



Mainframe Migration and Upgrade Guide

Version 6.2, May 2005

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 © 2001–2005 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: 08-Jun-2005

Contents

List of Tables	ix
Preface	xi
Part 1 Overview	
Chapter 1 Introduction	3
Advantages of Orbix 6.2	4
Migration Resources	6
Part 2 Migrating from 2.3.x	
Chapter 2 Migration Possibilities and Main Differences	9
Migration Possibilities	10
C++ Applications	11
COBOL and PL/I Applications	12
Chapter 3 Installation Requirements	15
Chapter 4 IDL Migration Issues	19
The Opaque Type	20
IDL Fixed Type Definitions	21
IDL Defined in Fixed Block Data Sets	22
imsraw and cicsraw IDL changes	23
Orbix 6.2 C++ IDL Compiler Output	25

Chapter 5 C++ Migration Issues	27
C++ Compiler Issues	28
C++ Client Migration	29
CORBA Object Location and Binding	30
Interface Repository Interoperability	35
IDL-to-C++ Mapping	36
Client-Side CORBA Compliancy	37
Callback Objects	39
System Exception Semantics	40
Dynamic Invocation Interface (DII)	41
C++ Server Migration	42
BOA to POA Migration	43
Activation Modes	45
Object/Servant Lifecycles	46
Creating Object References Without Servants	48
Function Signatures	49
Exception-Safe Servant Implementations	50
Migrating Proprietary 2.3 Features	51
Orbix Filters and CORBA 2.3 Alternatives	52
Transformers	56
Orbix-Specific APIs	57
Connection Management	58
Callbacks and Bidirectional GIOP	60
Chapter 6 COBOL Migration Issues	61
Name Mapping Issues	63
Fully Qualified Level 01 Data Names	64
Operation and Level 88 Data Names	68
IDL Constant Definitions Mapped to Fully Qualified Names	72
Derived Interface Names and Fully Qualified Names	77
Numeric Suffixes for Data Names	80
160-Character Limit for String Literals	81
Maximum Length of COBOL Data Names	86
Copybook Names Based on IDL Member Name	89
Introduction to IDL Member Name Migration Issues	90
IDL Member Name Different from its Interface Names	92
More than One Interface in an IDL Member	94
Length of IDL Member Names	96

Name Scoping and the COBOL Compilers	97
Same Container Name Used More than Once	98
Same Fieldname Used More than Once	105
Typecode Name and Length Identifiers	107
Comparing Compiler Output	108
IDL Member Name Different from its Interface Name	109
More than One Interface in an IDL Member	112
Reserved COBOL and OMG Keywords	116
Reserved COBOL Keywords for Module or Interface Names	117
Use of Result as an Argument Name in IDL	118
OMG Mapping Standard for Unions and Exceptions	120
Error Checking and Exceptions	122
COBOL-Specific Issue Relating to Error Checking	123
Error Checking Generation at Runtime for Batch Servers	125
Nested Unions in IDL	126
Mapping for Arrays	131
Working Storage data Items and Group Moves	133
Mapping for IDL type Any	135
CORBA Copybook Additions	137
Parameter Passing of Object References in IDL Operations	138
CORBA Object Location and Binding	139
Migration Overview and Example	140
The Naming Service	142
Object-String Conversion	144
API Migration Issues	145
Deprecated APIs	146
ORBEXEC and USER Exception parameters	147
ORBSTAT	148
ORBALLOC	149
COBOL IMS Server Migration Issues	151
Server Mainline Program Requirement for IMS Servers	152
The Linkage Section for IMS Servers	156
Access to the Program Communication Block for IMS Servers	162
Error Checking Generation at Runtime for IMS Servers	164
COBOL IMS Client Migration Issues	165
The Linkage Section for IMS Clients	166
Error Checking Generation at Runtime for IMS Clients	168
Extra Copybooks in Orbix 6.2 for IMS Clients	169

COBOL CICS Server Migration Issues	171
Server Mainline Program Requirement for CICS Servers	172
Access to the EXEC Interface Block Data Structure	176
Error Checking Generation at Runtime for CICS Servers	177
COBOL CICS Client Migration Issues	178
Error Checking Generation at Runtime for CICS Clients	179
Extra Copybooks in Orbix Mainframe 6.2	180
Miscellaneous	181
Chapter 7 PL/I Migration Issues	183
Fully Qualified Level 1 Data Names	185
Maximum Length of PL/I Data Names	188
IDL Constant Definitions Mapped to Fully Qualified Names	192
Typecode Name and Length Identifiers	195
Include Member names Based on the IDL Member name	196
IDL Member names Different from Interface Names	199
More than One Interface in an IDL Member	201
Reserved PL/I Keywords for Module or Interface Names	203
Orbix PL/I Error Checking	204
CORBA Object Location and Binding	205
Migration Overview and Example	206
Naming Service	208
Object-String Conversion	210
CORBA Include Member Additions	211
API Migration Issues	212
Deprecated APIs	213
PODSTAT in Orbix 6.2	214
PODEXEC and User Exception parameters	215
Server Accessor (Z Member)	216
PL/I IMS Server Migration Issues	222
Server Mainline Module	223
Access to the Program Communication Block	228
PL/I IMS Client Migration issues	230
Program Communication Block Definitions Modifications	231
DLIDATA Include Member Modifications	234
Error Checking Generation at Runtime for IMS Clients	235
PL/I CICS Server Migration Issues	236
Server Mainline Program Requirements for CICS Servers	237
Access to the EXEC Interface Block Data Structure	242

PL/I CICS Client Migration Issues	243
Miscellaneous	244
Chapter 8 Diagnostic Output	247
Chapter 9 CORBA Services	249
Naming Service	250
Interface Repository	251
IMS Adapter	252
CICS Adapter	254
Chapter 10 Administrative Tools	257
Chapter 11 Interoperability	261
Use of the Orbix Protocol	262
GIOP Versions	263
Launch and Invoke Rights	265
Codeset Negotiation	267
Introduction to Codeset Negotiation	268
Configuring Codeset Negotiation	269
Default Codesets	270
Configuring Legacy Behavior	273
Part 3 Migrating from 5.1	
Chapter 12 Upgrading from Mainframe Edition 5.x	277
Installation Requirements	279
Configuration Changes	281
New Node Daemon	289
Database Migration	290
C++ Migration	291
COBOL Migration	293
PL/I Migration	294

