
    "IONA";

```

Example 11: *Sample Security Configuration for IMS Server Adapter (Sheet 3 of 3)*

```

        policies:csi:attribute_service:target_supports =
            ["IdentityAssertion"];
    };
};
...

```

Summary of `iona_services` configuration items

The following is a summary of the security-related configuration items associated with the `iona_services` scope:

<code>plugins:iiop_tls:racf_keyring</code>	Specifies the RACF keyring to be used as the source of X.509 certificates.
<code>principal_sponsor:use_principal_sponsor</code>	Indicates whether an SSL/TLS principal sponsor is to be used to specify an X.509 certificate.
<code>principal_sponsor:auth_method_id</code>	Indicates whether the X.509 certificate is contained in a RACF keyring or an HFS database.
<code>principal_sponsor:auth_method_data</code>	Specifies an alternative, custom X.509 certificate contained in a RACF keyring or HFS database.
<code>policies:target_secure_invocation_policy:requires</code>	Specifies the invocation policy required by the server for accepting secure SSL/TLS connection attempts.
<code>policies:target_secure_invocation_policy:supports</code>	Specifies the invocation policies supported by the server for accepting secure SSL/TLS connection attempts.
<code>policies:client_secure_invocation_policy:requires</code>	Specifies the invocation policy required by the client for opening secure SSL/TLS connections.
<code>policies:client_secure_invocation_policy:supports</code>	Specifies the invocation policies supported by the client for opening secure SSL/TLS connections.

Note: See the *Mainframe Security Guide* for more details of these configuration items.

Summary of iS2 configuration items

The following is a summary of the configuration items associated with the `iona_services:imsa:iS2` security plug-in:

<code>orb_plugins</code>	List of standard ORB plug-ins the IMS server adapter should load when running in secure mode.
<code>event_log:filters</code>	Specifies the types of events that the IMS server adapter logs in secure mode.
<code>binding:client_binding_list</code>	Specifies a list of potential client-side bindings.
<code>binding:server_binding_list</code>	Specifies a list of potential server-side bindings.
<code>plugins:imsa:direct_persistence</code>	Specifies the persistence mode adopted by the IMS server adapter service in secure mode. This is an optional item. <code>iiop_tls:port</code> is required if this is specified as <code>yes</code> .
<code>plugins:imsa:iiop_tls:port</code>	Specifies the TCP/IP port number that the IMS server adapter uses to listen for incoming secure requests. Valid values are in the range 1025–65535. This is an optional item. Default is 5106.
<code>plugins:imsa:iiop:port</code>	Specifies the TCP/IP port number that the IMS server adapter uses to listen for incoming insecure requests. Valid values are in the range 1025–65535. This is an optional item. Default is 5006.
<code>plugins:imsa:respository_id</code>	Specifies the type information source to use in secure mode. This source supplies the IMS server adapter with operation signatures as required. Valid values are <code>"ifr"</code> , <code>"type_info"</code> , and <code>"none"</code> . The default is <code>"ifr"</code> . Refer to “Type information mechanism” on page 59 for more information.

<code>plugins:imsa:type_info:source</code>	This value is used if <code>repository_id</code> is set to <code>"type_info"</code> . The <code>type_info:source</code> variable denotes the location of a <code>type_info</code> store from which the IMS server adapter can obtain operation signatures when running in secure mode. Refer to “type_info store” on page 60 for more information.
<code>plugins:imsa:use_client_principal</code>	Indicates whether the IMS server adapter should verify the client principal user ID with SAF before trying to start the target IMS transaction under that ID. The default is <code>no</code> .
<code>plugins:imsa:check_security_credentials</code>	Indicates whether the IMS server adapter should query the CSI received credentials for a user ID before defaulting to the GIOP Principal value, on receiving a client request.
<code>initial_references: IT_SecurityService:reference</code>	Specifies the IOR for the off-host Security service.
<code>policies:csi:auth_over_transport: target_supports</code>	Specifies that the target server supports receiving username/password authentication data from the client.
<code>policies:csi:auth_over_transport: target_requires</code>	Specifies that the target server requires the client to send username/password authentication data.
<code>policies:csi:auth_over_transport: server_domain_name</code>	Specifies the server’s CSIv2 authentication domain name.
<code>policies:csi:attribute_service: target_supports</code>	Specifies that the target server supports receiving propagated user identities from the client.

Common Security Considerations

Overview

This section provides details of common security considerations involved in using the IMS server adapter. These security considerations are relevant regardless of which protocol the server adapter is using to communicate with IMS.

This section discusses the following topics:

- [Orbix SSL/TLS](#)
- [iS2 integration](#)
- [Client authorization](#)
- [SAF plug-in](#)
- [Mapping client principal values to OS/390 user IDs](#)

Orbix SSL/TLS

Orbix provides transport layer security (TLS) that enables secure connectivity over IIOP. TLS includes authentication, encryption, and message integrity. As with all Orbix servers, you can configure the IMS server adapter to use TLS. See the *Mainframe Security Guide* for details on securing CORBA applications with SSL/TLS.

iS2 integration

The IONA security framework (iSF) provides a common security framework for all Orbix components in your system. This framework is involved at both the transport layer (using TLS) and the application layer (using the CORBA CSiv2 protocol and the IONA generic security plug-in (GSP)). At the application level, one of the following authentication credentials can be passed, using the CSiv2 protocol:

- username/password/domain name
- propagated username
- Single sign-on (SSO) token

You can configure the IMS server adapter to use CSI/GSP support. See the *Mainframe Security Guide* for details on iSF and integration with an off-host iS2 service.

Note: The GSP realm/role authorization functionality is not supported in Orbix Mainframe 6.0.

Client authorization

Even though Orbix Mainframe 6.0 does not support GSP authorization functionality, authorization checks can be performed against the client's Principal value. These authorization checks can be performed in the following ways:

- Using the SAF plug-in, which provides Principal-based access control. Refer to [“SAF plug-in” on page 170](#) for more details.
- As part of the IMS security mechanisms (for example, checking that the user is allowed to run the specified program). Refer to [“APPC-Based Security Considerations” on page 171](#) and [“OTMA-Based Security Considerations” on page 177](#) for more details.

The client's Principal value is a string that is passed as part of an Orbix request that identifies the user on the client side. If Orbix SSL/TLS has not been configured, this value cannot be authenticated in any way. Sophisticated client-side users could fabricate this value, and therefore gain access to server-side resources that those users would not otherwise be allowed to use.

SAF plug-in

This Orbix plug-in provides optional Principal-based access control, similar to that found in IONA's Orbix 2.3-based mainframe solutions. A server might accept or reject incoming requests, based upon a `CORBA::Principal` value in the request header. The value is treated as an OS/390 user ID and access is checked against an operation-specific SAF profile name. Access can therefore be controlled on a per-operation basis, or (using generic profiles) on a per-server basis. More details can be found in the `orbixhlq.DOC` PDS which is created as part of the software installation.

Mapping client principal values to OS/390 user IDs

For the purposes of checking access to IMS resources, the only translation that the server adapter performs between the client principal value and the OS/390 user ID is to convert lowercase letters to uppercase and to restrict the ID to no more than 8 characters. Long principal values from other platforms have their principals truncated to 8 characters. This means principals longer than 8 characters must have the first 8 characters match a valid OS/390 user ID. Principals with 8 characters or less in length must entirely match a valid OS/390 user ID.

APPC-Based Security Considerations

Overview

This section provides details of security considerations that are specific to using the APPC-based server adapter. It describes the various security modes that the APPC-based server adapter supports, with particular emphasis on how each mode affects the existing IMS security mechanisms.

This section discusses the following topics:

- [Overview of APPC \(LU 6.2 Protocol\)](#)
- [Characteristics of the APPC-based server adapter](#)
- [LU 6.2 conversation security levels](#)
- [Preventing unauthorized access](#)
- [Security for users already logged on](#)
- [Session-level verification](#)
- [APPCLU class profiles](#)
- [Restricting authorized use of LU names](#)
- [Extra IMS command for securing conversations](#)

Overview of APPC (LU 6.2 Protocol)

APPC is an implementation of the SNA LU 6.2 protocol for program-to-program communication across networks. An LU allocates a conversation to another LU and exchanges data with it. LU 6.2 defines a number of characteristics that can be established for a conversation. These include throughput, transactional behavior, and levels of security. APPC provides a set of programming interfaces that are used to construct programs that can send or receive LU 6.2 conversations.

Characteristics of the APPC-based server adapter

In version 4, IMS first provided an APPC transaction program that could act as an *inbound* (or receiver) LU. Its function is to accept data from an *outbound* LU, queue it as segments to the IMS message queue for scheduling as an IMS transaction, and then return any output segments from the transaction to the outbound LU.

The APPC-based Adapter has been constructed as an outbound LU. This means that it accepts data from CORBA clients on a TCP/IP network, sends that data on to the IMS LU via an LU 6.2 conversation, and then returns the data it receives from IMS back to the TCP/IP network.

LU 6.2 conversation security levels

The LU 6.2 protocol, of which APPC/MVS is an implementation, defines three levels of conversation security:

`security_none` No user identification is passed during the conversation. Access to resources on the receiving (inbound) side is limited to those that are universally available. In RACF terms, this means that the only resources used are those protected by profiles with a UACC other than `NONE`.

When the receiving side is an IMS LU, this results in the IMS transaction being scheduled under no user, with a blank value in the `userid` field of the I/O PCB.

`security_same` The identity of the initiating (outbound) user is passed when starting the conversation. On the receiving side, access is granted to all resources for which that user has appropriate permissions. Essentially, the program running on the receiving side is expected to have the same access privileges as if the user had logged in directly. No authentication of the user is performed, because the inbound side of the conversation is expected to pass an `already verified` flag, to indicate that the user's identity has already been checked.

The IMS server adapter attempts to use `security_same` when allocating its conversations with the APPC/IMS inbound transaction program. This allows the IMS transaction that is being scheduled to be associated with a particular user, so that existing IMS mechanisms can be used for resource-access checking (for example, `TIMS` RACF class profiles) and auditing (for example, the `userid` value in the I/O PCB). However, `security_none` might be used if VTAM refuses `already verified` connections to the LU. This is explained in more detail later in [“Security for users already logged on” on page 173](#).

`security_pgm` The initiating side sends a user identity value to be used on the receiving side. This is not necessarily the identity of the user initiating the conversation. The program on the receiving side is expected to run with the privileges of the specified user. For authentication purposes, the inbound side must also send an associated password value for the user, which is checked via RACF services.

A conversation using `security_pgm` is not possible with the IMS server adapter, because it has no access to passwords for its clients.

Note: Although the LU 6.2 protocol can be used for network communication, the IMS server adapter is only intended to be run on the same machine as the IMS region with which it is communicating.

Preventing unauthorized access

Generally, in a network environment, it is a ridiculous idea that a client should be authenticated by a server merely on the basis that it claims to have been `already-verified`. After all, it is possible for a sophisticated user on a workstation to forge any desired identity merely by fabricating the appropriate LU 6.2 protocol exchanges with the OS/390 host. Therefore, to prevent such unauthorized access, OS/390 provides a way to specify what information must be passed, to connect to a particular LU. This is done by specifying the `SECACPT=CONV` key in the `APPL` definition for the VTAM ACB associated with the LU.

When allocating a conversation with an LU defined in this way, the initiating LU must provide a user ID and password: the `already-verified` indicator is not accepted. If the required data is not passed, VTAM permits the connection, but the level of conversation security is reduced to `security_none`, and only universally available resources are accessible on the receiving side. Therefore, to get access to resources on the inbound side, the outbound user must provide a password.

Security for users already logged on

Consider the special case of a user already logged onto the host, who is using APPC/MVS to communicate with an LU on the same OS/390 host. This is known as an `LU=LOCAL` conversation. In this case, the security information that is passed between the two sides for a `security_same` conversation is contained entirely within APPC/MVS itself: the outbound LU extracts the user's identity automatically for presentation to the inbound LU.

There is no opportunity for the user to insert a fabricated identity. In such cases, there should be no need for APPC/MVS to enforce the password requirement: the user has already provided a password to gain access to the host in the first place.

When running on OS/390, the IMS server adapter is in a similar situation to a logged-on user. If it initiates conversations to the IMS LU under its own identity (the default mode), that identity has either been verified when the user that started the server adapter logged on (if the server adapter is submitted as a job or started interactively), or it has been assigned by the security product when the work is started by an operator (if the server adapter is run as a started task). Even if the server adapter is initiating conversations under the identity of its clients, with the `plugins:imsa:use_client_principal` configuration item set to `yes`, it can only do that if it is running under a user ID that has been given authority to do that. Additionally, it must have gone through at least one of the checks already mentioned, to run under that user ID.

Session-level verification

A secure but efficient APPC environment is, therefore, one that permits only `security_pgm` conversations from remote machines, but which allows `security_same` for `LU=LOCAL` conversations. In fact, prior to OS/390 V1R3, this is what APPC/MVS provided for LUs defined with `SECACPT=CONV`, because VTAM did not enforce the `SECACPT=CONV` specification for `LU=LOCAL` conversations. Since OS/390 V1R3, however, this is enforced¹, so an alternate means of allowing `security_same` for `LU=LOCAL` conversations must be used. This is accomplished on OS/390, using *session-level verification*. Session-level verification introduces the concept of a *session key* that can be used instead of a password for conversations between two specific LU names only. If `VERIFY=OPTIONAL` is coded on the `APPL` definition of the VTAM ACB for an LU, VTAM allows a `security_same` conversation to be established, provided the other LU can correctly respond to a demand for the session key that has been defined for these two LU names. On OS/390, these session keys are maintained by RACF in `APPCLU` class profiles.

APPCLU class profiles

`APPCLU` class profiles have names that take the following form:

```
'networkid.local-lu-name.partner-lu-name'
```

1. Refer to the IBM publication *OS/390 V1R3.0 MVS Conversion Notebook, GC28-1747* for more details.

They contain information to be used by APPC/MVS on one side of a conversation. Even if both LUs are on the same OS/390 host, each LU examines a different profile, because each side of the conversation considers itself to be the local LU.

For example, if an LU named `OUTLU` initiates a conversation with an LU named `INLU` that has `SECACPT=CONV` and `VERIFY=OPTIONAL` coded on its ACB, APPC/MVS on the inbound side determines the correct session key by consulting the `networkid.INLU.OUTLU` APPCLU profile. On the outbound side, when challenged for a session key, the initiating APPC/MVS consults the `networkid.OUTLU.INLU` profile, for the key value to return. VTAM, on the inbound side, permits the conversation to proceed as `security_same`, only if the key values in the two profiles match and `CONVSEC(ALREADYV)` is also coded in the inbound APPCLU profile.

Restricting authorized use of LU names

Additionally, because *session-level verification* is performed on the basis of LU name rather than on the basis of user name, it is necessary to restrict the users that are authorized to use those particular LU names. This is done via the RACF `APPCPORT` class. By defining a profile in this class with the name of an LU, you can use its access list to control who can initiate or accept APPC conversations with that LU on this system.

Extra IMS command for securing conversations

The IMS support for APPC includes an extra command for securing LU 6.2 conversations into IMS. The options are as follows:

<code>/SECURE APPC FULL</code>	RACF calls are made to check access to transactions, using the <code>TIMS</code> class. Additionally, a complete security environment for the user ID that is provided by the IMS server adapter is set up in the dependent region. This is the recommended option.
<code>/SECURE APPC CHECK</code>	The RACF checks are made, but the security environment is not cloned.
<code>/SECURE APPC NONE</code>	RACF security calls are not made. Users are allowed to invoke transactions, regardless of the corresponding RACF <code>TIMS</code> class profile.

`/SECURE APPC PROFILE` Either `FULL`, `CHECK`, or `PROFILE` is used, depending on the value specified in the APPC transaction program profile data set (typically called `SYS1.APPCTP`) for the transaction. It is not necessary to define separate TP profiles for each IMS transaction to use the IMS server adapter, and IONA does not recommend doing so.

OTMA-Based Security Considerations

Overview

OTMA security is provided by the `IMSXCF.group.member` and `IMSXCF.OTMACI` resources in the RACF facilities class.

Refer to the IBM publications *OTMA Guide and Reference, SC26-8743* and *OTMA C/I, SC26-8743-01* for details about how to set up the RACF classes for OTMA. The IBM redbook *IMS V6 Security Guide, SG24-5363* also provides details about how to set up the RACF security for OTMA and OTMA C/I.

This section discusses the following topics:

- [Joining the XCF group](#)
- [Setting the OTMA security level](#)

Joining the XCF group

The user ID under which the server adapter is started is used for security when joining the XCF group. If the `plugins:imsa:use_client_principal` configuration item is set to `no`, this user name is also used for each transaction invocation. If `plugins:imsa:use_client_principal` is set to `yes`, the client Principal is used as the user ID for each transaction invocation. The group name used for each transaction is read from SAF by the OTMA C/I. Access to transactions is controlled using the standard `TIMS` RACF class.

If `IMSXCF.group.member` is defined in the `FACILITY` class (where `group` is the XCF group for IMS, and `member` is the member name of IMS or the IMS server adapter or both), and if IMS security is not set to `NONE`, the user token must be a valid SAF user with at least `READ` access. The user token can be either the client Principal or the user ID that is used to start the server adapter, depending on whether the `-S` parameter is used.

If `IMSXCF.OTMACI` is defined in the `FACILITY` class, the user ID under which the server adapter is started must have at least `READ` access.

Any transactions not listed in the `TIMS` class are allowed using `/SECURE OTMA`, regardless of the option that is set.

Setting the OTMA security level

IMS supports the following commands to set the OTMA security level:

- | | |
|-----------------------------------|--|
| <code>/SECURE OTMA FULL</code> | The user token passed to IMS is verified, using SAF. If the <code>plugins:imsa:use_client_principal</code> configuration item is set to <code>yes</code> , this token is the client Principal; otherwise, it is the user ID under which the server adapter was started. Full security is the recommended option in a production environment. |
| <code>/SECURE OTMA PROFILE</code> | This provides the same level of security as <code>FULL</code> in the case of the IMS server adapter. |
| <code>/SECURE OTMA NONE</code> | The user token passed to IMS by the IMS server adapter is not validated. This is useful for development environments where full security is not always required. |

IMS Server Adapter Security Modes

Overview

The IMS server adapter supports two modes of operation with regard to security. The two modes are distinguished by which user identity is made available to IMS and to either APPC or OTMA.

This section discusses the following topics:

- [Determining the user ID](#)
- [Default mode](#)
- [use_client_principal mode for APPC-based adapters](#)
- [use_client_principal mode for OTMA-based adapters](#)
- [check_security_credentials iS2 option](#)

Determining the user ID

For every incoming client request, the IMS server adapter has two user IDs at its disposal:

- Its own user ID (that is, the ID under which the server adapter executable is running).
- The client user ID (that is, the Principal value converted to uppercase, and potentially truncated, to match the requirements of OS/390).

By default, the client user ID is the string value that is passed in the GIOP Principal field. For GIOP 1.2 or later versions, the `CORBA::Principal` field has been deprecated; however, as an alternative, Orbix can be configured to pass the Principal user ID in a special service context that is marshaled by the GIOP plug-in.

For installations that have been configured to use the iS2 service, the client user ID can be obtained from the CSI received credentials. If a user ID is not available in the security credentials, the GIOP Principal value is used instead. See [“check_security_credentials iS2 option” on page 180](#) for more details.

The Orbix IMS security mode that is chosen when starting the server adapter determines the user ID that is used for security.

Default mode

In the default mode, IMS and either APPC or OTMA use the IMS server adapter's user ID to verify access to the LU names, to the IMS region, to the IMS transaction, to PSBs and databases, and so on. This means that the server adapter's user ID must be given access to not just the APPC or OTMA resources, but also to every IMS resource that any potential client can access. Otherwise, an incoming request might fail, even though the client itself has access to every IMS resource it needs.

use_client_principal mode for APPC-based adapters

If you set the `plugins:imsa:use_client_principal` configuration item to `yes`, the APPC-based server adapter assumes the identity of the client before initiating the APPC conversation. This means that the client Principal is used for the APPC and IMS checks. In this mode, the server adapter is more transparent, and security checking is similar to that of a user working from a 3270 terminal. Although users now require access to the server adapter LU and the IMS LU, the remaining resources to which users need access should be the same as if they had signed in from a terminal.

The `use_client_principal` mode works by having the server adapter use the services of OS/390 to establish a thread-level security environment with the identity of the client for portions of its processing. This causes APPC and IMS to use that user ID for their checks. This does incur some extra overhead on each client request compared to the default mode.

Because of the requirements of the `pthread_security_np()` service, the server adapter must be either run as super-user or given access to the `BPX.SERVER RACF FACILITY` class profile and have its executable placed in a controlled library. Refer to [“Additional Requirements for IMS Protocol Plug-Ins” on page 102](#) for more details.

use_client_principal mode for OTMA-based adapters

If you set the `plugins:imsa:use_client_principal` configuration item to `yes`, the client Principal is used as the user ID for each transaction invocation on the OTMA C/I. The same runtime requirements apply as for the APPC version of the server adapter. Additionally, the OTMA-based server adapter must be run APF-authorized, regardless of whether it is running on native OS/390 or UNIX System Services.

check_security_credentials iS2 option

If you set the `plugins:imsa:check_security_credentials` configuration item to `yes`, the IMS server adapter queries the CSI received credentials for a user ID before defaulting to the GIOP Principal value, on receiving a client

request. Assuming that the `plugins:imsa:use_client_principal` configuration item is set to `yes`, it then attempts to verify that this user ID is authorized to run the specified transaction.

When the `plugins:imsa:check_security_credentials` is set to `yes`, the client access ID that is used is one of the following (in order of priority):

1. The propagated user ID that is passed using the identity assertion mechanism.
2. The GSSUP token username.
3. The GIOP Principal.

If a user ID is not available from any of these sources, the client request is rejected.

Note: The `plugins:imsa:check_security_credentials` item only takes effect if the Orbix domain has been configured to use iS2. See the *Mainframe Security Guide* for more details.

Choosing between OTMA and APPC Modes

Overview

This section discusses security-related issues relevant to OTMA and APPC. It discusses the following topics:

- [Transparency versus non-authentication](#)
 - [Administrative overhead and requirements](#)
-

Transparency versus non-authentication

The `use_client_principal` security mode allows for the most transparency, because it brings the identity of the Orbix client all the way into the IMS region, for authority checking and auditing. However, because Orbix clients are not yet authenticated, you might want to run in the default mode.

Administrative overhead and requirements

The administrative overhead of each mode is approximately the same. Choosing the `use_client_principal` mode means having to permit all potential clients access to the APPC or OTMA resources that are needed to conduct conversations. Choosing the default mode, however, means having to permit the IMS server adapter user ID access to all the IMS resources that its clients might need.

If your installation already has a RACF group profile that allows selected user IDs to have global access to IMS resources (such as those of other IMS regions or IMS administrators), it is probably easier to add the server adapter to that group and run in the default mode.

Setting up APPC and OTMA Modes

Overview

This section summarizes the steps involved in setting up security for both APPC-based and OTMA-based server adapters. It discusses the following topics:

- [Summary of steps for APPC-based adapters](#)
- [Summary of steps for OTMA-based adapters](#)

Summary of steps for APPC-based adapters

The following is a summary of the APPC-specific steps involved in setting up a secure environment that lacks only the authentication of the server adapter's clients:

Step	Action
1	Define unique LU names for the server adapter and for IMS. Use RACF <code>APPCPORT</code> profiles to restrict the use of these LU names. Use the <code>-L</code> argument when starting the IMS server adapter to specify the server adapter's LU name. Specify the IMS LU name to IMS by using the ID of the region as a scheduler name for the LU in <code>SYS1.PARMLIB(APPCPMxx)</code> .
2	Define VTAM APPLs for the IMS server adapter and IMS LUs with <code>SECACPT=CONV</code> and <code>VERIFY=OPTIONAL</code> , to ensure authenticated conversations with network users.
3	If you are using OS/390 V1R3 or later versions, or you want to ease the eventual migration to that release, define RACF <code>APPCLU</code> class profiles for each side of the conversation that include identical session keys and <code>CONVSEC(ALREADYV)</code> . These profiles should specify <code>UACC(NONE)</code> to protect the session-key values from being exposed to unauthorized users.

Step	Action
4	Specify <code>APPCSE=F</code> as an IMS start-up parameter, or issue <code>/SECURE APPC FULL</code> on a running IMS system. This ensures that existing IMS resource-access checks are made against the user ID presented by the server adapter. Using the <code>use_client_principal</code> option means that this is the (unauthenticated) Principal of the client; otherwise, it is the IMS server adapter user ID.
5	If Orbix SSL/TLS for OS/390 is used with the IMS server adapter, you can also authenticate client certificates by setting certificate constraints in the server adapter's configuration.

Summary of steps for OTMA-based adapters

The following is a summary of the OTMA-specific steps involved in setting up a secure environment that lacks only the authentication of the IMS server adapter's clients:

Step	Action
1	Define the <code>IMSXCF.group.member</code> and <code>IMSXCF.OTMACI</code> resources in the RACF facilities class. If you do not define these, universal access is assumed by OTMA.
2	If the clients (that is, Principal values) have valid RACF user IDs, use the <code>use_client_principal</code> option of the IMS server adapter, and use the usual <code>TIMS</code> and <code>GIMS</code> RACF classes to control access to IMS transactions.
3	If Orbix SSL/TLS for OS/390 is used with the IMS server adapter, you can also authenticate client certificates by setting certificate constraints in the server adapter's configuration.

Mapping IDL Interfaces to IMS

This chapter provides information on how an IMS server adapter exposes IMS transactions as CORBA servers. It details the role that the mapping file plays in mapping CORBA operations and attributes for a given interface to a target transaction. It also details the role of the type information source (IFR or type_info store) in marshalling data from a client request.

In this chapter

This chapter discusses the following topics:

The Mapping File	page 186
Using the IFR as a Source of Type Information	page 192
Using type_info store as a Source of Type Information	page 202

The Mapping File

Overview

This section describes how the mapping file is used by the IMS server adapter. It also describes the contents of this file and how it can be generated using the Orbix IDL compiler.

In this section

This section discusses the following topics:

Characteristics of the Mapping File	page 187
Generating a Mapping File	page 189

Characteristics of the Mapping File

Overview

This subsection describes the mapping file, its format, how it supports IDL attributes, and its relationship with type information sources.

This subsection discusses the following topics:

- [Description](#)
- [Mapping file format](#)
- [Support for IDL attributes](#)

Description

The mapping file is a simple text file that determines what interfaces and operations the IMS server adapter supports, and the transaction names to which it should map each operation. The file is read when the IMS server adapter starts, and can be written or re-read during the server adapter operation by using the `MappingGateway` interface or the `itadmin mfa` commands. Refer to [“Using the MappingGateway Interface” on page 222](#) for more details.

Mapping file format

Each mapping entry in the file is specified as a tuple that specifies the following:

```
(interface name, operation name, IMS transaction name)
```

Tuples can span lines. All white space (including blanks embedded in names) is ignored.

In the tuples, if an IDL interface is scoped within a module or modules, the module name or names must then be included in the interface name. The module names are separated from each other and from the interface name with `/` characters. The interface name therefore has the following layout if it is scoped within two modules:

```
module_name/module_name/interface_name.
```

Support for IDL attributes

Attributes of IDL interfaces are supported by using `_get_attribute` and `_set_attribute` to read and write a particular attribute. For example, consider the `Simple` IDL:

```
module Simple {  
    interface SimpleObject  
    {  
        void  
        call_me();  
    };  
};
```

The following file maps the operation `call_me` on the `SimpleObject` interface to the IMS transaction named `SIMPLESV`:

```
(Simple/SimpleObject, call_me, SIMPLESV)
```

If the `SimpleObject` interface had a read-only attribute; for example, `something` (which it does not have in the sample application supplied by IONA), it needs an entry as follows in the mapping file:

```
(Simple/SimpleObject, _get_something, SIMPLESV)
```

Because the `something` attribute of the `SimpleObject` interface is specified as read-only in the IDL file, no `_set_something` operation is necessary.

Generating a Mapping File

Overview

An IDL compiler plug-in is available, called `mfa`, that is used to generate IMS server adapter mapping files.

This subsection discusses the following topics:

- [Adapter mapping file versus other mapping files](#)
- [Sample IDL](#)
- [Generating mapping files on OS/390 UNIX System Services](#)
- [Generating mapping files on native OS/390](#)
- [Making runtime modifications to mappings](#)

Adapter mapping file versus other mapping files

The IMS server adapter mapping file is completely unrelated to the mapping file used by the COBOL and PL/I IDL compilers. The IMS server adapter mapping file is used by the server adapter to select which transaction to run inside IMS, while the mapping file used by the COBOL and PL/I IDL compilers changes the names of specific items of source code generated by the IDL compiler.

Sample IDL

The code samples for generating an IMS server adapter mapping file are based on `Simple` IDL:

```
module Simple {
    interface SimpleObject
    {
        void
        call_me();
    };
};
```

Generating mapping files on OS/390 UNIX System Services

To generate a mapping file on OS/390 UNIX System Services, run the following command:

```
idl -mfa:-tSIMPLESV simple.idl
```

The `-t` parameter specifies the transaction that is run inside IMS for each IDL operation.

Refer to [“Mapping file format” on page 187](#) for details of the format of the mapping file generated.

Generating mapping files on native OS/390

The following is an example of JCL you can use to generate a mapping file on native OS/390:

```
//MAPFILE JOB (),
//          CLASS=A,
//          MSGCLASS=X,
//          MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID,
//          REGION=0M,
//          TIME=1440
//*
//          JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
//          INCLUDE MEMBER=(ORXVARS)
//*
/** Make the following changes before running this JCL:
/**
/** 1. Change 'SET DOMAIN='DEFAULT@' to your configuration
/**    domain name.
/**
//          SET DOMAIN='DEFAULT@'
/**
/** Generate an operation mapping file IMS Server Adapter
/**
//IDLMAP EXEC ORXIDL,
//          SOURCE=SIMPLE,
//          IDL=&ORBIX..DEMOS.IDL,
//          IDLPARM='-mfa:-tSIMPLESV'
//IDLMFA DD DISP=SHR,DSN=&ORBIX..DEMOS.IMS.MFAMAP
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR
```

The `-t` parameter specifies the transaction that is run inside IMS for each IDL operation.

Note: If the `-mfa` option is specified to the Orbix IDL compiler, the `IDLMFA DD` statement defines the PDS used to store the generated IMS server adapter mapping file.

Refer to [“Mapping file format” on page 187](#) for details of the format of the mapping file generated.

Making runtime modifications to mappings

An IMS server adapter caches mapping files internally during execution. This cache can be modified allowing mappings to be added, changed, or deleted. This functionality is exposed by the `itadmin mfa` command (refer to [“Using the MappingGateway Interface” on page 222](#) for a complete list of `itadmin mfa` commands). The syntax is as follows:

```
mfa
  add      -interface <name> -operation <name> <mapped value>
  change   -interface <name> -operation <name> <mapped value>
  delete   -interface <name> -operation <name>
```

The contents of this internal cache can be re-written (using `mfa save`) to file, to ensure that the mapping file is kept up-to-date. Functionality to reload or switch is also provided. The syntax is as follows:

```
mfa
  reload
  save   [<mapping_file name>]
  switch <mapping_file name>
```

Using the IFR as a Source of Type Information

Overview

This section describes how the IFR can be used as the source of type information by the IMS server adapter.

In this section

This section discusses the following topics:

Introduction to Using the IFR	page 193
Registering IDL interfaces with the IFR	page 195
Informing IMS Server Adapter of a New Interface in the IFR	page 198
Using an IFR Signature Cache File	page 200

Introduction to Using the IFR

Overview

This subsection introduces how the IFR can be used to supply type information to the IMS server adapter. It details how interfaces can be registered with the IFR, and the operation of the server adapter when using the IFR. It also describes how an IFR cache can be employed to improve performance.

This subsection discusses the following topics:

- [Description of the IFR](#)
- [Configuring the IFR](#)
- [Operation of IFR when no IFR signature cache file is specified](#)
- [Steps for using the IFR](#)

Description of the IFR

The IDL for the interfaces and operations specified in the mapping file must be available to the IFR server that the IMS server adapter uses. This information is required by the server adapter to marshal a request from a client. Therefore, IDL for supported interfaces must be added to the IFR. The steps for doing this are detailed below. To improve performance the IFR can be used with an optional IFR signature cache file.

Configuring the IFR

If you want to use the IFR you must ensure that the appropriate configuration variables are set. Additionally, if you want to use an IFR signature cache file, the relevant configuration variable must also be set. Refer to [“IFR signature cache file” on page 60](#) for more information.

Operation of IFR when no IFR signature cache file is specified

The server adapter contacts the IFR during start-up and attains operation signatures for operations defined in the mapping file. If an operation signature changes (for example, changing the return type from `void` to `float`) and the server adapter is notified (for example, if `itadmin mfa refresh` is called), it contacts the IFR to retrieve this modified signature.

If you want to use the IFR signature cache file refer to [“Using an IFR Signature Cache File” on page 200](#).

Steps for using the IFR

To use the IFR follow these steps:

Step	Action
1	Register IDL interfaces with the IFR. Refer to “Registering IDL interfaces with the IFR” on page 195 for further details.
2	Inform the IMS server adapter that the contents of the IFR have been modified. Refer to “Informing IMS Server Adapter of a New Interface in the IFR” on page 198 for more details.

Registering IDL interfaces with the IFR

Overview

This subsection describes how to register IDL interfaces with the IFR. It discusses the following topics:

- [Sample IDL](#)
- [Registering IDL on native OS/390](#)
- [Registering IDL on OS/390 UNIX System Services](#)
- [Specifying a -ORB argument](#)

Sample IDL

The code samples for registering IDL with the IFR are based on the following `Simple::SimpleObject` interface in the `simple.idl` file:

```
module Simple {
    interface SimpleObject
    {
        void
        call_me();
    };
};
```

Registering IDL on native OS/390

To add IDL (for example, the `SIMPLE` IDL member) to the IFR on native OS/390, use the following JCL:

```
//ADDIFR JOB (),
// CLASS=A,
// MSGCLASS=X,
// MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=0M,
// TIME=1440
//*
// JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
// INCLUDE MEMBER=(ORXVARS)
//*
/* Make the following changes before running this JCL:
/*
/* 1. Change the 'SET DOMAIN='DEFAULT@' to your configuration
/* domain name.
/*
// SET DOMAIN='DEFAULT@'
/*
/* Add an interface to the IFR
/*
//IDLMAP EXEC ORXIDL,
// SOURCE=SIMPLE,
// IDL=&ORBIX..DEMOS.IDL,
// IDLPARAM='-R'
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR
```

Registering IDL on OS/390 UNIX System Services

To add IDL (for example, the `simple.idl` file) to the IFR on OS/390 UNIX System Services, use the following command:

```
$ idl -R simple.idl
```

Specifying a -ORB argument

When registering IDL with the IFR, the `idl -R` command invokes an IDL back end that acts as a CORBA client to the IFR server. The client sends the IDL definitions by invoking CORBA calls on the IFR. Therefore, you might want to specify an ORB argument that can be used in the client's `ORB_init()` call before it communicates with the IFR. For example, to specify a different Orbix domain name on OS/390 UNIX System Services, enter the following command:

```
idl -R:--ORBdomain_name=domain2
```

Informing IMS Server Adapter of a New Interface in the IFR

Overview

After you add an interface to the IFR, the IMS server adapter must be notified for the updates to take effect. If adding support for a new interface or operation, the `itadmin mfa add` command can be used. In addition to creating a new binding between operation and IMS transaction in the mapping file, it also causes the IMS server adapter to contact the IFR to retrieve the operation signature for the new operation.

This subsection discusses the following:

- [Informing the server adapter of a new IDL interface on native OS/390](#)
- [Informing the server adapter of a new IDL interface on OS/390 UNIX System Services](#)
- [Notifying the server adapter of modifications to the IFR](#)

Informing the server adapter of a new IDL interface on native OS/390

To inform the IMS server adapter that the `SimpleObject` interface (see [“Sample IDL” on page 205](#) for an example) has been added to the IFR on native OS/390, use the following JCL:

```
//ADDMFA JOB (),
// CLASS=A,
// MSGCLASS=X,
// MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=0M,
// TIME=1440
/**
// JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
// INCLUDE MEMBER=(ORXVARS)
/**
/** Make the following changes before running this JCL:
/**
/** 1. Change 'SET DOMAIN='DEFAULT@' to your configuration
/** domain name.
/**
// SET DOMAIN='DEFAULT@'
/**
/** Add an interface mapping to the IMS Adapter
/**
//IMSADD EXEC ORXADMIN,
// PARM='-ORBname iona_services.imsa'
```

```
//SYSIN DD *
mfa add \
  -interface Simple/SimpleObject \
  -operation call_me \
  SIMPLESV
/*
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR
```

Informing the server adapter of a new IDL interface on OS/390 UNIX System Services

To inform the IMS server adapter that the `SimpleObject` interface (see [“Sample IDL” on page 205](#) for an example) has been added to the IFR on OS/390 UNIX System Services, use the following command:

```
$ itadmin -ORBname iona_services.imsa mfa add -interface
Simple/SimpleObject -operation call_me SIMPLESV
```

Notifying the server adapter of modifications to the IFR

The `itadmin mfa refresh` command is used to notify the IMS server adapter that an already supported operation signature has changed. It causes the IMS server adapter to contact the IFR and retrieve the updated operation signature and place this in its internal cache.

You can also use `refreshInterface()` or `refreshOperation()`. These functions are available via the `MappingGateway` interface and can be used to refresh the server adapter’s internal cache of operation signatures by contacting the IFR. This requires that a corresponding entry exist for the operation(s) in the mapping file.

Using an IFR Signature Cache File

Overview

This subsection describes how an IFR signature cache file can be used in conjunction with the IFR to improve performance of the IMS server adapter. It discusses the following topics:

- [Prerequisites to using the IFR signature cache file](#)
- [First run of the server adapter after configuration](#)
- [Subsequent runs of the server adapter](#)
- [Runtime modifications to the IFR](#)
- [Updating an IFR signature cache file](#)

Prerequisites to using the IFR signature cache file

Before you use a signature cache file you must specify the name of the signature cache file you want to use, in the `plugins:imsa:ifr:cache` configuration item in the `iona_services:imsa` configuration scope. Refer to [“IFR signature cache file” on page 60](#) for more details.

First run of the server adapter after configuration

When the server adapter is started after this configuration item is set, a new signature cache file is generated with this name, and the contents of the IFR are saved to it. If an operation signature is not available for an operation defined to the IMS server adapter via the mapping file, a warning message is output. For example, the warning message for an IDL interface called `Simple/SimpleObject` with a single operation called `call_me` is similar to the following:

```
Tue, 03 Dec 2002 12:35:30.000000 [MYMACHINE:16777601]
(IT_MFA:100) W - synchronization problem occurred for mapping
(Simple/SimpleObject,call_me) - unable to obtain type
information for the operation
```

Subsequent runs of the server adapter

With subsequent runs of the server adapter the IFR is not contacted during start-up. Instead it reads the list of operation signatures directly from the signature cache file. This should lead to an improvement in how long it takes to start the server adapter, especially if you need to start multiple server adapters simultaneously. This means the server adapters can be ready and available more quickly for client requests.

Runtime modifications to the IFR

During runtime, the IMS server adapter can contact the IFR to load or refresh an operation entry. Upon shutdown, the server adapter updates the signature cache file with the operation signatures it has used.

Note: The IFR signature cache file is only ever accessed twice. First, it is first accessed in read mode, during start-up. This boosts performance by preventing the IFR being contacted initially. Second, it is accessed in write mode, during shut-down. This dumps the operation signatures used by the server adapter to a signature cache file, so that this may be used when the server adapter is restarted.

Updating an IFR signature cache file

If type information subsequently changes in the IFR, you can update the information in the signature cache file in either of the following ways:

- `refreshInterface()` or `refreshOperation()`

If you are using the IFR signature cache file, either or both of these can be used on the `MappingGateway` interface, to consult the IFR and update the cached IFR operation signatures in-memory in the IMS server adapter with a specified interface or operation (or both).

- Stop the IMS server adapter, delete the IFR signature cache file and restart the server adapter.

When the server adapter is restarted it automatically uploads the operation signatures from the IFR into the IFR signature cache file. There is no need to inform the server adapter that the IFR signature cache file has been updated.

Using type_info store as a Source of Type Information

Overview

This section describes how a type_info store can be used as the source of type information by the IMS server adapter.

In this section

This section discusses the following topics:

Introduction to Using a type_info Store	page 203
Generating type_info Files using the IDL Compiler	page 205
Informing IMS Server Adapter of a new type_info Store File	page 207

Introduction to Using a type_info Store

Overview

This subsection describes the type_info store in terms of how the Orbix IDL compiler can be used to generate these files, the operation of the server adapter when using a type_info store, and how the store can be updated.

This subsection discusses the following topics:

- [Description](#)
- [Configuration](#)
- [Operation of IMS server adapter using type_info stores](#)
- [Steps for using a type_info store](#)

Description

The type_info store is one method of supplying IDL interface information to the IMS server adapter. It is an alternative approach to the IFR, and uses a file-based approach to represent operation signatures. The IMS server adapter can access these files at start-up and runtime, to obtain operation signatures, which it requires to marshal data from the CORBA client.

Note: If you are using a type_info store, the IMS server adapter does not require the IFR. This means that an IMS server adapter using a type_info store can be run in standalone mode, by configuring it to run in direct persistent mode.

Configuration

If you want to use a type_info source you must ensure that the appropriate configuration items are set. Refer to [“type_info store” on page 60](#) for more information.

Operation of IMS server adapter using type_info stores

The Orbix IDL compiler generates type_info files. When the IMS server adapter is started it accesses the type_info store and, for all operations for which an operation-to-transaction mapping entry exists, it loads the operation signatures into an internal cache. These operation signatures are required by the IMS server adapter to unmarshal operation arguments from a client request, and to marshal the response back.

During runtime, the `type_info` store can be updated dynamically (for example, to add support for a new interface, or to reflect a change in one or more operation signatures). This simply requires generating a new `type_info` file and then requesting the IMS server adapter to refresh its internal operation signature cache with the latest version in the `type_info` store.

Steps for using a `type_info` store

To use a `type_info` store do the following:

Step	Action
1	Use the IDL compiler to generate (or regenerate for subsequent additions or other modifications) a <code>type_info</code> file for IDL. Refer to “Generating <code>type_info</code> Files using the IDL Compiler” on page 205 for further details.
2	Inform the IMS server adapter of a new or modified interface. Refer to “Informing IMS Server Adapter of a new <code>type_info</code> Store File” on page 207 for further details.

Generating type_info Files using the IDL Compiler

Overview

This subsection describes the process of generating type_info store files. It discusses the following topics:

- [Sample IDL](#)
- [On OS/390 UNIX System Services](#)
- [On native OS/390](#)

Sample IDL

The code samples for generating a type_info file are based on `simple` IDL:

```
module Simple {  
    interface SimpleObject  
    {  
        void  
        call_me();  
    };  
};
```

On OS/390 UNIX System Services

To generate a type_info file on OS/390 UNIX System Services for the `simple` IDL, run the IDL compiler as follows:

```
idl -mfa:-inf simple.idl
```

This generates a type_info file named `simpleB.inf`.

Note: By default, the mfa backend generates type_info files with a suffix of `B`. This can be modified by editing the `MFAMappings` scope in `orbixhlq.CONFIG(IDL)`.

On native OS/390

To generate a `type_info` file on native OS/390 for the `simple` IDL, submit the following JCL to run the IDL compiler:

```
//ADDMFA JOB (),
// CLASS=A,
// MSGCLASS=X,
// MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=0M,
// TIME=1440
//*
// JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
// INCLUDE MEMBER=(ORXVARS)
//*
/* Make the following changes before running this JCL:
/*
/* 1. Change 'SET DOMAIN='DEFAULT@' to your configuration
/* domain name.
/*
// SET DOMAIN='DEFAULT@'
/*
/* Add an interface mapping to the IMS Server Adapter
/*
//IDLCBL EXEC ORXIDL,
// SOURCE=SIMPLE,
// IDL=&ORBIX..DEMOS.IDL,
// COPYLIB=&ORBIX..DEMOS.IMS.COBOL.COPYLIB,
// IMPL=&ORBIX..DEMOS.IMS.COBOL.SRC,
// IDLPARM='-mfa:-inf'
//IDLTYP EI DD DISP=SHR,DSN=&ORBIX..DEMOS.TYPEINFO
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR
```

This generates a `type_info` file named `orbixhlq.DEMOS.TYPEINFO(SIMPLEB)`.

Note: By default, the `mfa` backend generates `type_info` files with a suffix of `B`. This can be modified by editing the `MFAMappings` scope in `HLQ.ORBIX60.CONFIG(IDL)`.

Note: If the `-mfa:-inf` option is specified to the Orbix IDL compiler, the `IDLTYP EI DD` statement defines the PDS used to store the generated `type_info` file.

Informing IMS Server Adapter of a new type_info Store File

Overview

After you add a file to the type_info store, the IMS server adapter must be notified for the updates to take effect. If adding support for a new interface or operation, the `itadmin mfa add` command can be used. In addition to creating a new binding between operation and IMS transaction in the mapping file, it also causes the IMS server adapter to access the type_info store to retrieve the operation signature for the new operation.

This subsection discusses the following:

- [Informing the server adapter of a new IDL interface on OS/390 UNIX System Services](#)
- [Informing the server adapter of a new IDL interface on native OS/390](#)
- [Notifying the server adapter of modifications to the type_info store](#)

Informing the server adapter of a new IDL interface on OS/390 UNIX System Services

To inform the IMS server adapter that the `SimpleObject` interface (see [“Sample IDL” on page 205](#) for an example) has been added to the type_info store on OS/390 UNIX System Services, use the following command:

```
$ itadmin -ORBname iona_services.imsa mfa add -interface  
Simple/SimpleObject -operation call_me SIMPLESV
```

Informing the server adapter of a new IDL interface on native OS/390

To inform the IMS server adapter that the `SimpleObject` interface (see “Sample IDL” on page 205 for an example) has been added to the `type_info` store on native OS/390, use the following JCL:

```
//ADDMFA JOB ( ),
// CLASS=A,
// MSGCLASS=X,
// MSGLEVEL=(1,1),
// NOTIFY=&SYSUID,
// REGION=0M,
// TIME=1440
/**
// JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
// INCLUDE MEMBER=(ORXVARS)
/**
/** Make the following changes before running this JCL:
/**
/** 1. Change 'SET DOMAIN='DEFAULT@' to your configuration
/** domain name.
/**
// SET DOMAIN='DEFAULT@'
/**
/** Add an interface mapping to the IMS Adapter
/**
//IMSADD EXEC ORXADMIN,
// PPARM='-ORBname iona_services.imsa'
//SYSIN DD *
mfa add \
-interface Simple/SimpleObject \
-operation call_me \
SIMPLESV
/**
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR
```

Notifying the server adapter of modifications to the `type_info` store

The `itadmin mfa refresh` command is used to notify the IMS server adapter that an already supported operation signature has changed. It causes the IMS server adapter to access the `type_info` store and retrieve the updated operation signature and place this in its internal cache.

You can also use `refreshInterface()` or `refreshOperation()`. These functions are available via the `MappingGateway` interface and can be used to refresh the server adapter's internal cache of operation signatures by accessing the `type_info` store. This requires that a corresponding entry exists for the operation(s) in the mapping file.

Using the IMS Server Adapter

This chapter provides information on running and using the IMS server adapter. It provides details on how to start and stop the server adapter. It provides details on how to use the server adapter to act as a dynamic bridge to pass IDL-based requests into IMS. It describes how to use the MappingGateway interface of the server adapter. It explains how to run mapped IMS transactions in Wait-For-Input (WFI) mode as well as how to add a portable interceptor to the server adapter and gather accounting information in the server adapter. It also explains how to enable the server adapter to export object references for the interfaces it supports either to a file or to the Naming Service.

In this chapter

This chapter discusses the following topics:

Preparing the Server Adapter	page 213
Starting the Server Adapter	page 217
Stopping the IMS Server Adapter	page 219
Running Multiple Server Adapters Simultaneously	page 220

Using the MappingGateway Interface	page 222
Locating IMS Server Adapter Objects Using itmfaloc	page 225
WFI Support for IMS Transactions	page 228
Conversational Support	page 229
LTERM Propagation	page 233
Adding a Portable Interceptor to the IMS Server Adapter	page 234
Gathering Accounting Information in the Server Adapter	page 244
Exporting Object References at Runtime	page 250

Preparing the Server Adapter

Overview

This section describes what needs to be done to run the server adapter in prepare mode. It discusses the following topics:

- [Prerequisites to running the server adapter in prepare mode](#)
- [Running the IMS server adapter in prepare mode](#)
- [Sample JCL to run the IMS server adapter in prepare mode](#)
- [Location of IMS server adapter IORs](#)
- [The IT_MFA IOR](#)
- [The IT_MFA_IMSRAW IOR](#)
- [Sample configuration file](#)
- [Running the IMS server adapter on OS/390 UNIX System Services](#)

Prerequisites to running the server adapter in prepare mode

If you are using the IFR as the type information source (as is the default), you must ensure that the Orbix Mainframe services have been deployed. This process is described in the *Mainframe Installation Guide*.

If you are using a type_info store as the type information source, you can run the IMS server adapter in standalone mode, if you wish. This requires setting the IMS server adapter to run in direct persistent mode. In direct persistent mode, the IMS server adapter does not require the other Orbix Mainframe services.

Running the IMS server adapter in prepare mode

Run the server adapter in prepare mode. This generates two IORs and writes them to a file, which you can then include in your configuration file. A job to run the IMS server adapter in prepare mode is provided in the `PREPIMSA` member in the JCL library.

Sample JCL to run the IMS server adapter in prepare mode

This JCL contains the default high-level qualifier, so change it to reflect the proper value for your installation:

```
//PREPIMSA JOB ( ),
//          CLASS=A,
//          MSGCLASS=X,
//          MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID,
//          REGION=0M,
//          TIME=1440
//*
//          JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
//          INCLUDE MEMBER=(ORXVARS)
//*
/* Prepare the Orbix IMS Adapter
/* Make the following changes before running this JCL:
/*
/* 1. If you ran DEPLOY1 (or DEPLOYT) to configure in a domain
/*    other than the default, please ensure that dataset
/*    HLQ.ORBIX60.CONFIG(ORBARGS) has the domain
/*    name used by the DEPLOY1 (or DEPLOYT).
/*
//PREPARE EXEC PROC=ORXG,
//          PROGRAM=ORXIMSA,
//          PPARM='prepare -publish_to_file=DD:ITCONFIG(IORIMSA)'
//MFAMAPS DD DUMMY
//ORBARGS DD DSN=&ORBIX..CONFIG(ORBARGS),DISP=SHR
/*
/* Update configuration domain with IMS Adapter's IOR
/*
//ITCFG1 EXEC ORXADMIN
//SYSIN DD *
          variable modify \
              -type string \
              -value --from_file:3 //DD:ITCONFIG(IORIMSA) \
              LOCAL_MFA_IMS_REFERENCE
/*
//ORBARGS DD DSN=&ORBIX..CONFIG(ORBARGS),DISP=SHR
```

Location of IMS server adapter IORs

When complete, the IORs for the server adapter should be in `orbixhlq.CONFIG(IORIMSA)`. The file contains two IORs.

The IT_MFA IOR

One IOR is for `IT_MFA`. This is the IOR for the server adapter `MappingGateway` interface. The `orbixhlq.JCL(PREPIMSA)` JCL copies this IOR into the `LOCAL_MFA_IMS_REFERENCE` configuration item, which is found in the `orbixhlq.CONFIG` PDS, in the member that corresponds to your configuration domain name. (The default configuration domain name is `DEFAULT@.`) This IOR is used by `itadmin` to contact the correct server adapter. Refer to [“Using the MappingGateway Interface” on page 222](#) for more details.

The IT_MFA_IMSRAW IOR

The other IOR is for `IT_MFA_IMSRAW`. This is the IOR for the IMS server adapter `imsraw` interface. This IOR should be made available to client programs of the server adapter that want to use the `imsraw` interface. Refer to [“The IMS Server Adapter `imsraw` Interface” on page 23](#) for more details.

Sample configuration file

The following is an extract from a working configuration file for you to compare your file with.

Note: The position of the first quote is moved to the next line, directly preceding the start of the IOR. (Ellipses denote text omitted for the sake of brevity.)