Part 4 Migrating from 6.0

Chapter 13 Upgrading from Orbix Mainframe 6.0	299
Installation Requirements	301
Configuration Changes	303
Database Migration	308
C++ Migration	309
COBOL Migration	311
PL/I Migration	312
JCL Updates	314

Part 5 Migrating from Artix Mainframe Developer 2.0

Chapter 14 Upgrading from Artix Mainframe Developer 2.0	319
Index	1

List of Tables

Table 1: Migration Possibilities for z/OS	10
Table 2: Differences in PDS Naming Conventions	17
Table 3: C++ Compiler Output Comparison for UNIX System Services	25
Table 4: POA Policy Types and Their Values for Callback Objects	39
Table 5: Migrated System Exceptions	40
Table 6: COBOL Compiler Output for IDL Constant Definitions	72
Table 7: COBOL Compiler Output for GRID IDL Member	95
Table 8: COBOL Mapping Changes for IDL Data Types	133
Table 9: Deprecated COBOL APIs and Their Replacements	146
Table 10: ORBALLOC and Mapping Changes for IDL Data Types	149
Table 11: Extra Copybooks that ship with Orbix 6.2	169
Table 12: Extra Copybooks that ship with Orbix 6.2	180
Table 13: PL/I Compiler Output for IDL Constant Definitions	192
Table 14: PL/I Compiler Output Comparison GRID Include Member Names	199
Table 15: PL/I Compiler Deprecated IDL Generated Members and Their Replacements	202
Table 16: Deprecated PL/I APIs and Their Replacements	213
Table 17: Differences in Controlling OTMA-Based IMS Adapters	252
Table 18: Differences in Controlling APPC-Based IMS Adapters	253
Table 19: Differences in Controlling EXCI-Based CICS Adapters	254
Table 20: Differences in Controlling APPC-Based CICS Adapters	255
Table 21: CORBA-Specified Minimum GIOP Versions	263
Table 22: Orbix-Specific Minimum GIOP Versions	264
Table 23: Default GIOP Version Used by Orbix Clients	264
Table 24: CORBA Codeset Configuration Variables (Orbix 6.2)	269
Table 25: CORBA C++ Codesets (Non-MVS Platforms)	270
Table 26: CORBA C++ Codesets (MVS Platform)	271

LIST OF TABLES

Table 27: CORBA Java Codesets (ISO-8859-1/Cp-1292/US-ASCII locale)	271
Table 28: CORBA Java Codesets (Shift_JIS locale)	271
Table 29: CORBA Java Codesets (EUC_JP locale)	272
Table 30: CORBA Java Codesets (other locale)	272
Table 31: Differences in PDS Naming Conventions	280
Table 32: Differences in PDS Naming Conventions	302

Preface

Overview

This guide describes the issues that surround the migration of applications from earlier IONA mainframe solutions to an Orbix Mainframe 6.2 solution. [Part 1](#) provides an introduction to Orbix Mainframe migration. The bulk of this guide ([Part 2](#)) focuses on migrating from Orbix 2.3.x-based solutions, because much fewer changes are required to migrate from Orbix E2A Mainframe Edition 5.x. [Part 3](#) describes Orbix E2A Mainframe Edition 5.x migration issues. [Part 4](#) describes Orbix Mainframe 6.0 migration issues. [Part 5](#) describes Artix Mainframe Developer 2.0 migration issues.

This guide describes migration issues relating specifically to COBOL and PL/I applications in a native z/OS environment, and to C++ applications in both a native z/OS and UNIX System Services environment.

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 application programmers who want to migrate their applications from earlier IONA mainframe solutions to an Orbix Mainframe 6.2 solution. It is assumed that the reader is familiar with the basic concepts of CORBA 2.6.

Related Documentation

Orbix Mainframe 6.0 documentation includes the following related guides:

- *CORBA Programmer's Guide, C++*
- *CORBA Programmer's Reference, C++*
- *COBOL Programmer's Guide and Reference*

- *PL/I Programmer's Guide and Reference*
- *CORBA Administrator's Guide*
- *IMS Adapters Administrator's Guide*
- *CICS Adapters Administrator's Guide*
- *Mainframe CORBA Concepts Guide*
- *Mainframe Security Guide*
- Mainframe Management 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 two main parts as follows:

Part 1, "Overview"

Chapter 1, "Introduction"

This chapter introduces the main differences between previous IONA mainframe solutions and Orbix Mainframe 6.2. It also summarizes the main migration impact involved.

Part 2, "Migrating from 2.3.x"

Chapter 3, Installation Requirements

Orbix Mainframe 6.2 is substantially different from Orbix 2.3-based IONA mainframe solutions in terms of the DLLs and build procedures it contains. This chapter outlines the installation requirements for upgrading from an Orbix 2.3.x-based IONA mainframe solution to Orbix Mainframe 6.2.

Chapter 4, "IDL Migration Issues"

This chapter discusses the main IDL differences between an Orbix 2.3-based IONA mainframe solution and Orbix Mainframe 6.2.

Chapter 5, "C++ Migration Issues"

This chapter describes the main issues involved in migrating C++ applications on native z/OS and on z/OS UNIX System Services, from an Orbix 2.3-based IONA mainframe solution to Orbix Mainframe 6.2.

Chapter 6, "COBOL Migration Issues"

This chapter describes the issues involved in migrating COBOL applications from an Orbix 2.3.x-based IONA mainframe solution to Orbix Mainframe 6.2.

Chapter 7, “PL/I Migration Issues”

This chapter describes the issues involved in migrating PL/I applications from an Orbix 2.3.x-based IONA mainframe solution to Orbix Mainframe 6.2.

Chapter 8, “Diagnostic Output”

This chapter summarizes the differences between how diagnostic data is output for Orbix 2.3.x and Orbix 6.2.

Chapter 9, “CORBA Services”

This chapter summarizes the differences in CORBA services between Orbix 2.3.x and Orbix 6.2.

Chapter 10, “Administrative Tools”

This chapter summarizes the differences between Orbix 2.3.x and Orbix 6.2 administration tools.

Chapter 11, “Interoperability”

This chapter describes the issues relating to interoperability when migrating from an Orbix 2.3-based IONA mainframe solution to Orbix Mainframe 6.2.

Part 3, “Migrating from 5.1”**Chapter 12, “Upgrading from Mainframe Edition 5.x”**

This chapter outlines the requirements for upgrading from an Orbix E2A Mainframe Edition 5.x-based solution to Orbix Mainframe 6.2.

Part 4, “Migrating from 6.0”**Chapter 13, “Upgrading from Orbix Mainframe 6.0”**

This chapter outlines the requirements for upgrading from Orbix Mainframe 6.0 to Orbix Mainframe 6.2.

Part 5, Migrating from Artix Mainframe Developer 2.0**Chapter 14, “Upgrading from Artix Mainframe Developer 2.0”**

The Artix Mainframe Developer 2.0 product has been incorporated into Orbix Mainframe 6.2. This chapter outlines the requirements for upgrading from Artix Mainframe Developer 2.0 to Orbix Mainframe 6.2.

Document 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: <pre>#include <stdio.h></pre>
<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: <pre><i>install-dir/etc/domains</i></pre>
Code Bold	Code bold is used to highlight a piece of code within a particular code sample.

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, no prompt is used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
\$	A dollar sign represents the z/OS 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.

PREFACE

Part 1

Overview

In this part

This part contains the following chapters:

Introduction	page 3
------------------------------	------------------------

Introduction

This chapter introduces the main differences between previous IONA mainframe solutions and Orbix Mainframe 6.2. It also provides an overview of the resources available to assist with your migration to Orbix Mainframe 6.2.

In This Chapter

This chapter discusses the following topics:

Advantages of Orbix 6.2	page 4
Migration Resources	page 6

Advantages of Orbix 6.2

Overview

Orbix Mainframe 6.2 is IONA's new product offering for the z/OS environment. This release of Orbix Mainframe offers COBOL and PL/I application support on native z/OS. It also offers C++ application support on native z/OS and z/OS UNIX System Services.

The recommended path for customers upgrading to a new version of Orbix is to move to Orbix 6.2. The extra features offered by Orbix can be divided into the following categories:

- CORBA 2.6-compliant features.
- Unique features.

CORBA 2.6-compliant features

Orbix 6.2 is based on the CORBA 2.6 specification, which standardizes almost every aspect of CORBA programming. Migrating your source code to Orbix 6.2, therefore, represents a valuable investment because your code will be based on a stable, highly standardized programming interface.

Because Orbix 6.2 contains a CORBA 2.6-compliant ORB, it offers the following advantages over Orbix 2.x and Orbix 3.x (that is, all minor versions of Orbix 2 and Orbix 3):

- Portable interceptor support.
- Codeset negotiation support.
- Value type support.
- Asynchronous method invocation (AMI) support.
- Persistent State Service (PSS) support.
- Dynamic any support.

Unique features

Orbix 6.2 also offers some unique benefits over other commercial ORB implementations, including:

- ORB extensibility using IONA's patented Adaptive Runtime Technology (ART).

Orbix 6.2 has a modular structure built on a micro-kernel architecture. Required ORB modules, ORB plug-ins, are specified in a configuration file and loaded at runtime, as the application starts up. The advantage of this approach is that new ORB functionality can be dynamically loaded into an Orbix application without rebuilding the application.

- Improved performance.

The performance of Orbix 6.2 has been optimized, resulting in performance that is faster than Orbix 2.x, Orbix 3.x, and OrbixWeb 3.x in every respect.

Migration Resources

Overview of resources

IONA is committed to assisting you with your migration effort, to ensure that it proceeds as easily and rapidly as possible. The following resources are currently available:

- This migration and upgrade guide.

This technical document provides detailed guidance on converting to Orbix Mainframe 6.2. It aims to provide comprehensive coverage of migration issues, and to demonstrate how features supported in earlier versions can be mapped to Orbix Mainframe 6.2 features.

- Professional Services migration packages.

IONA's Professional Services organization has put together a set of consultancy packages that facilitate rapid migration to Orbix Mainframe 6.2. Details of Professional Services assessment and migration packages are available at:

<http://www.iona.com/info/services/migration.htm>.

Part 2

Migrating from 2.3.x

In this part

This part contains the following chapters:

Migration Possibilities and Main Differences	page 9
Installation Requirements	page 15
IDL Migration Issues	page 19
C++ Migration Issues	page 27
COBOL Migration Issues	page 61
PL/I Migration Issues	page 183
Diagnostic Output	page 247
CORBA Services	page 249
Administrative Tools	page 257
Interoperability	page 261

Migration Possibilities and Main Differences

This chapter introduces the migration possibilities when upgrading from a 2.3.x-based IONA mainframe solution to Orbix 6.2. It also provides an introductory overview of the main migration impact involved for C++, COBOL and PL/I applications.

In This Chapter

This chapter discusses the following topics:

Migration Possibilities	page 10
C++ Applications	page 11
COBOL and PL/I Applications	page 12

Migration Possibilities

Overview

This section summarizes the migration possibilities available.

Summary

The migration possibilities with this release can be summarized as follows:

Table 1: *Migration Possibilities for z/OS*

Migrate From	Migrate To
Orbix 2.3-based C++ on native OS/390 and on OS/390 UNIX System Services.	Orbix Mainframe 6.2 C++ on native z/OS and on z/OS UNIX System Services.
Orbix 2.3-based COBOL on native OS/390.	Orbix Mainframe 6.2 COBOL on native z/OS.
Orbix 2.3-based PL/I on native OS/390.	Orbix Mainframe 6.2 PL/I on native z/OS.

Note: This release of Orbix Mainframe is not binary compatible with the Orbix 2.3.x based product. Therefore, when migrating applications, all IDL must be compiled with the Orbix 6.2 IDL Compiler, the language-specific mappings regenerated, and the applications recompiled and linked.