```
...
LOCAL_MFA_IMS_REFERENCE =
    "IOR:000000000000002549444c3a696f6e612e636f6d2f49545f/
4c6f636174696f6e2f4c6f6361746f723a312e30000000000000001000000/
0000007e00010200000000056a756e6f00003a99000000253a3e0233311752/
5706c69636174656453696e676c65746f6e504f410007d3968381a39699000/
000000003000000010000001c00000001002041700000001000100010001/
1000000001000101090000001a000000040100000000000600000006000/
0000001c";
...
```

**Running the IMS server adapter
on OS/390 UNIX System Services**

You can also run the IMS server adapter in prepare mode from the UNIX System Services prompt. The command is as follows:

```
$ itimsa -ORBname iona_services.imsa prepare
```

The two IORs for `IT_MFA` and `IT_MFA_IMSRAW` are then displayed on the console. You can copy them to the appropriate places as described above. However, in general, it might be easier to obtain the `IT_MFA` IOR, using the JCL. This is because it is then already in the correct format to place it in the PDS-based configuration file.

Starting the Server Adapter

Overview

This section describes how to start the IMS server adapter. It discusses the following topics:

- [Starting the server adapter on native OS/390](#)
 - [Starting the server adapter on OS/390 UNIX System Services](#)
 - [Adapter logging information](#)
-

Starting the server adapter on native OS/390

In a native OS/390 environment, you can start the IMS server adapter in any of the following ways:

- As a batch job.
- Using a TSO command.
- As a started task (by converting the batch job into a started task).

The default IMS server adapter is the server adapter whose configuration is defined directly in the `iona_services.imsa` scope, and not in some sub-scope of this. The following is sample JCL to run the default IMS server adapter:

```
//IMSA      JOB      ( ),
//          CLASS=A,
//          MSGCLASS=X,
//          MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID,
//          REGION=0M,
//          TIME=1440
//          *
//          JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
//          INCLUDE MEMBER=(ORXVARS)
//          *
//          * Run the Orbix IMS Adapter
//          *
//          * Make the following changes before running this JCL:
//          *
//          * 1. Change 'SET DOMAIN='DEFAULT@' to your configuration
//          *    domain name.
//          *
//          *          SET DOMAIN='DEFAULT@'
//          *
```

```
//GO EXEC PROC=ORXG,
// PROGRAM=ORXIMSA,
// PPARM='run'
//MFAMAPS DD DUMMY
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR
```

Starting the server adapter on OS/390 UNIX System Services

On OS/390 UNIX System Services, you can start the IMS server adapter from the shell. The command to run the default IMS server adapter is similar to the following if you have an `initial_references:IT_MFA:reference` entry in the root scope (that is, not inside any `{}` brackets) of your configuration file:

```
$ itimsa
```

The command to run extra server adapters is similar to the following:

```
$ itimsa -ORBname iona_services.imsa.gateway2
```

Refer to [“Running Multiple Server Adapters Simultaneously” on page 220](#) for more details on running multiple server adapters.

Adapter logging information

When the adapter is started, if a sufficient logging level is enabled, some basic information is displayed on how the particular adapter is configured, including which region it is going to connect with. If client principal support is not enabled, the logged information includes the user ID under which the server adapter is running. This is normally the TSO/E user ID running the adapter. However, if a `USERIDALLIASTABLE` is in use in OS/390 UNIX System Services, the user ID that is displayed instead is the alias associated with the user ID. Regardless of which user ID (that is, TSO/E or alias) is displayed, for OS/390 it is the same user ID, so it does not affect the functionality of the server adapter.

Stopping the IMS Server Adapter

Overview

This section describes how to stop the server adapter. It discusses the following topics:

- [Stopping the adapter via the admin interface](#)
- [Stopping the adapter on native OS/390](#)
- [Stopping the adapter on OS/390 UNIX System Services](#)

Stopping the adapter via the admin interface

The IONA administrative interface is used to configure and manage Orbix installations. This interface can be invoked via the `ORXADMIN` JCL on OS/390 or the `itadmin` shell command on OS/390 UNIX System Services. As with the other IONA services, you can stop the IMS server adapter by issuing an `admin stop` command that uses the appropriate admin plug-in (in this case, the `mfa` plug-in). For example, the format of the command is as follows on OS/390 UNIX System Services:

```
% itadmin mfa stop
```

This instructs the adapter to shut down.

Stopping the adapter on native OS/390

To stop an IMS server adapter job on native OS/390, issue the `STOP (P)` operator command from the console.

Stopping the adapter on OS/390 UNIX System Services

To stop an IMS server adapter process on OS/390 UNIX System Services, use the `kill` command or, if the adapter is running in an active rlogin shell, press **Ctrl-C**.

Running Multiple Server Adapters Simultaneously

Overview

This section describes how to run multiple server adapters simultaneously. It discusses the following topics:

- [Running multiple server adapters simultaneously](#)
- [Using itadmin on OS/390 UNIX System Services](#)

Running multiple server adapters simultaneously

To run multiple IMS server adapters perform the following steps.

Step	Action
1	Set up a configuration scope for each server adapter (for example, the <code>gateway2</code> scope) in the partial configuration file. (Refer to the example in “An IMS Server Adapter Sample Configuration” on page 44.)
2	Set up a corresponding configuration scope for usage with the <code>admin</code> utility. For example, add a <code>gateway2</code> sub-scope to the <code>iona_utilities.imsa</code> scope in the configuration file, and add the following configuration setting under it: <code>initial_references:IT_MFA:reference=%{LOCAL_MFA_IMS_REFERENCE2}</code>
3	Specify a unique <code>imsa:poa_prefix</code> variable for each server adapter if you are using the locator (indirect persistent). This is a good idea anyway, even for direct persistent server adapters, because the IORs are easier to distinguish when examined with the <code>iorump</code> utility.
4	Set the unique port number.

Step	Action
5	<p>Get the initial reference for each adapter.</p> <p>On native OS/390, change the IMS server adapter prepare JCL to use the new <code>ORBname</code>, and replace the <code>LOCAL</code> variable with the new <code>LOCAL_MFA_IMS_REFERENCE2</code> variable.</p> <p>On OS/390 UNIX System Services, enter the following command to obtain the IOR:</p> <pre>\$ itimsa -ORBname iona_services.imsa.gateway2 prepare</pre> <p>Enter the following command on OS/390 UNIX System Services, to add the new reference to the configuration file:</p> <pre>\$ itadmin variable create -value IOR:00000..0 LOCAL_MFA_IMS_REFERENCE2.</pre>
6	<p>Ensure that each server adapter has:</p> <ul style="list-style-type: none"> • A unique mapping file. • A unique IFR signature cache file, if one is being used. • A unique type-info store, if one is being used. • A unique XCF member name, if OTMA is being used. • A unique resource manager name, if RRS is being used.

Using itadmin on OS/390 UNIX System Services

It might be useful to run in shell mode, so that you do not have to type the long `ORBname` in the JCL's `itadmin` parameter. To run `itadmin` on OS/390 UNIX System Services:

```
$ itadmin -ORBname iona_utilities.imsa.gateway2
% mfa list
% mfa resolve .....
```

Note: When using JCL to issue `itadmin` commands on native OS/390, include the full `ORBname` in the JCL's `itadmin` parameter.

Using the MappingGateway Interface

Overview

The `MappingGateway` interface is used to control a running IMS server adapter. It discusses the following topics:

- [Uses of the MappingGateway interface](#)
- [Access to the MappingGateway interface](#)
- [Selecting a specific server adapter](#)

Uses of the MappingGateway interface

You can use `MappingGateway` interface to list the transaction mappings that the server supports, to add or delete individual interfaces and operations, or to alter the transaction to which an operation is mapped. You can use it to read a new mapping file, or write an existing mapping to a new file.

Additionally, the `MappingGateway` interface provides the means by which IIOB clients can invoke on the exported interfaces. Using the `resolve` operation, an IOR can be retrieved for any exported interface. This IOR can then be used directly by IIOB clients, or registered with an OrbixNames server as a way of *publishing* the availability of the interface.

Access to the MappingGateway interface

The `MappingGateway` interface is provided both via the `itadmin` interface and as an IDL interface. The IDL for the `MappingGateway` interface is provided with the other IDL in the installation and can be used by client applications to invoke operations on the `MappingGateway` interface.

Access to the `MappingGateway` interface, using `itadmin`, is provided as a plug-in. This plug-in is selected with the `mfa` keyword. This `itadmin mfa` plug-in is an IONA-supplied client of the `MappingGateway` interface, and is provided to make it easier to access the `MappingGateway` interface. For example, to obtain a list of all the operations provided by the `mfa itadmin` plug-in, issue the following command (from the UNIX System Services shell or via JCL on native OS/390):

```
$ itadmin mfa -help
```

The output looks as follows:

```
mfa list
  add      -interface <name> -operation <name> <mapped value>
  change   -interface <name> -operation <name> <mapped value>
  delete   -interface <name> -operation <name>
  resolve   <interface name>
  refresh  [-operation <name>] <interface name>
  reload
  save     [<mapping_file name>]
  switch   <mapping_file name>
  stats
  resetcon
  stop
```

Items shown in angle brackets (<...>) must be supplied and items shown in square brackets ([...]) are optional. Module names form part of the interface name and are separated from the interface name with a / character.

The parameter after `mfa` specifies the operation to be invoked. The options are:

- `list` This prints a list of the (interface, operation, and name) mappings that the IMS server adapter currently supports.
- `add` This allows you to add a new mapping.
- `change` This allows you to change the transaction to which an existing operation is mapped.
- `delete` This allows you to get the IMS server adapter to stop exporting a particular operation.
- `resolve` This prints a stringified IOR for the object in the server adapter that supports the specified interface. This IOR string can then be given to clients of that interface, or stored in an OrbixNames server. The IOR produced contains the TCP/IP port number for the locator if the IMS server adapter is running with direct persistence set to `no`; otherwise, it contains the IMS server adapter's port number.
- `refresh` This causes the IMS server adapter to obtain up-to-date type information for the specified operation. If you omit the operation argument, all operations being mapped in the specified interface are refreshed.
- `reload` This causes the IMS server adapter to reload the list of mappings from its mapping file.

<code>save</code>	This causes the IMS server adapter to save its current mappings to either its current mapping file or to a filename you provide.
<code>switch</code>	This causes the IMS server adapter to switch over to a new mapping file, and to export only the mappings contained within it.
<code>stats</code>	Displays some statistical information on the running server adapter. Information includes the current time according to the server adapter, the pending request queue length, the total number of worker threads, worker threads currently active, total number of requests processed by the server adapter since start-up, and the server adapter start-up time.
<code>resetcon</code>	If the server adapter is using OTMA to communicate with IMS, when this operation on the <code>MappingGateway</code> interface is called, the server adapter closes its connection with OTMA and reconnect. This is done in such a way that it does not affect any clients connected to the server adapter, by briefly queueing client requests in the server adapter until the connection is re-established. The purpose of this operation is to free any cached security ACEE's on the OTMA connection. This operation should therefore be used after changes in the security profiles of the users in the OS/390 security package, for example RACF, have been made that would affect their access to IMS.
<code>stop</code>	Instructs the IMS server adapter to shut down.

Note: The `add`, `change`, and `delete` operations only update the IMS server adapter internal information, unless a `save` operation is issued, in which case the new details are written to the server adapter mapping file.

Selecting a specific server adapter

To select a specific server adapter, provide the `ORBname` for the server adapter on a request. For example, to obtain the IOR for the `SimpleObject` interface, use the following command:

```
itadmin -ORBname iona_utilities.imsa mfa resolve
Simple/SimpleObject
```

Locating IMS Server Adapter Objects Using itmfaloc

Overview

The IMS server adapter maintains object references that identify CORBA server programs running in IMS. A client must obtain an appropriate object reference to access the target server. The `itmfaloc` URL resolver plug-in supplied with your Orbix Mainframe installation facilitates and simplifies this task.¹

This section discusses the following topics:

- [Locating IMS servers using IORs](#)
- [Locating objects using itmfaloc](#)
- [Format of an itmfaloc URL](#)
- [What happens when itmfaloc is used](#)
- [Example of using itmfaloc](#)

Locating IMS servers using IORs

One way of obtaining an object reference for a target server, managed by the IMS server adapter, is to retrieve the IOR via the `itadmin` utility. This calls the `resolve` method on the server adapter's `MappingGateway` interface and returns a stringified IOR. For example, to retrieve an IOR for the `SimpleObject` IDL interface, issue the following command:

```
itadmin mfa resolve Simple/SimpleObject
```

After it has been retrieved, the IOR can be distributed to the client and used to invoke on the target server running inside IMS.

Locating objects using itmfaloc

In some cases, the use of `itadmin` and the need to persist stringified IORs is not very manageable, and thus a more dynamic approach is desirable. The `itmfaloc` resolver is designed to provide an alternative approach. It follows a similar scheme to that of the `corbaloc` URL technique. (Refer to the *CORBA Programmer's Guide, C++* for more information).

1. This plug-in is not yet available on other Orbix platforms.

In this way, the Orbix CORBA client can specify a very simple URL format which identifies the target service required. This text string can therefore be used programmatically in place of the rather cumbersome stringified IOR representation.

Format of an itmfaloc URL

An itmfaloc URL is a string of the format:

```
itmfaloc:<InterfaceName>
```

In the preceding example, *<InterfaceName>* represents the fully scoped name of the IDL interface implemented by the target IMS server, as specified in the server adapter mapping file.

What happens when itmfaloc is used

When an itmfaloc URL is used in place of an IOR, the Orbix client application contacts the server adapter to attain an object reference for the desired IMS server. The itmfaloc URL string only encodes the interface name, not the server adapter's location. To establish the initial connection to the server adapter, the `IT_MFA:initial_references` configuration item is used.

If multiple server adapters are deployed, it is imperative that the client application specifies the correct `IT_MFA:initial_references` setting, to talk to the correct IMS server adapter. This can be achieved by specifying the appropriate ORBname which represents the particular configuration scope; for example, `-ORBname iona_utilities.imsa`.

If the client application successfully connects to the server adapter, it then calls the `resolve` operation on the `MappingGateway` object reference, thus retrieving an object reference for the target server managed by the IMS server adapter.

Example of using itmfaloc

The simple demonstration client code that is shipped with Orbix uses a file-based mechanism to access the target server's stringified IOR. If the target server resides in IMS, an alternative approach is to specify an itmfaloc URL string in the string-to-object call. For example:

```
itmfaloc:Simple/SimpleObject
```

The relevant Orbix APIs are:

- `str2obj` (PL/I)
- `STRTOOBJ` (COBOL)
- `string_to_object()` (C++)

WFI Support for IMS Transactions

Overview

The IMS server adapter runtime inside IMS provides implicit support to run mapped transactions in Wait-for-Input (WFI) or Pseudo Wait-for-Input (PWFI) mode.

This section discusses the following topics:

- [Running the IMS server adapter in WFI mode](#)
- [Defining a WFI transaction](#)

Running the IMS server adapter in WFI mode

In WFI mode, the server mainline code is executed only once and the transaction then waits in IMS for additional requests. (It therefore works similar to a batch server.) Only the implementation code then runs for each transaction. Refer to the *COBOL Programmer's Guide and Reference* and the *PL/I Programmer's Guide and Reference* for details on the difference between the server mainline code and the server implementation code. The IMS server adapter continues to wait for input requests until one of the normal IMS events (for example, a timeout) takes place and the server transaction then stops running.

Defining a WFI transaction

A transaction can be defined as WFI as follows:

```
TRANSACTION  
CODE=SIMPLESV,  
EDIT=(ULC),WFI,  
PROCLIM=(60,60)
```

No changes to the IMS Orbix server source code are needed. The Orbix DLLs inside IMS handle the processing of multiple transactions in a single scheduling session.

Conversational Support

Overview

The IMS server adapter provides a facility to run conversational transactions, using the OTMA or APPC plug-in. The `imsraw` interface has four operations that makes this possible. This section describes how to use `imsraw` to access conversational transactions. It discusses the following topics:

- [Steps to run a conversational transaction](#)
 - [imsraw IDL example](#)
 - [imsraw IDL explanation](#)
 - [Client code examples](#)
-

Steps to run a conversational transaction

Running a conversational transaction in IMS consists of three steps:

1. Start the conversational transaction.
 2. Issue a set of requests on the conversational transaction, receiving a reply for each request. This set of requests and replies should follow the logical flow of the transaction to its conclusion.
 3. End the conversational session, thereby freeing up the resources it used in the server adapter, OTMA or APPC (depending on the plug-in used), and IMS.
-

imsraw IDL example

The IDL in the `imsraw` interface that handles conversational transactions is as follows:

Example 12: `imsraw` IDL Interface

```

1 module IT_MFA_IMS
  {
    interface imsraw {
      ...
      ...
      // Run conversational imsraw transactions
      //
      typedef sequence<octet> SessionHandle;

      // Start the conversation in IMS
      void start_session(in tranName tran_name,
```

Example 12: *imsraw IDL Interface*

```

        out SessionHandle session)
        raises(internalError);
    //
    // Methods for invoking conversational IMS transactions.
    // The first uses CharSegments, so data is subject
    // to ASCII-EBCDIC conversion cross-platforms, the
    // second uses ByteSegments so no conversion will be
    // done.
    //
2 CharSegments run_conv_transaction(in SessionHandle session,
    in CharSegments din)
    raises(segmentTooLarge,
        IMSunavailable,
        unknownTransactionName,
        userNotAuthorized,
        transactionFailed,
        internalError);

3 ByteSegments run_conv_transaction_binary(
    in SessionHandle session,
    in ByteSegments din)
    raises(segmentTooLarge,
        IMSunavailable,
        unknownTransactionName,
        userNotAuthorized,
        transactionFailed,
        internalError);

    // End the conversation in IMS
3 void end_session(in SessionHandle session)
    raises(internalError);
};
...
...
};

```

imsraw IDL explanation

[Example 12](#) can be explained as follows:

1. The first operation in the `imsraw` interface for conversational transactions is `start_session()`. This operation takes one input parameter (that is, the IMS transaction name) and returns a session handle for the transaction. The returned session handle must be used on all subsequent operations for this transaction.
2. The next two operations—`run_conv_transaction()` and `run_conv_transaction_binary()`—are used to interact with the transaction during the conversation. They work in the same way as the non-conversational operations. The only difference is that they take a session handle instead of a transaction name. This session handle is the one created via the `start_session()` call. Also, these operations can be called as often as is necessary to complete a conversation.
3. The `end_session()` operation is used to free the resources of a conversation after the conversation has been completed. After this call has been made, the session handle is no longer valid and should not be used. Again, it uses the session handle created via the `start_session()` call.

Client code examples

The C++ client code to create a session is as follows:

```
IT_MFA_IMS::imsraw_var IMSBridge = ...;
const char* tran_name = "XXXXXXXX";
IT_MFA_IMS::imsraw::SessionHandle_var session;
// Start the session
IMSBridge->start_session(tran_name, session);
```

The C++ client code to interact with another "screen" in the conversation is as follows:

```
IT_MFA_IMS::imsraw::CharSegments in;
IT_MFA_IMS::imsraw::CharSegments_var result;

// Fill in the input segments here
...
// Call the conversation transaction
result = IMSBridge->run_conv_transaction(session, in);
```

The C++ client code to end the session when the conversation is finished is as follows:

```
// Free the conversation session  
IMSBridge->end_session(session);
```

See the `imsraw` sample application supplied with your Orbix Mainframe installation for an example of how to obtain the `IMSBridge` object reference and handle the input and subsequent results from each transaction call.

LTERM Propagation

Overview

The OTMA-based server adapter can propagate into IMS a logical terminal (LTERM) field that originates from the client application. It can also subsequently return the LTERM value obtained from IMS back to the client.

Passing the LTERM field

The client application uses a request service context to send the LTERM value to the OTMA-based server adapter. The IONA service context ID allocated for this purpose is `0x49545F46`. The client application can use a portable interceptor to set the LTERM service context with the appropriate LTERM value. See the *CORBA Programmer's Guide, C++* for details on writing a portable interceptor.

The OTMA-based server adapter looks for the LTERM service context in requests that it receives from the client. If the LTERM service context is present, the LTERM value in the service context is used in the `otma_send_receive()` call that sends the client request into IMS.

When the `otma_send_receive()` call completes in the OTMA-based server adapter, the LTERM value (which might have been updated inside IMS) is returned to the client. The return value is sent in a reply service context that uses the same LTERM service context ID as that used for the request service context (that is, `0x49545F46`).

The client application's portable interceptor is then responsible for retrieving the LTERM value that is returned in the LTERM service context. An IDL constant declaration for the LTERM ID can be located in the `imsraw` IDL definition file in your Orbix Mainframe installation. The LTERM value itself is encoded as a CORBA string in the service context. The adapter truncates the received value to eight characters, if necessary, before starting the IMS transaction.

Note: No additional Orbix configuration is required to enable this feature. See the portable interceptor ORB service demonstration supplied with your Orbix Mainframe installation at `install-dir/asp/6.0/demos/corba/orb/portable_interceptor/orb_service` for an example of how a client can pass a string within a service context.

Adding a Portable Interceptor to the IMS Server Adapter

Overview

This section describes how to add a portable interceptor (or multiple interceptors) to the server adapter. This can be used to perform the usual functions available in portable interceptors. Refer to the *CORBA Programmer's Reference, C++* and *CORBA Programmer's Guide, C++* for more details on portable interceptors. Additionally, a portable interceptor can be used to manipulate the client principal that the IMS server adapter receives from the client.

In this section

This section discusses the following topics:

Developing the Portable Interceptor	page 235
Compiling the Portable Interceptor	page 239
Loading the Portable Interceptor into the IMS Server Adapter	page 241

Developing the Portable Interceptor

Overview

A portable interceptor should be developed as described in the *CORBA Programmer's Guide, C++*. For the server adapter, only server-side interceptors are of interest, because the IMS server adapter is a CORBA server.

This subsection discusses the following topics:

- [Server adapter portable interceptor sample locations](#)
 - [Contents of the ORB plug-in implementation](#)
 - [Contents of the ORB initializer implementation](#)
 - [Contents of the server interceptor implementation](#)
 - [Server interceptor sample code](#)
 - [Server interceptor sample code explanation](#)
-

Server adapter portable interceptor sample locations

An example of a portable interceptor framework for use in the server adapter is provided in `orbixhlq.DEMOS.CPP.SRC` and `orbixhlq.DEMOS.CPP.H`. The header file members are `ORBINITI` and `SRVINTRC`. The source file members are `PLUGIN`, `ORBINITI`, and `SRVINTRC`.

For an OS/390 UNIX System Services installation, the demonstration is located in `$IT_PRODUCT_DIR/asp/6.0/demos/corba/pdk/security_pi`. The header files are located in `orb_initializer_impl.h` and `server_interceptor_impl.h`. The implementation files are located in `plugin.cxx`, `orb_initializer_impl.cxx` and `server_interceptor_impl.cxx`.

The portable interceptor is packaged as a standard ORB plug-in, to enable it to be loaded by an existing Orbix server (in this case, the IMS server adapter).

Contents of the ORB plug-in implementation

The ORB plug-in implementation contains code to register this DLL as an ORB plug-in. The ORB plug-in implementation also contains code in its `ORB_init()` method to register the portable interceptor's ORB initializer object with the ORB. The ORB plug-in mechanism is used here to enable the server adapter to load this DLL when the adapter is started. (See

[“Loading the Portable Interceptor into the IMS Server Adapter” on page 241.](#)) Sample source is provided in the `PLUGIN` member on OS/390 and in the `plugin.cxx` file on OS/390 UNIX System Services.

Contents of the ORB initializer implementation

The ORB initializer implementation contains code to register the server request interceptor with the ORB. Refer to the *CORBA Programmer’s Guide*, C++ for details on how to implement an ORB initializer. The initializer is registered in the `IT_Security_Plugin` class (that is, the ORB plug-in implementation). Sample source is provided in the `ORBINITI` members on OS/390, and in the `orb_initializer.h` and `orb_initializer.cxx` files on OS/390 UNIX System Services.

Contents of the server interceptor implementation

The server request interceptor implementation illustrates how you can intercept the incoming CORBA request, check the GIOP principal value, and potentially modify this principal value before it is subsequently used by the server adapter. (See [“Activating Client Principal Support” on page 97](#) for more details.) To achieve this functionality, the interceptor only implements the `receive_request()` interception point. This is the point at which the principal has been read in from the GIOP request message and is available via the GIOP `current` API. Sample source is available in the `SRVINTRC` dataset members on OS/390, and in the `server_interceptor_impl.h` and `server_interceptor_impl.cxx` files on OS/390 UNIX System Services.

Server interceptor sample code

The `receive_request()` interceptor looks as follows:

Example 13: Sample Server Interceptor code (Sheet 1 of 2)

```
void
Demo_ServerInterceptorImpl::receive_request(
    PortableInterceptor::ServerRequestInfo_ptr ri
) IT_THROW_DECL((
    CORBA::SystemException,
    PortableInterceptor::ForwardRequest
))
{
11  CORBA::OctetSeq_var received_val_binary =
22  m_current->received_principal();
    if (received_val_binary->length() != 0)
    {
```

Example 13: *Sample Server Interceptor code (Sheet 2 of 2)*

```

33         if (received_val_binary[received_val_binary->length()-1]
            == '\0')
            {
                cout << "Received a string principal in PI" << endl;
            }
            else
            {
                cout << "Received a binary principal in PI" << endl;
                return;
            }
        }
        else
        {
            cout << "Did not received any principal!" << endl;
            return;
        }
44 // Show the principal value
CORBA::String_var received_val =
    m_current->received_principal_as_string();

    if (strlen(received_val.in()) != 0)
    {
        cout << "Received principal string in PI "
            << received_val.in() << endl;
55 // This is very contrived, but shows how to change a principal
        cout << "If principal is JOHN, change to PETER" << endl;
        if (strcmp(received_val.in(), "JOHN") == 0)
        {
            char* new_user = "PETER";
66         m_current->change_received_principal_as_string(new_user);
        }
    }
    else
    {
        cout << "Did not received any principal!" << endl;
    }
}

```

Server interceptor sample code explanation

The sample server interceptor code can be explained as follows:

- 1 Obtain the principal in binary format. In binary format, the principal value does not undergo ASCII-to-EBCDIC conversion.
- 2 Check if a principal has been received.
- 3 Check if the principal value ends in a null terminator, which indicates that it is probably a string. (This depends on the conventions agreed with the client application.)
- 4 Because the interceptor returns if the principal value is not a string, it now re-obtains the principal value as a string with ASCII-to-EBCDIC conversion taking place.
- 5 In this example, it checks if the principal is `JOHN`.
If the principal is `JOHN`, it is changed to `PETER`. This is just an example to show how to change a principal. Production applications probably have more complex rules for modifying principals.
- 6 Other interceptor points can also be implemented. For example, the `send_exception()` interceptor point can be implemented if tracking or logging of exceptions is desired. The `receive_request_service_contexts()` interceptor can be implemented if access to additional service contexts is required.

Compiling the Portable Interceptor

Overview

This subsection outlines the build information used to compile the portable interceptor demonstration. It also provides information about the naming of the compiled DLL, and the location of the readme files that provide additional information about compiling the portable interceptor.

This section discusses the following topics:

- [Compiling on native OS/390](#)
- [Compiling on OS/390 UNIX System Services](#)
- [Specifying the correct DLL name when loading the portable interceptor](#)
- [Location of additional information for compiling the portable interceptor](#)

Compiling on native OS/390

Sample JCL to compile the portable interceptor can be found in `orbixhlq.DEMOS.CPP.BLD.JCL(ADTPICL)`. This compiles the two sample source files and links them into a DLL called `SECP11`.

Compiling on OS/390 UNIX System Services

The `$IT_PRODUCT_DIR/asp/6.0/demos/corba/pdk/security_pi` directory contains a makefile that is used to build the `SECP11` DLL on OS/390 UNIX System Services.

Specifying the correct DLL name when loading the portable interceptor

The DLL name, `SECP11`, has been chosen for this example, because it is a valid name in both a native OS/390 and OS/390 UNIX System Services environment. Any valid DLL name can be used for your target deployment environment. The correct DLL name must then be specified when selecting the portable interceptor that is to be loaded into the server adapter. Refer to [“Loading the Portable Interceptor into the IMS Server Adapter” on page 241](#) for more details.

Location of additional information for compiling the portable interceptor

On native OS/390, the `ADTPI` member in `orbixhlq.DEMOS.CPP.README` also provides a description of how to compile the portable interceptor. You can refer to this for additional information.

On OS/390 UNIX System Services, similar information tailored to compiling the portable interceptor is provided in `$IT_PRODUCT_DIR/asp/6.0/demos/corba/pdk/security_pi/README_CXX.txt`.

Loading the Portable Interceptor into the IMS Server Adapter

Overview

This subsection describes how the portable interceptor is loaded into the IMS server adapter. It discusses the following topics:

- [Loading the portable interceptor on native OS/390](#)
 - [Loading the portable interceptor on OS/390 UNIX System Services](#)
 - [Setting related configuration items](#)
 - [Sample IMS server adapter configuration scope](#)
-

Loading the portable interceptor on native OS/390

Add the PDS containing the portable interceptor DLL to the STEPLIB for the IMS server adapter. On native OS/390, this can be done by updating the JCL used to run the server adapter. For example, add a `LOADLIB` value as follows:

```
//GO EXEC PROC=ORXG,
// PROGRAM=ORXIMSA,
// LOADLIB=&ORBIX..DEMOS.CPP.LOAD,
// PPARM='run'
```

Note: If the `LOADLIB` symbolic is already in use, you might wish to update the `ORXG` procedure and add the PDS that contains the portable interceptor into the `STEPLIB` concatenation.

Loading the portable interceptor on OS/390 UNIX System Services

If the server adapter is run from OS/390 UNIX System Services, and the portable interceptor was built using JCL on native OS/390, so the `SECP11` DLL resides in a PDS, add the PDS to the `STEPLIB` environment variable. The following is an example of how to do this, where `IT_PRODUCT_HLQ` is set to the relevant Orbix HLQ install area:

```
export STEPLIB=$IT_PRODUCT_HLQ.DEMOS.CPP.LOAD:$STEPLIB
```

If the server adapter is run from OS/390 UNIX System Services, and the portable interceptor was built in OS/390 UNIX System Services using a makefile (so the `SECP11` DLL resides in a UNIX System Services directory), add the directory that contains the `SECP11` DLL to the `LIBPATH` environment

variable. The following is an example of how to do this, where *IT_PRODUCT_DIR* is set to the relevant Orbix install area for OS/390 UNIX System Services:

```
export LIBPATH=$IT_PRODUCT_DIR/asp/6.0/demos/corba/pdk/
security_pi:$LIBPATH
```

Setting related configuration items

The following configuration items must be set to load the plug-in:

`orb_plugins`

The list must include the `demo_sec` ORB plug-in, which is the name that was used in the ORB plug-in demonstration code. This plug-in must appear before the `portable_interceptor` plug-in in the `orb_plugins` list.

The list must also include the `portable_interceptor` plug-in, to allow for portable interceptor support to be activated.

`binding:server_binding_list`

The name of the server request interceptor must be added to this list, to allow it to gain control when a server request is being processed. For the purposes of this example, add the `DemoPI` interceptor.

`plugins:demo_sec:shlib_name`

Specifies the name of the ORB plug-in library, without the version suffix.

plugins:demo_sec:shlib_version

Specifies the version number of the ORB plug-in library.

Note: On OS/390, unlike on other platforms, a particular ORB plug-in DLL name is resolved from the Orbix configuration simply by appending the `shlib_version` to the `shlib_name`.

Sample IMS server adapter configuration scope

For example, the following can be added to the IMS server adapter's configuration scope:

```
orb_plugins = ["iiop_profile", "giop", "iiop",
              "local_log_stream", "ots", "demo_sec",
              "portable_interceptor"];
binding:server_binding_list = ["DemoPI"];
plugins:demo_sec:shlib_name = "SECPI";
plugins:demo_sec:shlib_version = "1";
```

When the IMS server adapter is then started, the portable interceptor should be loaded and included in the server-side communication bindings.

Gathering Accounting Information in the Server Adapter

Overview

This section describes how to add a DLL to the IMS server adapter which can gather and log accounting type information. This is done in the form of a user-replaceable DLL. A sample DLL is provided, which can be modified to collate results and write them to the desired location; for example, some kind of log file.

In this section

This section discusses the following topics:

Developing the Accounting DLL	page 245
Compiling the Accounting DLL	page 248
Loading the Accounting DLL into the Server Adapter	page 249

Developing the Accounting DLL

Overview

The accounting DLL consists of a call to the function `IT_MFA_display_account_information()` for mapped requests, and a call to the function `IT_MFA_display_raw_interface_account_information()` for `imsraw` requests, after each IMS server adapter request has been completed. You can implement your own version of these functions and replace the DLL called `ORXACCT1`, to gather the customized accounting information.

This subsection discusses the following topics:

- [IT_MFA_display_account_information\(\) parameters](#)
 - [Sample use of IT_MFA_display_account_information\(\)](#)
 - [Location of sample source code](#)
-

IT_MFA_display_account_information() parameters

The parameters for the function contain the following information:

<code>interface</code>	This is the interface name of the request.
<code>operation</code>	This is the operation name of the request.
<code>mapped_name</code>	This is the transaction or program name that is invoked in IMS.
<code>request_length</code>	This is the total length of inbound data received from TCP/IP, excluding the 12-byte fixed GIOP header.
<code>reply_length</code>	This is the total length of outbound data sent back via TCP/IP, excluding the 12-byte fixed GIOP header.
<code>principal</code>	The Client principal, if available; otherwise, an empty string.
<code>local_arglist</code>	This is an NVList of all the arguments for the request. This NVList is in the state after the reply has been transmitted back to the client application, so only limited data is available in it.
<code>dynany_set</code>	Indicates if the first argument has been saved in a dynamic <code>any</code> when the request was received from the client. This dynamic <code>any</code> is the next parameter. Saving the argument has to be activated via configuration.

da	First argument, if saved. Refer to the chapter on Any's and Dynamic Any's in the <i>CORBA Programmer's Guide</i> , C++ for details on how to access the data contained in this parameter.
orb	Pointer to the server adapter ORB, if needed, for example, to call <code>resolve_initial_references()</code> to obtain a current object.

Sample use of `IT_MFA_display_account_information()`

Here is an example of what can be done in the function:

Example 14: Sample use of `IT_MFA_display_account_information()` (Sheet 1 of 2)

```
#include <it_cal/iostream.h>
#include <it_cal/fstream.h>
#include <string.h>
#include <it_mfa/account.h>

IT_USING_NAMESPACE_STD
void
IT_MFA_display_account_information(
    const char* interface,
    const char* operation,
    const char* mapped_name,
    CORBA::Long request_length,
    CORBA::Long reply_length,
    const char* principal,
    CORBA::NVList_ptr local_arglist,
    CORBA::Boolean dynany_set,
    DynamicAny::DynAny_ptr da,
    CORBA::ORB_ptr orb
)
{
    cout << "Accounting information: " << endl;
    cout << " Interface:   " << interface << endl;
    cout << " Operation:   " << operation << endl;
    cout << " Tran:       " << mapped_name << endl;
    cout << " Request len: " << request_length << endl;
    cout << " Reply len:  " << reply_length << endl;
    cout << " Principal:  " << principal << endl;
}
```

Example 14: *Sample use of `IT_MFA_display_account_information()` (Sheet 2 of 2)*

```

// Gather type information from the NVList
cout << " Number of Arguments: " << local_arglist->count() <<
endl;

// Display information from the first parameter
if (dynany_set == IT_TRUE)
{
    CORBA::TypeCode_ptr type = da->type();

    cout << " Kind: " << type->kind() << endl;
    cout << " Id:   " << type->id() << endl;
    if ((type->kind() == CORBA::tk_struct))
    {
        cout << " Member count: " << type->member_count() <<
endl;
        for (int ii=0; ii < type->member_count(); ii++)
        {
            CORBA::TypeCode_ptr type1 = type->member_type(ii);
            cout << " Kind of member: " << type1->kind() <<
endl;
        }
    }
    cout << endl;
}
}

```

Location of sample source code

The source code for this sample function is contained in `orbixhlq.DEMOS.CPP.SRC(ACCOUNT)`. This example can be used as a basis for a function which logs the request accounting information in the desired format.

Compiling the Accounting DLL

Overview

The functions `IT_MFA_display_account_information()` and `IT_MFA_display_raw_interface_account_information()` must be compiled into a C++ DLL, called `ORXACCT1`. This is the DLL name which the IMS server adapter attempts to load if it is configured to produce accounting information.

This subsection discusses the following topics:

- [Location of sample JCL to compile `IT_MFA_display_account_information\(\)`](#)
- [Location of additional information for compiling `IT_MFA_display_account_information\(\)`](#)

Location of sample JCL to compile `IT_MFA_display_account_information()`

Sample JCL to compile the DLL can be found in `orbixhlq.DEMOS.CPP.BUILD.JCL(ACCTCL)`.

Location of additional information for compiling `IT_MFA_display_account_information()`

The `orbixhlq.DEMOS.CPP.README(ACCOUNT)` file also provides a description of how to compile the DLL, which can be referred to for additional information.

Loading the Accounting DLL into the Server Adapter

Overview

This subsection describes how the accounting DLL is loaded into the server adapter. It discusses the following topics:

- [Loading the accounting DLL on native OS/390](#)
 - [Loading the accounting DLL on OS/390 UNIX System Services](#)
 - [Setting required configurative variables](#)
-

Loading the accounting DLL on native OS/390

To load the accounting DLL on native OS/390, add the PDS containing the accounting DLL to the `STEPLIB` for the server adapter. This can be done by updating the server adapter JCL. For example, add a `LOADLIB` value as follows:

```
//GO EXEC PROC=ORXG,
// PROGRAM=ORXIMSA,
// LOADLIB=&ORBIX..DEMOS.CPP.LOAD,
// PPARM= 'run'
```

Loading the accounting DLL on OS/390 UNIX System Services

To load the accounting DLL on OS/390 UNIX System Services, add the PDS to the `STEPLIB` environment variable, for example using:

```
export STEPLIB=orbixhlq.DEMOS.CPP.LOAD:$STEPLIB
```

In the preceding example, *orbixhlq* represents the relevant high-level qualifier for the PDS.

Setting required configurative variables

If the `plugins:imsa:call_accounting_dll` configuration item is set to "yes", the DLL `ORXACCT1` is loaded by the server adapter and the function is called after each request.

If the `plugins:imsa:capture_first_argument_in_dynany` configuration item is set to "yes", the first argument of the request, if it is an input argument, is also preserved and passed to the function.

Exporting Object References at Runtime

Overview

When you start the server adapter it can export object references for the interfaces it supports. These object references relate to the `MappingGateway` interface, the `imsraw` interface, and (optionally) any other mapped interfaces that have been defined to the server adapter via its mapping file at start-up. The server adapter can export these object references to a file, to the Naming Service, or both.

In this section

This section discusses the following topics:

Configuration Items for Exporting Object References	page 251
Exporting Object References to a File	page 253
Exporting Object References to Naming Service Context	page 254
Exporting Object References to Naming Service Object Group	page 256

Configuration Items for Exporting Object References

Overview

This subsection describes the configuration items that are used to control the export of object references from the server adapter.

Configuration Items Summary

The following table summarizes the configuration items that are used to control the export of object references from the server adapter:

Note: None of these configuration items are included by default in the adapter configuration file. If you want to configure the server adapter to export object references, you must add these configuration items, as appropriate.

<code>plugins:imsa:write_iors_to_file</code>	If you want to export object references to a file, this specifies the file that should be used. See “Configuration Example” on page 253 for more details.
<code>plugins:imsa:write_iors_to_ns_context</code>	If you want to export object references to a Naming Service context, this specifies the Naming Service context that should be used. If you specify a value of "", the object references are written to the root context of the Naming Service.
<code>plugins:imsa:publish_all_iors</code>	If this is set to "yes", the object references for the <code>MappingGateway</code> interface, the <code>imsraw</code> interface, and all interfaces specified in the adapter mapping file are exported. If this is set to "no", only the object references for the <code>MappingGateway</code> and <code>imsraw</code> interfaces are exported. The default is "no".

```
plugins:imsa:remove_ns_iors  
  _on_shutdown
```

If this is set to "yes", the server adapter attempts to unbind the object references from the Naming Service when it shuts down normally (for example, via an operator `stop` command). The default is "no".

```
plugins:imsa:write_iors_to_ns  
  _group_with_prefix
```

This specifies the prefix that should be added to each generated name indicating an interface. This prefix is attached to the generated name, to specify the object group that is to be used. If a prefix of "" is specified, no prefix is added. If this configuration setting is not present, no object references are exported to any object groups.

```
plugins:imsa:write_iors_to_ns  
  _group_member_name
```

This specifies the member name that the server adapter should use in the object group. A unique member name must be specified for each adapter; otherwise, one adapter might end up replacing the object group members of another adapter.

Exporting Object References to a File

Overview

When it comes to the server adapter exporting object references, the simplest option is to have the adapter export them to a file. This subsection provides an example of the configuration settings that are required to enable the export of object references to a file, and the subsequent output produced.

Configuration Example

The following configuration settings indicate that the server adapter should export object references for all the interfaces it supports to the home directory of user1:

```
plugins:imsa:write_iors_to_file = "/home/user1/test.txt";
plugins:imsa:publish_all_iors = "yes";
```

Alternatively, the following configuration settings indicate that the server adapter should export object references for only the `MappingGateway` and `imsraw` interfaces to a data set called `MFAIORS`:

```
plugins:imsa:write_iors_to_file = "DD:MFAIORS";
plugins:imsa:publish_all_iors = "no";
```

Example Output

The following is an example of the output produced in the file for the first of the preceding configuration examples, assuming the `simple` demonstration has been added to the adapter mapping file:

```
IT_MFA = IOR:0000000000000027494...
Simple:SimpleObject = IOR:00000000000001c4944...
IT_MFA_IMS:imsraw = IOR:0000000000000254944...
```

Exporting Object References to Naming Service Context

Overview

When it comes to exporting object references to the Naming Service, the server adapter can be configured to export to either a Naming Service context or a Naming Service object group. This subsection provides details about exporting to a Naming Service context.

Prerequisites

If the server adapter is configured to export its object references to a Naming Service context, the following prerequisites apply:

- The Naming Service used must support the `CosNaming::NamingContextExt` interface.
- The initial reference for this Naming Service must be supplied to the adapter either in the adapter's configuration file or via the command line at start-up.

Configuration

The `plugins:imsa:write_iors_to_ns_context` configuration item specifies the Naming Service context to which the adapter should export its object references. If a value of "" (that is, an empty context) is specified for this item, the object references are written to the root context. To indicate a nested context, the specified value must take a format of `"context/context/context"`.

Note: The context must exist when the adapter is started. See the *Orbix Administrator's Guide* for details of how to create contexts with `itadmin`, in particular how to create and specify nested Naming Service contexts.

If `plugins:imsa:remove_ns_iors_on_shutdown` is also specified, the adapter calls `unbind()` on the object references in the Naming Service as part of a normal shut-down operation.

Object ID

The ID for the object bound into the Naming Service is derived from the module and interface name. First, all the module names are used and then the interface name, each separated by a colon. For example, the ID for the interface for the `simple` demonstration is `Simple:SimpleObject`. The `kind` parameter is always left empty. The `MappingGateway` interface uses `IT_MFA` as the ID.

rebind() function

The adapter uses `rebind()` to add the object references to the Naming Service, so any existing object reference with the same name in the same context is replaced.

Example

The following `itadmin` command creates a context called "test" in the Naming Service:

```
itadmin ns newnc test
```

Note: You can also create a context via an equivalent piece of JCL.

The following configuration settings indicate that when the adapter starts, it should write all of its object references to the Naming Service context called `test`, and subsequently remove the object references again on shutting down (during a normal shut-down):

```
plugins:imsa:write_iors_to_ns_context = "test";  
plugins:imsa:publish_all_iors = "yes";  
plugins:imsa:remove_ns_iors_on_shutdown = "yes";
```

Exporting Object References to Naming Service Object Group

Overview

When it comes to exporting object references to the Naming Service, the server adapter can be configured to export to either a Naming Service context or a Naming Service object group. This subsection provides details about exporting to a Naming Service object group.

Note: See the *Orbix Administrator's Guide* for more details on Naming Service object groups.

Prerequisites

If the server adapter is configured to export its object references to a set of Naming Service object groups, the following prerequisites apply:

- The Naming Service used must support the Orbix load balancing extensions to the Naming Service.
- The initial reference for the Naming Service must be available to the adapter either in the adapter's configuration file or via the command line at start-up.
- The object group must be predefined, so that the load balancing algorithm defined for each object group can be used—the load balancing algorithm might be round-robin, random, or some other custom load balancing algorithm.

Summary of rules

The following rules apply when mapping object references to a Naming service object group:

- An object group must be created for each object before the adapter is started; otherwise, the objects will not be exported. If you are unsure about the names of the object groups, start the adapter without any object groups created and check the error messages to see which object groups are needed.
- The object groups must then be bound to “objects”, so that clients can locate them. The fact that object groups are used is transparent to the clients.
- Each adapter must have a unique member name to ensure that it does not overwrite object group members created by other adapters.

- Members are only removed if the adapter shuts down normally; for example, by using an operator `stop` command, by using `itadmin mfa stop`, or by calling the `stop` operation on the adapter's `MappingGateway` interface.

Configuration

Both the `plugins:imsa:write_iors_to_ns_group_with_prefix` and `plugins:imsa:write_iors_to_ns_group_member_name` configuration items indicate that the adapter should write its object references to a Naming Service object group.

If a value of "" (that is, an empty prefix) is specified for `plugins:imsa:write_iors_to_ns_group_with_prefix`, the object references are written to object groups derived from the interface name only; otherwise, the prefix is attached to the derived names for each object group.

Note: The object groups must exist when the adapter is started. See the *Orbix Administrator's Guide* for details on how to create and specify nested Naming Service contexts.

The object reference for each interface is placed in the relevant object group, with the member name obtained from the `write_iors_to_ns_group_member_name` configuration variable. A unique member name must be specified for each adapter that is to use the set of object groups.

Object group name

The object group name used for each object bound into the Naming Service is derived from the module and interface name. First, all the module names are used and then the interface name, each separated by a colon. For example, the object group name for the interface for the `simple` demonstration is `Simple:SimpleObject`. If the prefix is not blank, it is attached to the start of each derived object group name before the object group is located in the naming service. The `MappingGateway` interface uses `IT_MFA` as the object group name.

rebind() function

The adapter uses `rebind()` to add the object references to the Naming Service, so any existing member in the object group is replaced.

Example

For example, consider the following configuration settings:

```
plugins:imsa:write_iors_to_ns_group_with_prefix = "group1_";
plugins:imsa:write_iors_to_ns_group_member_name = "adapter1";
plugins:imsa:publish_all_iors = "yes";
plugins:imsa:remove_ns_iors_on_shutdown = "yes";
```

Assuming the interface for the `simple` demonstration is the only one exported by the adapter, the following `itadmin` commands create object groups called `group1_IT_MFA`, `group1_IT_MFA_IMS:imsraw`, and `group1_Simple:SimpleObject`:

```
itadmin nsog create -type rr group1_IT_MFA
itadmin nsog create -type rr group1_IT_MFA_IMS:imsraw
itadmin nsog create -type rr group1_Simple:SimpleObject
```

Note: You can also create object groups via an equivalent piece of JCL.

Now, with the three round-robin object groups created, each needs to be bound to a context in the Naming Service, so that clients can locate the object references. For example, the following command creates a context called `testog`:

```
itadmin ns newnc testog
```

Each object group should be subsequently created in this context, using the following commands, so that clients can locate the objects:

```
itadmin nsog bind -og_name group1_IT_MFA testog/IT_MFA
itadmin nsog bind -og_name group1_IT_MFA_IMS:imsraw
testog/imsraw
itadmin nsog bind -og_name group1_Simple:SimpleObject
testog/simple
```

Based on the preceding command, the content of the `testog` context should now be listed as follows (when you specify an `itadmin ns list testog` command):

```
IT_MFA Object
imsraw Object
simple Object
```

If a client now resolves one of the object references before any adapter is started, a nil object will be returned. For example, consider the following command:

```
itadmin ns resolve testog/imsraw
```

If the preceding `itadmin` command is entered before an adapter is started, the following output is returned:

```
IOR:00000000000000010000000000000000
```

If the preceding `itadmin` command is entered after an adapter is started, the following output is returned:

```
IOR:000000000000000254944...
```

Running simultaneous adapters

If more than one adapter is started, each time `resolve()` is used it gives a different object reference, based on the load balancing algorithm specified when the object group was created. If all the adapters are stopped normally and `plugins:imsa:remove_ns_iors_on_shutdown` has been specified, `resolve` again returns a nil object reference.

Part 5

Securing and Using the Client Adapter

In this part

This part contains the following chapters:

Securing the Client Adapter	page 263
Using the Client Adapter	page 273

Securing the Client Adapter

This chapter provides details on how to secure the client adapter.

In this chapter

This chapter discusses the following topics:

Introduction to Client Security	page 264
APPC Security Considerations	page 265

Introduction to Client Security

Overview

This section introduces common security considerations and network security considerations for the client adapter. It discusses the following topics:

- [Common security considerations](#)
 - [Network security considerations](#)
-

Common security considerations

The client adapter is the bridge between an IMS client transaction and a target object. IMS and the client adapter use APPC to communicate with each other. The client adapter and the target object use TCP/IP to communicate with each other.

Securing the client adapter involves securing the APPC LU-LU connection between IMS and the client adapter, as well as securing the TCP/IP connection between the client adapter and the target object.

Network security considerations

The client adapter accepts APPC requests from Orbix IMS clients, and forwards these as GIOP requests over the network to the target server objects. By default, the GIOP traffic is transmitted over the insecure IIOp protocol. However, as with any Orbix application in your system, the client adapter can be configured to use IIOp over TLS (the Transport Layer Security protocol), thus ensuring a secure connection to each target server.

Refer to the *Mainframe Security Guide* for more information on how to secure applications such as the client adapter within your Orbix domain.

APPC Security Considerations

Overview

This section provides details on how to secure the client adapter in an APPC environment. APPC/MVS provides the following levels of security:

- LU security
- Conversation security

In this section

This section discusses the following topics:

LU Security	page 266
Conversation Security	page 269

LU Security

Overview

The client adapter processes client transactions from IMS. Therefore, IMS should be allowed to establish sessions with the client adapter. Other APPC applications on the network, however, are not intended to process requests via the client adapter. In some environments it might be considered a security breach if any application other than IMS establishes an APPC connection with the client adapter.

This subsection discusses the following topics:

- [Preventing non-IMS applications establishing sessions with the client adapter](#)
- [Defining VTAM APPLs for IMS and the client adapter](#)
- [Sample RACF APPCLU profile names for IMS and client adapter LUs](#)
- [Defining profiles for IMS and client adapter LUs example](#)
- [Activating the profiles in RACF](#)
- [Refreshing the profiles in VTAM](#)

Preventing non-IMS applications establishing sessions with the client adapter

To prevent applications other than IMS from establishing sessions with the client adapter, APPC LU security can be used. Enable APPC LU security by doing the following:

- Define the VTAM APPLs for IMS and the client adapter with the appropriate keywords.
- Define APPCLU RACF profiles.

Defining VTAM APPLs for IMS and the client adapter

Make sure the following keywords are defined on the VTAM APPL definition:

Table 7: *IMS LU and Client Adapter LU Required Keyword Definitions*

Keyword	Description
IMS LU required keyword definitions	

Table 7: IMS LU and Client Adapter LU Required Keyword Definitions

Keyword	Description
SECACPT=CONV	This keyword allows IMS to provide security information on a request to allocate a conversation. The security information includes the user ID making the request to allocate the conversation, the user's group ID, and an "already verified" indicator.
VERIFY=OPTIONAL	<p>If there is a RACF APPCLU profile defined for this LU, this keyword instructs VTAM to verify that the session keys defined in the RACF APPCLU profiles match for the IMS LU and the client adapter LU. If the keys do not match, the session between the IMS LU and the client adapter LU cannot be established.</p> <p>VERIFY=REQUIRED could be defined for even tighter security. However, an installation might be using the IMS server adapter as well as the client adapter, and the IMS server adapter security asks for VERIFY=OPTIONAL on the IMS LU APPL definition.</p>
Client Adapter LU required keyword definitions	
SECACPT=ALREADYV	This keyword allows the client adapter to receive security information on the conversation allocation request. The security information includes the user ID making the request to allocate the conversation, the user's group ID, and an "already verified" indicator.
VERIFY=REQUIRED	This keyword requires that a RACF APPCLU profile is defined for this LU and for any LU that attempts to establish a session with it. If RACF APPCLU profiles do not exist, the session cannot be established. If profiles do exist, the session keys in each profile must match; otherwise, the session cannot be established.

Sample RACF APPCLU profile names for IMS and client adapter LUs

The IMS LU and the client adapter LU require RACF APPCLU profiles. The names have the following pattern:

```
NETID.LU01.LU02
```

NETID represents your network ID. *LU01* and *LU02* are the LU names to be secured. Each LU requires its own profile. The profile name in the preceding example would be for *LU01*. The profile name for *LU02* would be as follows:

```
NETID.LU02.LU01
```

Defining profiles for IMS and client adapter LUs example

The following is an example of defining the profiles for the IMS LU and the client adapter LU:

```
RDEFINE APPCLU P390.IMSLU01.ORXLUCA1
UACC(NONE) SESSION(SESSION(137811C0) CONVSEC(ALREADYV))

RDEFINE APPCLU P390.ORXLUCA1.IMSLU01
UACC(NONE) SESSION(SESSION(137811C0) CONVSEC(ALREADYV))
```

Activating the profiles in RACF

To activate the profiles in RACF, use the following command:

```
SETROPTS CLASSACT(APPCLU)
```

Refreshing the profiles in VTAM

To refresh the profile in VTAM, use the following VTAM command:

```
F VTAM,PROFILES, ID=IMSLU01
F VTAM,PROFILES, ID=ORXLUCA1
```

In the preceding example, *VTAM* is the name of the procedure used to start VTAM.

Note: Although APPC can be used for networked communication, the client adapter is only intended to be run on the same machine as the IMS region with which it is communicating.

Conversation Security

Overview

There are three levels of conversation security:

- `security_none`
- `security_same`
- `security_pgm`

The Orbix runtime inside IMS uses `security_same` when allocating its conversations with the client adapter.

A conversation using `security_pgm` is not possible with the client adapter, because the Orbix runtime inside IMS has no access to client passwords.

Refer to [“LU 6.2 conversation security levels” on page 172](#) for more details on each conversation security level.

This subsection discusses the following topics:

- [Controlling access to the client adapter LU](#)
- [Activating the APPL class](#)
- [Refreshing the RACLIST](#)
- [Controlling access to the IMS LU](#)

Controlling access to the client adapter LU

Some environments might want very strict controls regarding which users are permitted access to the client adapter. A RACF `APPL` class can be defined for the client adapter LU specifying a universal access of `NONE`. Individual users can then be permitted access to the client adapter LU.

An example of defining the RACF `APPL` class is as follows:

```
RDEFINE APPL ORXLUCA1 UACC(NONE)
```

Individual users can then be permitted access to the client adapter LU:

```
PERMIT ORXLUCA1 CLASS(APPL) ID(USER1) ACCESS(READ)
PERMIT ORXLUCA1 CLASS(APPL) ID(USER2) ACCESS(READ)
...
```

Note: To allow IMS to provide the user ID of the user that is running the transaction, rather than the user ID of the user that started the IMS control region, IMS exit `DFSBSEX0` must be used. See the IBM publication *IMS/ESA Customization Guide, SC28-8732* for more details.

Activating the APPL class

Activate the `APPL` class as follows:

```
SETROPTS CLASSACT(APPL) RACLIST(APPL)
```

Refreshing the RACLIST

Refresh the `RACLIST` as follows:

```
SETROPTS RACLIST(APPL) REFRESH
```

Controlling access to the IMS LU

Access to the client adapter LU can be controlled by controlling access to the IMS LU that wants to establish communications with the client adapter LU. The IMS LU is considered an `APPC` port of entry and can be secured with the `RACF APPCPORT` class.

Define the `APPCPORT` profile for the IMS LU as follows:

```
RDEFINE APPCPORT IMSLU01 UACC(NONE)
```

This profile defines a universal access of `NONE` to the IMS LU. To permit access to users, use the `RACF PERMIT` command:

```
PERMIT IMSLU01 CLASS(APPCPORT) ID(USER1) ACCESS(READ)
PERMIT IMSLU01 CLASS(APPCPORT) ID(USER2) ACCESS(READ)
...
```

Activate the `APPCPORT` class as follows:

```
SETROPTS CLASSACT(APPCPORT) RACLIST(APPCPORT)
```

When changes are made to an `APPCPORT` profile, refresh the profile for the change to take effect:

```
SETROPTS RACLIST(APPCPORT) REFRESH
```

Note: To allow IMS to provide the user ID of the user that is running the transaction, rather than the user ID of the user that started the IMS control region, IMS `exit` `DFSBSEX0` must be used. See the IBM publication *IMS/ESA Customization Guide, SC28-8732* for more details.

Using the Client Adapter

This chapter provides information on running and using the client adapter. It provides details on how to start and stop the client adapter, and also provides details on how to run multiple client adapters.

In this chapter

This chapter discusses the following topics:

Starting the Client Adapter	page 274
Stopping the Client Adapter	page 276
Running Multiple Client Adapters Simultaneously	page 277

Starting the Client Adapter

Overview

This section describes how to start the client adapter. It discusses the following topics:

- [Starting the client adapter on native OS/390](#)
 - [Starting the client adapter on OS/390 UNIX System Services](#)
-

Starting the client adapter on native OS/390

In a native OS/390 environment, you can start the client adapter in any of the following ways:

- As a batch job.
- Using a TSO command.
- As a started task (by converting the batch job into a started task).

The default client adapter is the client adapter for which configuration is defined directly in the `iona_services.mfu` scope, and not in some sub-scope of this. The following is sample JCL to run the default client adapter:

```
//MFCLA JOB ( ),
//      CLASS=A,
//      MSGCLASS=X,
//      MSGLEVEL=(1,1),
//      NOTIFY=&SYSUID,
//      REGION=0M,
//      TIME=1440
//*
//      JCLLIB ORDER=(HLQ.ORBIX60.PROCS)
//      INCLUDE MEMBER=(ORXVARS)
//*
/** Make the following changes before running this JCL:
/**
/** 1. Change 'SET DOMAIN='DEFAULT@' to your configuration
/**     domain name.
/**
//          SET DOMAIN='DEFAULT@'
/**
```

```

/** Run the Orbix Client Adapter
/**
//GO EXEC PROC=ORXG,
// PROGRAM=ORXMFUA,
// PPARM=' '
//ITDOMAIN DD DSN=&ORBIX..CONFIG(&DOMAIN),DISP=SHR

```

Starting the client adapter on OS/390 UNIX System Services

On OS/390 UNIX System Services, you can start the client adapter from the shell. The following command is used to run the default client adapter:

```
$ itmfua
```

Running with a different configuration scope

To run the client adapter with a different configuration scope:

- On native OS/390, set the value of `PPARM` to the new scope. For example:

```
PPARM='--ORBname iona_services.mfu_test'
```

- On OS/390 UNIX System Services, run a command similar to the following:

```
$ itmfua -ORBname iona_services.mfu_test
```

Refer to [“Running Multiple Client Adapters Simultaneously” on page 277](#) for more details on running multiple client adapters.

Stopping the Client Adapter

Overview

This section describes how to stop the client adapter. It discusses the following topics:

- [Stopping the client adapter on native OS/390](#)
- [Stopping the client adapter on OS/390 UNIX System Services](#)

Stopping the client adapter on native OS/390

To stop a client adapter job on native OS/390, issue the `STOP (P)` operator command from the console.

Stopping the client adapter on OS/390 UNIX System Services

To stop a client adapter process on OS/390 UNIX System Services, use the `kill` command or press **Ctrl-C** if it is running in an active rlogin shell.

Running Multiple Client Adapters Simultaneously

Overview

This section describes how to run multiple client adapters simultaneously.

In this section

This section discusses the following topics:

Load Balancing with Multiple Client Adapters	page 278
Running Two Client Adapters on the Same OS/390 Host	page 280

Load Balancing with Multiple Client Adapters

Overview

The client adapter is a multithreaded application that can concurrently service multiple requests. However, an installation can choose to run multiple client adapters to spread the workload over multiple address spaces.

This subsection discusses the following topics:

- [Load balancing scenario](#)
- [Graphical overview](#)
- [Load balancing scenario explanation](#)

Load balancing scenario

Suppose there are three IMS regions that can run client transactions to be processed via the client adapter. An installation might choose to run two client adapters to process the load. If one of the client adapters is stopped, the other can still service client requests from IMS.

Graphical overview

[Figure 7](#) illustrates the load balancing scenario.

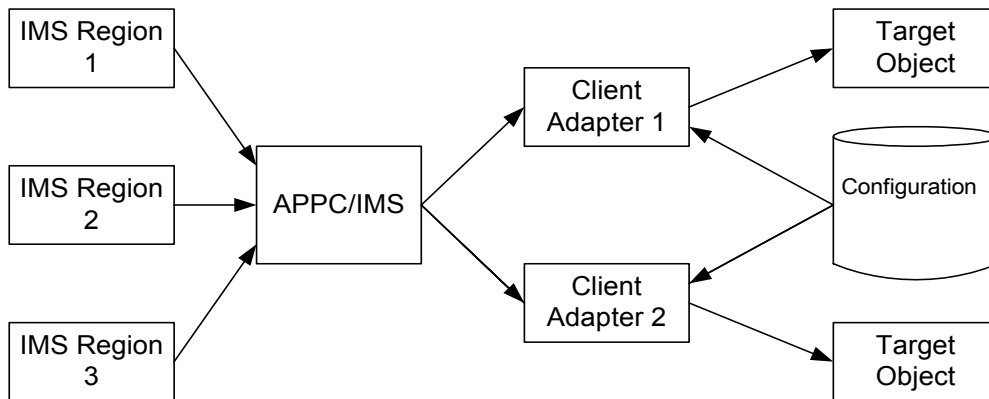


Figure 7: *Graphical Overview of a Load Balancing Scenario*

**Load balancing scenario
explanation**

Each IMS region contains an Orbix runtime. Each Orbix runtime has a configuration that specifies the same symbolic destination. The symbolic destination determines the client adapter that IMS client transaction requests are being directed to. From the IMS perspective, it appears as if there is only one client adapter running.

APPC/MVS processes the IMS client transaction requests. It queues the requests in an allocation queue. The allocation queue is determined by the symbolic destination. Because all IMS regions are using the same symbolic destination, IMS client transaction requests are directed to a single allocation queue.

Both client adapters are using the same configuration file and same configuration scope. Therefore, they are using the same symbolic destination, and share the same allocation queue that APPC/MVS uses for IMS client transaction requests. Each client adapter has one or more threads that are waiting for allocation requests from APPC/MVS, all from the same allocation queue.

APPC/MVS hands off an allocation request to a thread in one of the client adapters. Determining which thread to give an allocation request to is an internal function of APPC/MVS. Therefore, it is APPC/MVS that spreads the load over the two client adapters. If one of the client adapters is stopped, APPC/MVS hands off all allocation requests to the client adapter that is still running.

Running Two Client Adapters on the Same OS/390 Host

Overview

An installation might choose to run a test and production client adapter on the same OS/390 host. In this scenario, it is not desirable for the client adapters to share the APPC/MVS allocate queues.

This subsection discusses the following topics:

- [Running a test and production client adapter on the same host](#)
 - [Graphical overview](#)
 - [Setting up a test and a production client adapter on the same host](#)
-

Running a test and production client adapter on the same host

Each IMS region contains an Orbix runtime. Each Orbix runtime has a configuration that specifies different symbolic destinations. The production IMS region is configured to communicate with the production client adapter. The test IMS region is configured to communicate with the test client adapter.

APPC/MVS processes the IMS client transaction requests. It queues the requests to separate allocation queues—one for the production client adapter using the production symbolic destination, and one for the test client adapter using the test symbolic destination.

Both client adapters are using the same configuration file but different configuration scopes. The configuration scopes define different symbolic destinations. Therefore, the client adapters each have their own allocation queues.

Graphical overview

Figure 8 illustrates how two client adapters can run on the same OS/390 host.

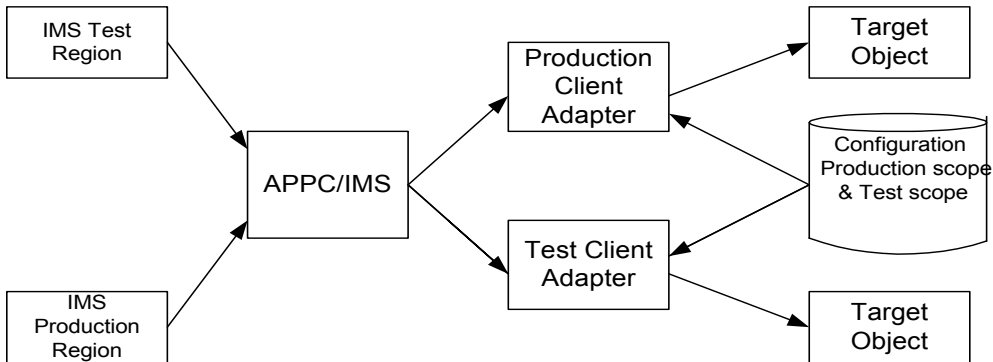


Figure 8: *Running Two Client Adapters on the Same OS/390 Host*

Setting up a test and a production client adapter on the same host

The steps to set up a test and production client adapter on the same OS/390 host are as follows:

Step	Action
1	Define separate symbolic destinations in APPC/MVS for the test and production client adapters to use. Refer to “Defining an APPC Destination Name for the Client Adapter” on page 130 for more information on defining symbolic destinations.
2	Configure the Orbix runtime inside IMS for the test and production regions. The test region is configured with the test symbolic destination. The production region is configured with the production symbolic destination. Refer to “Customizing Orbix APPC Symbolic Destination” on page 158 for more information on configuring the symbolic destination.

Step	Action
3	Define a test configuration scope in the client adapter configuration file such as <code>iona_services.test_mfu</code> . The existing <code>iona_services.mfu</code> configuration scope can be used for production. Set the test symbolic destination in the test configuration scope, and set the production symbolic destination in the production configuration scope. Refer to “Customizing Orbix APPC Symbolic Destination” on page 158 for more information on configuring the symbolic destination.
4	Start the production client adapter without specifying any configuration scope. It runs with the default scope of <code>iona_services.mfu</code> . Start the test client adapter specifying the test configuration scope defined in step 3, that is, <code>iona_services.test_mfu</code> . Refer to “Starting the Client Adapter” on page 274 for more information on running the client adapter with a different configuration scope.

Glossary of Acronyms

This glossary provides an expansion for each of the acronyms used in this guide.

For more details of each of these terms, refer to the following, as appropriate:

- The IBM documentation series at <http://www.ibm.com>.
- The OMG CORBA specification at <http://www.omg.org>.
- The Sun Microsystems J2EE specification at <http://www.sun.com>.

Table 8: *Glossary of Acronym Extensions*

Acronym	Extension
ACB	Access Control Block
ACEE	Accessor Environment Entry
APAR	Application Program Authorized Report
APPC	Advanced Program to Program Communication
ASCII	American National Standard Code for Information Interchange
C/I	Callable Interface
CORBA	Common Object Request Broker Architecture

Table 8: *Glossary of Acronym Extensions*

Acronym	Extension
DASD	Direct Access Storage Device
DLL	Dynamic Link Library
EBCDIC	Extended Binary-Coded Decimal Interchange Code
EJB	Enterprise Java Beans
GIOP	General Inter-ORB Protocol
HFS	Hierarchal File System
IDL	Interface Definition Language
IFR	Interface Repository
IIOP	Internet Inter-ORB Protocol
IMS	Information Management System
IOR	Interoperable Object Reference
IPL	Initial Program Load
JCL	Job Control Language
LE	Language Environment
LU	Logical Unit
MVS	Multiple Virtual Systems
OMG	Object Management Group
OMVS	Open Multiple Virtual Systems
ORB	Object Request Broker
OTMA	Open Transaction Manager Access
OTS	Object Transaction Service
PADS	Program Access to Data Sets
PCB	Program Control Block

Table 8: *Glossary of Acronym Extensions*

Acronym	Extension
PDS	Partitioned Data Set
PSB	Program Specification Block
PWFI	Pseudo Wait-for-Input mode
RACF	Resource Access Control Facility
RRS	Resource Recovery Services
SAF	System Authorization Facility
SNA	System Network Architecture
SPA	Save Program Area
TCP/IP	Transmission Control Protocol over Internet Protocol
TP	Transaction Program
TPN	Transaction Program Name
TLS	Transport Layer Security
TSO	Time Sharing Option
USS	UNIX System Services
VTAM	Virtual Telecommunications Access Method
WFI	Wait For Input
WTO	Write-To-Operator
XCF	Cross-Coupling Facility

Index

A

- abends
 - SF92 66
 - User 119 69
- ACBNAME= parameter 78
- accounting DLL
 - compiling 248
 - developing 245
 - loading into Server Adapter 249
- amtp_appc plug-in 120
- AMTP function timeout 144
- APAR's for OTMA 64
- APPC/IMS, enabling 82
- APPC communication threads 144
- APPC destination 144
- APPC destination names 73, 75
 - defining 130
 - multiple 132
- APPCHOST logon mode 76, 131
- APPC local LU, customizing 160
- APPCLU class profiles 174
- APPC LU name for IMS region 83
- APPCLU profile name format 80
- APPCLU profiles, accessing 81
- APPC-side information data set example 75
- APPC symbolic destination name 128
- APPL class, activating 270
- APPL definition of the VTAM ACB 174
- APPL macros 77
- APPL statement label 78
- ASCII-to-EBCDIC
 - binary format 238
 - plugins:imsa:use_client_principal 98
- ASCII-to-EBCDIC translation 30, 31
- ATBSDFMU APPC/MVS utility program 75, 130
- authorization checks, client 170

B

- BPX.SERVER 100
 - Server Adapter user ID 103
- BPX.SERVER RACF FACILITY class profile 180
- ByteSegments 29, 30, 31

C

- CharSegments 29, 30, 31
- Client Adapter
 - APPC security 265
 - characteristics and functions 36
 - common and network security 264
 - defining new configuration scope 282
 - event logging 123
 - graphical overview 38
 - load balancing 278
 - multiple Client Adapters 128
 - multiple on same OS/390 host 280
 - starting 274
 - stopping 276
- Client Adapter LU 127
 - access to 141, 269
 - RACF APPCLU profiles 268
- Client Adapter LU name and APPCLU profile name 128
- Client Adapter LU required keywords 267
- client_principal support, configuring 97
- Client Principal value
 - user ID 170
- client Principal value 170
- clients 9
 - invoking on CORBA objects 10
 - using naming service 14
- client stub code 10
- Configuration domains 17
- configuration scope example 44
- configuration templates 44
- conversation security, Client Adapter 269
- conversation security levels for LU 6.2 172
- CORBA 5
 - application basics 10
 - introduction to 4
- CORBA::Principal 97
- CORBA objects 7
 - and IDL 8
 - client invocations on 10
- COUPLExx parmlib member 87
- coupling facility log streams 88
- CSECT 109, 153

D

DASD-only log streams 88
 data types for imsrw IDL interface 29
 DESTNAME 76, 131
 DFSPBxxx 64
 din parameter 30
 DIS OTMA command 67
 DLL names for portable interceptor 239
 do_trans() operation 26

E

EBCDIC, translating from ASCII 30, 31
 EPERM errors 103
 errors in execution of IMS transactions 29
 event_log:filters 57, 123
 event logging 109, 153
 exceptions
 address space not programmed controlled 104
 imsraw 29
 portable interceptor 238
 server interceptor and 238

F

function timeout, customizing 157

G

GIOP
 12 by fixed header 245
 CORBA::Principal 97
 security 264
 use_client_principal 98
 GRNAME parameter 64

I

IDL compiler 10
 configuration file 112
 -mfa plug-in 112
 operation parameters 11
 IDL exceptions, imsrw interface 29
 IDL interfaces 8
 located for Server Adapter 25
 IDL operations 11
 parameter passing modes 11
 Server Adapter processing of 25
 IDL types, unsupported 35
 IEFSSNxx member 93

IFR 17

 introduction to 193
 modifications to and the Server Adapter 198, 207
 registering IDL with 195
 IFR cache file 200
 IFR cache file configuration items 60
 IIOP 5
 imsa plug-in configuration 49
 mapping gateway interface 222
 network security 264
 TCP-IP port number 56
 timestamps 59
 imsa plug-in configuration items 48
 ims_appc plug-in configuration items 53
 IMS control region
 and RRS 93
 IMS JCL, customizing 108
 IMS local LU
 access to 141
 IMS LU
 access to 270
 IMS LU, access to 82
 IMS LU definition 73
 IMS LU definition, Client Adapter 127
 IMS LU name and APPC destination name 75
 IMS LU required keyword 266
 IMS message queue length 69
 IMS message queue length configuration 84
 IMS on APPC functions, non-Orbix 74
 ims_otma plug-in configuration items 52
 imsrw IDL interface 26
 ByteSegments attribute 29, 30, 31
 C++ demo client 34
 CharSegments attribute 29, 30, 31
 din parameter 30
 maxSegmentSize attribute 31
 run_transaction_binary operation 30
 run_transaction operation 30
 tran_name parameter 30
 ims_rrs plug-in configuration items 53
 IMS security mechanisms 170
 IMS versions and OTMA callable interface 64
 IMSXCF.group.member 177
 IMSXCF.OTMACI 66, 177
 initial_references:IT_imsraw
 plugin 56
 initial_references:IT_ismraw:plugin 48
 initial_references:IT_RRS:plugin 94
 Interface Definition Language See IDL

Interface Repository See IFR
 IONAIMS 67
 iona_services.imsa configuration scope example 44
 iona_services.mfu configuration scope sample 119
 IORs 17
 IT_MFA 215
 IT_MFA_IMSRAW 215
 transactional processing support 86
 IsDefault setting 113
 itadmin mfa commands 222
 IT_MFA_display_account_information()
 parameters 245
 sample use of 246
 IT_MFA events 57
 IT_MFA_IMS module 26
 itmfaloc
 format 226
 using 225
 IT_MFU event logging subsystem 123
 IXCL1DSU 88
 IXCMIAPU utility 92

L

Location domains 16
 locator daemon 16
 LOGLVL variable 110
 LOGR couple data set 29
 log streams 88
 defining 92
 IBM recommended sizes 89
 LU 6.2 conversations
 securing 175
 security levels 172
 LU 6.2 networked Server Adapter, unauthorized
 access 173
 LU 6.2 protocol 171
 LU-LU session-level verification 78
 LU names
 access to default mode 180
 LU names, restricting use of 175
 LUs
 Client Adapter 268
 defining to VTAM 134
 protecting 141
 VTAM requirements for 77

M

mapping file 25
 and attributes 188
 characteristics and format 187
 generating 190
 runtime modifications 191
 Mapping Gateway interface 222
 maxSegmentSize attribute 31
 MFACLINK JCL member 110
 MFAMappingExtension 113
 MFAMappingSuffix 114
 mfa plug-in
 generating a mapping file 189
 mfu plug-in 121
 MODENAME parameter 76, 131

N

naming service 14
 node daemon 16

O

object ID 14
 object references 8, 13
 and the POA 14
 ORB (Object Request Broker) 9
 and the naming service 14
 ORB_init() 57
 ORBINITI files 235
 Orbix 5
 Orbix application 13
 Orbix IMS runtime
 configuration 109
 parameter-passing modes 11
 Orbix security mechanisms 170
 OrbixSSL for OS/390 184
 ORBname 44
 orb_plugins 57
 Client Adapter 124
 portable interceptor 242
 ORXACCT1 DLL 248
 configuration 249
 ORXMFACx 151
 ORXMFACx DLL 110, 154
 OTMA C/I, activating 65
 OTMAINIT procedure 66
 OTMANM parameter 64
 OTMA parameter 64
 output segments 69

P

- parameter-passing mode qualifiers 11
- PARM1 64
- PARTNER_LU 76
 - Client Adapter 131
- partner LUs 80
- passwords
 - bypassing 81
 - requirements 78
- PLEXCFG parameter 87
- plugins:amtp_appc:function_wait 144
- plugins:amtp_appc:max_comm_threads 145
- plugins:amtp_appc:min_comm_threads 144
- plugins:amtp_appc:symbolic_destination 144
- plugins:imsa:call_accounting_dll 249
- plugins:imsa:capture_first_argument_in_dynany 24
 - 9
- plugins:imsa:direct_persistence 56
- plugins:imsa:display_timings 58
- plugins:imsa:ifr:cache 60, 200
- plugins:imsa:mapping_file 59
- plugins:imsa:poa_prefix 58
- plugins:imsa:repository_id. 59
- plugins:imsa:type_info:source 60
- plugins:imsa:use_client_principal 98
 - and security 180
- plugins:ims_appc:appc_outbound_lu_name 83
- plugins:ims_appc:ims_destination_name 83
- plugins:ims_appc:mq_length 84
- plugins:ims_appc:timeout 83
- plugins:ims_appc:use_client_principal 174
- plugins:ims_otma:mq_length 69
- plugins:ims_otma:output_segment_num 69
- plugins:ims_otma:timeout 68
- plugins:ims_otma:use_client_principal 177
- plugins:ims_otma:xcf_adapter_member_name 67
- plugins:ims_otma:xcf_group_name 67
- plugins:ims_otma:xcf_ims_member_name 68
- plugins:ims_otma:xcf_tpipe_name 68
- plugins:ims_otma:xcf_tpipe_prefix 68
- plugins:mfu:target_cache_size 147
- plugins:rrs:rmname 94
- PLUGIN source file 235
- POA (Portable Object Adapter) 14
- POA name 58
- policies:giop
 - interop_policy:enable_principal_service_context 9
 - 8
 - interop_policy:principal_service_context_id 99

- policies:iiop:server_version_policy 98
- portable interceptor
 - compiling 239
 - developing 235
 - loading into Server Adapter 241
- portable object adapter See POA
- pragma prefix 29
- PREPIMSA member 213
- PresetOptions setting 113
- principal values, mapping to OS/390 user IDs 170
- proxy objects 11
- pthread_security_np() 102
- pthread_security_np() service 180

R

- RACF 172
- RACF APPCLU profiles, defining 80
- RACF APPCPORT class 175
- RACF FACILITY class 177
- RACF group profile and security 182
- RACF security calls 175
- RACF user profile 100
- receive_request_service_contexts() interceptor 238
- RRS
 - starting and stopping 93
- RRS Server Adapter resource manager name 94
- run_transaction 30
- run_transaction_binary 30

S

- SAF Plug-In 170
- SECACPT=CONV key 173
- SECACPT= parameter 78, 135
- SECURE APPC CHECK 175
- SECURE APPC FULL 175
- SECURE APPC NONE 175
- SECURE APPC PROFILE 176
- SECURE OTMA FULL 178
- SECURE OTMA NONE 178
- SECURE OTMA PROFILE 178
- security
 - APPC, default for 180
 - APPC, setting up 183
 - APPC issues 171
 - choosing OTMA or APPC 182
 - common issues 169
 - determining mode 179
 - OTMA, setting up 184

- OTMA issues 177
 - user already logged on 173
- security_none 172
- security_none conversation 82
- security_pgm 173
- security_same 172
- segment size, customizing 155
- send_exception() interceptor point 238
- Server Adapter
 - adding a portable interceptor 234
 - characteristics and functions 22
 - first run 200
 - gathering account information 244
 - graphical overview 24
 - LU name 73
 - obtaining operation signatures 25
 - passwords 80
 - plug-ins 48
 - preparing to run 213
 - running multiple 74, 220
 - selecting a specific 224
 - starting 217
 - stopping 219
 - using previous versions 26
 - WFI mode 228
- Server Adapter ORBname 44
- Server Adapter outbound LU name 83
- servers 9
- service context ID 99
- session key 80
 - and APPCLU class profiles 175
- session-level verification 174
- SETRRS CANCEL command 93
- SETSSI ADD,SUBNAME=RRS command 93
- SETXCF operator commands 91
- SF92 abend 66
- SIADD statement 132
- skeleton code 10
- SNA network 77
- SPA 31
- S RRS command 93
- SRVINTRC files 235
- SYS1.MIGLIB 92
- SYS1.SAMPLIB(ATBAPPL) definition 77, 134
- sysplex and RRS 87
- System Logger and RRS 89

T

- target cache size 147

- thread-level security environments 100
- thread_pool
 - high_water_mark 54
 - initial_threads 54
- TIMS RACF class 177
- TPIPE prefix 64
- TPIPE prefix name 68
- TPNAME specification 76, 131
- tran_name parameter 30
- transactional requests 86
- transaction processing times, displaying 58
- transaction request timeout 83
- TypeinfoFileExtension 113
- TypeinfoFileSuffix 113
- type information mechanism, configuration items 59
- type_info store
 - configuration 60
 - generating files 205
 - introduction 203

U

- use_client_principal security mode 182
- User 119 Abend 69
- user ID
 - client principle values 170
 - determining 179
 - partner LUs 80

V

- VERIFY= parameter 78, 136
- VSAM data set APPC-side information 73
- VTAM logon mode table 131

W

- WFI support 228
- WTO announce plug-in 57
 - Client Adapter 124

X

- XCF couple data set format utility 88
- XCF group, joining 177