C++ Applications

In This Section

This section discusses the following topics:

- [BOA replacement](#)
- [The Code Generation Toolkit](#)

BOA replacement

For C++ application programmers, most of the migration issues surround rewriting a server to replace the basic object adapter (BOA) with the portable object adapter (POA). Other issues are more subtle, especially those specific to Orbix, which were used either to work around old deficiencies of the CORBA specification, or to exploit value-added extensions.

The Code Generation Toolkit

The code generation toolkit can be used to develop C++ applications on a platform other than z/OS (for example, Windows or UNIX). Orbix Mainframe does not support use of the code generation toolkit in either native z/OS or UNIX System Services. However, you can use the code generation toolkit off-host, with Orbix on Windows or UNIX, and then copy the generated code to z/OS. Refer to the *CORBA Code Generation Toolkit Guide* for more details.

COBOL and PL/I Applications

In This Section

This section discusses the following topics:

- [The `gencb1` and `genpli` Utilities](#)
 - [Working Storage and Temporary Storage Labels](#)
 - [Generated Data Names](#)
 - [Orbix 6.2 IDL Compiler](#)
-

The `gencb1` and `genpli` Utilities

For COBOL and PL/I application programmers, the biggest difference between Orbix 2.3-based IONA mainframe solutions and Orbix Mainframe 6.2 is the way in which you can generate COBOL and PL/I code from IDL definitions. Orbix 2.3-based IONA mainframe solutions provide the `gencb1` and `genpli` utilities, which generate COBOL and PL/I code respectively from IDL registered in the Interface Repository. These utilities are deprecated in Orbix Mainframe 6.2.

Working Storage and Temporary Storage Labels

For COBOL and PL/I applications, no extra code or changes to application logic are required to achieve successful migration. All required changes to existing COBOL or PL/I code involve updating the source Working Storage labels generated by `gencb1` or the source Temporary Storage labels generated by `genpli`, to reflect the new labels generated by the Orbix 6.2 IDL Compiler.

Generated Data Names

For COBOL and PL/I applications, most migration changes revolve around the differences in the way the deprecated `gencb1` and `genpli` utilities and the Orbix 6.2 IDL Compiler generate data names. Therefore, the Orbix 6.2 IDL Compiler provides a number of arguments that you can use to facilitate integration of your regenerated data names with the legacy code from Orbix 2.3. Refer to the *COBOL Programmer's Guide and Reference* and the *PL/I Programmer's Guide and Reference* for details of these arguments.

Orbix 6.2 IDL Compiler

Orbix Mainframe 6.2 uses the Orbix 6.2 IDL Compiler to generate COBOL and PL/I code from IDL definitions. The Orbix 6.2 IDL Compiler is easier to use than the deprecated utilities. You simply have to run the Orbix 6.2 IDL Compiler with a flag that acts as a plug-in to indicate that you want to generate COBOL or PL/I code. The Orbix 6.2 IDL Compiler does not require an Interface Repository to successfully generate code from IDL.

WARNING: Orbix Mainframe 6.2 supports one set of POA policies. In Orbix Mainframe 6.2, POA names and server names are case sensitive and must therefore match exactly.

Installation Requirements

Orbix Mainframe 6.2 is substantially different from Orbix 2.3-based IONA mainframe solutions in terms of the DLLs and build procedures it contains. This chapter outlines the installation requirements for upgrading from an Orbix 2.3.x-based IONA mainframe solution to Orbix Mainframe 6.2.

In this chapter

This chapter discusses the following topics:

- [“Installing on native z/OS” on page 15.](#)
 - [“Installing on z/OS System Services” on page 16.](#)
 - [“Standard Customization Tasks” on page 16.](#)
 - [“Other Customization Tasks” on page 16.](#)
 - [“Rebuilding Existing Applications” on page 17.](#)
-

Installing on native z/OS

Even though you have already installed a previous version of IONA's mainframe product, you must perform in full the tasks described in the 6.2 version of the *Mainframe Installation Guide* that pertain to installing on z/OS, because of the inherent differences between this and previous versions.

You must perform all these installation tasks in the order in which they are described in the *Mainframe Installation Guide*. Some tasks might not be relevant to your setup, but this is highlighted where appropriate.

Installing on z/OS System Services

If you choose to install Orbix Mainframe 6.2 on z/OS UNIX System Services as well as on native z/OS, you must perform in full the tasks described in the 6.2 version of the *Mainframe Installation Guide* that pertain to installing on z/OS UNIX System Services.

Standard Customization Tasks

After successfully installing Orbix Mainframe 6.2 on z/OS (and on z/OS UNIX System Services if you want), you must perform in full the standard customization tasks described in the 6.2 version of the *Mainframe Installation Guide*.

You must perform all these standard customization tasks in the order in which they are described in the *Mainframe Installation Guide*. Some tasks might not be relevant to your setup, but this is highlighted where appropriate.

Other Customization Tasks

Depending on your setup, there are additional customization tasks that you might also need to perform. These customization tasks relate to:

- Naming Service and Interface Repository customization.
- IMS adapter customization.
- CICS adapter customization.

If you need to perform any of these tasks, you must perform them in the order in which they are described in the *Mainframe Installation Guide*.

PDS names

In Orbix Mainframe 6.2, PDS naming conventions are different from those in Orbix 2.3.x-based solutions. The differences can be summarized as follows:

Table 2: *Differences in PDS Naming Conventions*

2.3.x	6.2
COBOL	CBL
JCL	JCLLIB
LIB	OBJLIB
LOAD	LOADLIB
LPA	LPALIB
PROCS	PROCLIB
RUN	LOADLIB

Rebuilding Existing Applications

If you have built applications using a previous version of IONA's mainframe product, you must:

1. Recompile the IDL pertaining to these applications.

Note: See the relevant programmer's guide for the language you are using for details of how to use the Orbix 6.2 IDL compiler.

2. Check the rest of this guide for details of specific code changes that you might need to make to your applications.
3. Update any JCL that you have stored in non-IONA libraries, to ensure that your applications subsequently compile and link correctly with version 6.2.

Changing your applications and rebuilding them in this way is essential to allow existing applications to function in accordance with the changes inherent in version 6.2.

IDL Migration Issues

This chapter discusses the main IDL differences between an Orbix 2.3-based IONA mainframe solution and Orbix Mainframe 6.2.

In this chapter

This chapter discusses the following topics:

The Opaque Type	page 20
IDL Fixed Type Definitions	page 21
IDL Defined in Fixed Block Data Sets	page 22
imsraw and cicsraw IDL changes	page 23
Orbix 6.2 C++ IDL Compiler Output	page 25

The Opaque Type

Migrating to Orbix 6.2

The object-by-value (OBV) specification, introduced in CORBA 2.3 and supported in Orbix 6.2, replaces opaques.

IDL Fixed Type Definitions

In This Section

This section discusses the following topics:

- [Orbix 6.2](#)
- [Sample IDL](#)
- [In summary](#)

Orbix 6.2

The Orbix 6.2 IDL compiler complies with the CORBA 2.3 specification for IDL fixed type definitions. Each fixed type definition must be specified as a typedef.

Sample IDL

The following IDL illustrates a fixed type definition that is specified as a typedef:

```
//IDL fixed type specified as a typedef
typedef fixed<2,2> t_interest;
attribute t_interest interest;
```

In summary

This issue relates to all languages and all platforms.

IDL Defined in Fixed Block Data Sets

Overview

In the native z/OS environment, all IDL source stored in fixed block data sets must be formatted to adhere to a particular length, because Orbix 6.2 ignores the last eight columns in each record.

This section discusses the following topics:

- [Orbix 6.2](#)
- [Workaround](#)

Orbix 6.2

When Orbix 6.2 accesses fixed block data sets it ignores the last eight columns in each record — which are usually reserved for sequence numbers. For example, if your IDL data set is defined as a fixed block record length 80, the characters after column 72 are ignored.

Note: This is also the case for other Orbix 6.2 fixed block data sets for example configuration files and the license file.

Workaround

If this problem occurs you can do one of the following:

- Move the IDL to variable block data sets.
- Edit the IDL to get around the restriction.

imsraw and cicsraw IDL changes

Overview

This section discusses the impact of changes to `imsraw` and `cicsraw` IDL interfaces used with the IMS and CICS server adapters.

This section discusses the following topics:

- [Details](#)
- [Migration impact](#)

Details

In this release, the `imsraw` and `cicsraw` IDL interfaces have been modified in the following ways:

- The `imsraw` interface is now scoped within a module called `IT_MFA_IMS`.
- The `cicsraw` interface is now scoped within a module called `IT_MFA_CICS`.
- The `do_trans()` operation has been removed from both `imsraw` and `cicsraw`.

Migration impact

If you have existing `imsraw` or `cicsraw` clients that use the unscoped API, these clients can no longer interoperate with the new, scoped `imsraw` and `cicsraw` interface. To avoid the need to modify these existing clients, you can configure the IMS and CICS server adapters as follows, to expose the unscoped version of `imsraw` and `cicsraw`:

```
...
plugins:imsa:imsraw_api_support = "unscoped";
...
plugins:cicsa:cicsraw_api_support = "unscoped";
...
```

Valid values for the preceding configuration variables are:

<code>scoped</code>	Expose only the scoped <code>IT_MFA_IMS::imsraw</code> or <code>IT_MFA_CICS::cicsraw</code> API. This is the default setting.
<code>unscoped</code>	Expose only the unscoped <code>imsraw</code> or <code>cicsraw</code> API.
<code>both</code>	Expose both scoped and unscoped versions of the API.

The associated IDL for both the scoped and unscoped APIs is available in your Orbix installation. On native z/OS it is located in the *orbixhlq.INCLUDE.ORBIX@PD.IDL* PDS. On z/OS UNIX System Services it is located in the *install-dir/asp/6.0/idl/orbix_pdk* subdirectory.

Orbix 6.2 C++ IDL Compiler Output

Overview

Most C++ applications require the IDL compiler to generate both the client stub and server skeleton files. These generated output files have changed slightly in Orbix 6.2, and so too has the way the IDL compiler is invoked. Refer to the *CORBA Programmer's Guide, C++* for more information on how the IDL compiler is invoked.

This subsection discusses the following topics:

- [IDL Compiler Output](#).
 - [Migration Impact](#).
-

IDL Compiler Output

[Table 3](#) summarizes compiler output for both Orbix 6.2 and Orbix 2.3.x for an IDL file called the `grid.idl` in a UNIX System Services environment:

Table 3: C++ Compiler Output Comparison for UNIX System Services

Orbix 6.2	Orbix 2.3.x	File Description
<code>grid.hh</code>	<code>grid.hh</code>	Common header file
<code>gridC.cxx</code>	<code>gridC.cxx</code>	Client stubs
<code>gridS.cxx</code>	<code>gridS.cxx</code>	Server skeletons
<code>gridS.hh</code>		Server header file

Migration Impact

A server's servant implementation in Orbix 6.2 must contain `#include grids.hh`. Also, a server must be linked with `gridS.o` and `gridC.o`. This differs from Orbix 2.3.x where you only had to link with `grid.o`. This is because in Orbix 2.3.x the last line of `gridS.cxx` was always `#include gridC.cxx`.

Existing makefiles need to be updated to take account of any new IDL compiler options, and care must be taken to explicitly include the client stub object file in the server's link line.

Refer to the Orbix 6.2 demonstrations for details on how to upgrade your makefile structure.

C++ Migration Issues

This chapter describes the main issues involved in migrating C++ applications on native z/OS and on z/OS UNIX System Services, from an Orbix 2.3-based IONA mainframe solution to Orbix Mainframe 6.2.

In this Chapter

This chapter discusses the following topics:

C++ Compiler Issues	page 28
C++ Client Migration	page 29
C++ Server Migration	page 42
Migrating Proprietary 2.3 Features	page 51

C++ Compiler Issues

C++ runtime support

Orbix Mainframe 6.2 supports the IBM z/OS V1.4 ANSI C++ compiler, the z/OS 1.5 ANSI C++ compiler, and the z/OS 1.6 ANSI C++ compiler. Because IBM no longer supports the OS/390 V2R10 compiler, IONA has also removed support and certification for this compiler in this release.

Environment targets

Orbix Mainframe 6.2 supplies a C++ options file in `HLQ.ORBIX62.CONFIG(ORXCPP0)` that includes the "TARGET" compiler option of "ZOSV1R2". The "TARGET" compiler option enables the development of applications on higher release levels of z/OS for subsequent use on platforms that are running lower release levels of z/OS or OS/390. You can update the `ORXCPP0` member with "ZOSV1R4", "ZOSV1R5" or "ZOSV1R6", as appropriate. Alternatively, you can remove the "TARGET" option from the `ORXCPP0` member. This means that it uses the "CURRENT" option by default, which corresponds to the z/OS release on which your system is running. For more information see the IBM publication: *C/C++ User's Guide*.

Compiler options

Orbix 6.2 uses the Run Time Type Identification (RTTI) z/OS compiler option. This allows you to take advantage of C++ dynamic casting. In previous releases of Orbix, this option was not used because the OS/390 V2R10 compiler did not support it. Therefore, IONA relied on proprietary code to simulate this functionality. Because the current supported compilers all provide the RTTI option, Orbix 6.2 now uses the C++ dynamic casting (RTTI) option and no longer uses proprietary code.

C++ Client Migration

Overview

This section discusses the following topics:

CORBA Object Location and Binding	page 30
Interface Repository Interoperability	page 35
IDL-to-C++ Mapping	page 36
Client-Side CORBA Compliancy	page 37
Callback Objects	page 39
System Exception Semantics	page 40
Dynamic Invocation Interface (DII)	page 41

CORBA Object Location and Binding

Overview

This subsection summarizes the differences between Orbix 2.3.x object location mechanisms and Orbix 6.2 object location mechanisms. It discusses the following topics:

- [Migration Impact](#)
- [CORBA Naming Service](#)
- [Object-to-string conversion](#)
- [corbaloc URL](#)
- [ORB::resolve_initial_references](#)

Migration Impact

All calls to `_bind()` must be removed and replaced with one of the following object location mechanisms:

- CORBA Naming Service.
- Object-to-string conversion.
- corbaloc URL.
- `ORB::resolve_initial_references()`.

All these alternatives are based on the use of CORBA standard interoperable object references (IORs), the difference being in where the IORs are stored and how they are retrieved by the client application.

CORBA Naming Service

The naming service is the recommended replacement for `_bind()` in most applications. It is easy to understand and use if the application's naming graph is not too complex. Migration to the naming service is straightforward on the client side. The triplet of *markerName*, *serverName*, and *hostName*, used by the `_bind()` function to locate an object, is replaced by a simple name in the naming service.

All applications should use the interoperable Naming Service, which provides access to future Naming Service implementations.

Access to the Naming Service can easily be wrapped. The only potential drawback in using the Naming Service is that it might become a single point of failure or performance bottleneck. If you use the Naming Service only to retrieve initial object references, these problems are unlikely to arise.

When using the naming service, an object's name is an abstraction of the object location and the actual location details are stored in the naming service. Object names are resolved using these steps:

1. An initial reference to the naming service is obtained by calling `resolve_initial_references()` with `NameService` as its argument.
2. The client uses the naming service reference to resolve the names of CORBA objects, receiving object references in return.

Orbix 6.2 supports the CORBA Interoperable Naming Service, which is backward-compatible with the old CORBA Naming Service and adds support for stringified names.

The URL syntax that the Naming Service provides makes it easier to configure IORs—and is similar to `_bind()` by letting you specify host, port, and well known object key in readable format. An example of the syntax for both types is outlined as follows:

- Stringified IOR syntax example:

```
"IOR:004301EF100..."
```

- URL type IOR syntax example:

```
"corbaloc::1.2@myhost:3075/NamingService"
```

With the URL syntax, `corbaloc` is the protocol name, the IIOP version number is `1.2`, the host name is `myhost`, and the port number is `3075`.

Note: If you are using the URL type IOR syntax, Orbix 6.2 requires you to register the stringified IOR against a well known key with the Orbix 6.2 locator daemon. This centralizes the use of stringified IORs in a single place, and lets you widely distribute readable URLs for clients.

Object-to-string conversion

CORBA offers two CORBA-compliant conversion functions:

```
CORBA::ORB::object_to_string()
CORBA::ORB::string_to_object()
```

These functions can replace `_bind()`, because they allow a client to create an IOR with information that is similar to `_bind()`. The Orbix 6.2 locator daemon redirects the IOR, so it avoids the drawbacks of `_bind()`.

The `object_to_string()` and `string_to_object()` functions allow you to convert an object reference to and from the stringified interoperable object reference (stringified IOR) format. These functions enable a CORBA object to be located as follows:

1. A server generates a stringified IOR by calling `CORBA::ORB::object_to_string()`.
2. The server passes the stringified IOR to the client (for example, by writing the string to a file).
3. The client reads the stringified IOR from the file and converts it back to an object reference, using `CORBA::ORB::string_to_object()`.

Orbix 6.2 uses a sequence of octets to compose an object's ID. Orbix 2.3.x uses string markers. CORBA provides helper methods called `string_to_ObjectId()` and `ObjectId_to_string()` to convert between the two types, so migration from marker dependencies to Object IDs should be straightforward.

Because they are not scalable, the `object_to_string()` and `string_to_object()` functions are generally not useful in a large-scale CORBA system. Use them only to build initial prototypes or proof-of-concept applications.

corbaloc URL

A corbaloc URL is a form of human-readable stringified object reference. If you are migrating your clients to Orbix 6.2 but leaving your servers as Orbix 2.3 applications, the corbaloc URL offers a convenient replacement for `_bind()`.

To access an object in an Orbix 2.3 server from an Orbix 6.2 client, using a corbaloc URL, perform the following steps:

1. Obtain the object key, *ObjectKey*, for the object in question, as follows:
 - i. Get the Orbix 2.3 server to print out the stringified IOR using, for example, the `CORBA::ORB::object_to_string()` operation. The result is a string of the form `IOR:00...`
 - ii. Use the Orbix 6.2 `ior2key` utility to parse the stringified IOR. Copy the string that represents the object key field, *ObjectKey*.
2. Construct a corbaloc URL of the following form (where *DaemonHost* and *DaemonPort* are the Orbix daemon's host and port respectively):

```
corbaloc:iiop:1.0@DaemonHost:DaemonPort/ObjectKey%00
```

A null character, %00, is appended to the end of the *ObjectKey* string, because Orbix 2.3 applications expect object key strings to be terminated by a null character.

3. In the source code of the Orbix 6.2 client, use the `CORBA::ORB::string_to_object()` operation to convert the corbaloc URL to an object reference.

The general form of a corbaloc URL for this case is as follows:

```
corbaloc:iiop:GIOPVersion@Host:Port/Orbix3ObjectKey%00
```

In the preceding example, the components of the corbaloc URL are as follows:

- *GIOPVersion*—The maximum GIOP version acceptable to the server. Can be either 1.0 or 1.1.
- *Host* and *Port*—The daemon's (or server's) host and port. The *Host* can either be a DNS host name or an IP address in dotted decimal format.

The *Orbix3ObjectKey* takes the following general form:

```
:\Host:SvrName:Marker::IFRSvrName:InterfaceName%00
```

In the preceding example, the components of the Orbix 3 object key are as follows:

- *Host*—The server host. The *Host* can either be a DNS host name or an IP address in dotted decimal format.
- *SvrName*—The server name of the Orbix 2.3 server.
- *Marker*—The CORBA object's marker.
- *IFRSvrName*—Can be either IR or IFR.
- *InterfaceName*—The object's IDL interface name.

Note: Constructing an Orbix 2.3 object key directly based on the preceding format does not always work because some versions of Orbix impose extra restrictions on the object key format. Extracting the object key from the server-generated IOR is a more reliable approach. If you encounter any difficulties with using corbaloc URLs, please contact support@iona.com.

ORB:resolve_initial_references

The `CORBA::ORB::resolve_initial_references()` operation provides a mechanism for obtaining references to basic CORBA objects (for example, the naming service, the interface repository, and so on).

Orbix 6.2 allows the `resolve_initial_references()` mechanism to be extended, so it can use application-specific services along with typical ones such as the Naming Service. For example, to access the `BankApplication` service using `resolve_initial_references()`, simply add the following variable to the Orbix 6.2 configuration:

```
# Orbix 6.2 Configuration File
initial_references:BankApplication:reference =
    "IOR:010347923849..."
```

Use this mechanism sparingly. The OMG defines the intended behavior of `resolve_initial_references()` and the arguments that can be passed to it. A name that you choose now might later be reserved by the OMG. It is generally better to use the naming service to obtain initial object references for application-level objects.

Interface Repository Interoperability

Overview

Significant changes were made to the IDL definition of the Interface Repository (IFR) between CORBA 2.2 and CORBA 2.3. The Orbix 6.2 IFR is written to conform to the CORBA 2.4 specification and it has many advantages over the Orbix 2.3 IFR.

If you have both Orbix 2.3 and Orbix 6.2 applications that use the IFR, it is recommended that you change the Orbix 2.3 applications to use the Orbix 6.2 IFR.

Modifying Orbix 2.3 applications to use the Orbix 6.2 IFR

To change an Orbix 2.3 C++ application to use the Orbix 6.2 IFR, perform the following steps:

1. Take the IDL for the Orbix 6.2 IFR and generate stub code from it, using the Orbix 2.3 IDL compiler.
2. Modify the source code of your Orbix 2.3 application to be consistent with the IDL for the Orbix 6.2 IFR.
3. Link your Orbix 2.3 application with the IFR stub code generated in step 1.

IDL-to-C++ Mapping

Overview

The definition of the IDL-to-C++ mapping has changed little going from Orbix 2.3 to Orbix 6.2 (apart from some extensions to support valuetypes).

Two notable changes are:

- [The CORBA::Any Type](#).
 - [The CORBA::Environment parameter](#).
-

The CORBA::Any Type

In Orbix 6.2, it is not necessary to use the type-unsafe interface to `Any`. Recent revisions to the CORBA specification have filled the gaps in the IDL-to-C++ mapping that made these functions necessary. That is, the following functions are deprecated in Orbix 6.2:

```
// C++
// CORBA::Any Constructor.
Any(
    CORBA::TypeCode_ptr tc,
    void* value,
    CORBA::Boolean release = 0
);
// CORBA::Any::replace() function.
void replace(
    CORBA::TypeCode_ptr,
    void* value,
    CORBA::Boolean release = 0
);
```

The CORBA::Environment parameter

In Orbix 2.3, the signatures of IDL calls contain the `CORBA::Environment` parameter. In Orbix 6.2, the signatures of IDL calls do not contain the `CORBA::Environment` parameter.

You must therefore remove `CORBA::Environment` parameters from servant implementation classes. The `CORBA::Environment` parameter is needed for compilers that do not support native C++ exception handling, and as a hook for some Orbix proprietary mechanisms.

Client-Side CORBA Compliancy

Overview

Orbix 6.2 enforces strict compliance with the CORBA 2.3 specification. This sub-section describes the main client-side CORBA compliancy issues that should be encountered. It discusses the following topics:

- [Processing Requests](#)
 - [Clean Shutdown](#)
 - [Global Objects](#)
 - [CORBA::Orbix Object Support](#)
 - [Incorrect Raising of INV_OBJREF](#)
 - [Incorrect Raising of COMM_FAILURE](#)
-

Processing Requests

Call `CORBA::ORB_init()` before processing any requests.

Clean Shutdown

Call `CORBA::ORB::shutdown(1)` and `CORBA::ORB::destroy()` before the end of `main()` to ensure clean shutdown and to prevent resource leaks.

Global Objects

The global objects in Orbix 2.3.x means that all ORB initialization is completed before `main()` is entered. Orbix 6.2 requires you to initialize the ORB explicitly in your client and server mainlines.

CORBA::Orbix Object Support

The `CORBA::Orbix` object is not supported in Orbix 6.2. Because this object is unavailable, you must convert Orbix 6.2 client code that uses this convention to call methods on either `CORBA::ORB` or `PortableServer`.

Incorrect Raising of INV_OBJREF

The `INV_OBJREF` exception means that an object reference is corrupt or so malformed that an ORB cannot locate it, or even its server. Customers who use `INV_OBJREF` to remove proxy objects from memory must now use `OBJECT_NOT_EXIST`.

An Orbix 6.2 application must raise the `OBJECT_NOT_EXIST` exception, to indicate that an object does not exist after the client has successfully contacted the server.

Incorrect Raising of COMM_FAILURE

CORBA specifies to throw a `COMM_FAILURE` exception only when a network error occurs after a request is made, but before the reply is received. Orbix 6.2 throws the `TRANSIENT` exception when a connection to the server cannot be established. The `TRANSIENT` exception indicates that an object reference is currently unusable but might work later. This distinction is important to applications that catch `COMM_FAILURE` explicitly to implement connection retries.

Callback Objects

Overview

Callback objects must be contained in a POA like any other CORBA object. This subsection discusses the following topics:

- [POA Policies for Callback Objects](#)
 - [Multi-Threaded Clients](#)
-

POA Policies for Callback Objects

[Table 4](#) shows the most sensible POA policies for a POA that manages callback objects.

Table 4: *POA Policy Types and Their Values for Callback Objects*

Policy Type	Policy Value
Lifespan Policy	TRANSIENT
ID Assignment Policy	SYSTEM_ID
Servant Retention Policy	RETAIN
Request Processing Policy	USE_ACTIVE_OBJECT_MAP_ONLY

Note: By choosing a `TRANSIENT` lifespan policy, you remove the need to register the client with an Orbix 6.2 locator daemon.

These policies allow for easy management of callback objects and a straightforward upgrade path.

Multi-Threaded Clients

Callback objects offer one of the few cases where the root POA has reasonable policies, provided the client is multi-threaded (as it normally is for callbacks) to support callbacks efficiently.

System Exception Semantics

Overview

Orbix 2.3.x clients that catch specific system exceptions might need to change the exceptions they handle when they are migrated to Orbix 6.2.

System exceptions

Orbix 6.2 follows the latest CORBA standards for exception semantics. Table 2 shows the two system exceptions most likely to affect existing code.

Table 5: *Migrated System Exceptions*

When This Happens	Orbix 2.3.x Raise	Orbix 6.2 Raise
Server object does not exist	INV_OBJREF	OBJECT_NOT_EXIST
Cannot connect to server	COMM_FAILURE	TRANSIENT

Minor codes

System exception minor codes are completely different between Orbix 2.3.x and Orbix 6.2. Applications that examine minor codes need to be modified to use Orbix 6.2 minor codes.

Dynamic Invocation Interface (DII)

Overview

This subsection summarizes the differences in availability of DII methods between Orbix 2.3.x and Orbix 6.2. It discusses the following topics:

- [Orbix 2.3.x DIIs](#)
 - [Orbix 6.2 DIIs](#)
 - [Migration Impact](#)
-

Orbix 2.3.x DIIs

Orbix-specific DII methods are available in Orbix 2.3.x.

Orbix 6.2 DIIs

Orbix-specific DII methods are not available in Orbix 6.2. Stub code generated by Orbix 6.2 consists of sets of statically generated CORBA-compliant DII calls.

Migration Impact

Code that uses `CORBA::Request::operator<<()` methods and overloads must be changed to use CORBA-compliant DII methods.

C++ Server Migration

Overview

Server code typically requires many more changes than client code. It is relatively easy to migrate a BOA-based server to a POA-based server by putting all objects in a simple POA that uses an active object map. However, this approach is unable to exploit most of the functionality that a POA-based server offers. It is worthwhile redesigning and rewriting servers so they benefit fully from POA functionality.

In this Section

This section discusses the following topics:

BOA to POA Migration	page 43
Activation Modes	page 45
Object/Servant Lifecycles	page 46
Creating Object References Without Servants	page 48
Function Signatures	page 49
Exception-Safe Servant Implementations	page 50

BOA to POA Migration

Overview

Migrating an Orbix 2.3.x server largely consists of removing BOA-specific code and replacing it with POA functionality. This subsection describes the issues that you must consider. It discusses the following topics:

- [Writing POA-based Code](#)
 - [Choosing POA Policies](#)
 - [Object IDs versus Markers](#)
 - [Migrating Orbix Loaders](#)
 - [Servant Locators](#)
 - [Overriding the Default POA](#)
-

Writing POA-based Code

Several resources and strategies are available for learning how to write efficient POA-based code:

- Enroll in an Orbix 6.2 training course.
- Read Henning/Vinoski's *Advanced CORBA Programming with C++*.
- Examine the demonstrations that are provided with your Orbix 6.2 installation.
- Use the Orbix 6.2 code generation toolkit to generate test clients and automate the more routine aspects of server programming.

Note: Orbix Mainframe does not support use of the code generation toolkit in either native z/OS or UNIX System Services. However, you can use the code generation toolkit off-host, with Orbix on Windows or UNIX, and then copy the generated code to z/OS.

Choosing POA Policies

A POA that uses a servant manager, and especially a servant locator, can assert great control over object life cycles. A POA can also implement a default servant, which can simulate almost unlimited numbers of objects. IONA's Orbix 6.2 training course contains much advice, including a decision flowchart on how to choose POA policies.

Object IDs versus Markers

Orbix 6.2 uses a sequence of octets to compose an object's ID. Orbix 2.3.x uses string markers. CORBA provides helper methods `string_to_ObjectId()` and `ObjectId_to_string()` to convert between the two types, so migration from marker dependencies to Object IDs should be straightforward.

Migrating Orbix Loaders

Orbix loader architecture is constrained by BOA limitations. The BOA always maintains an object map internally. This can lead to duplicated efforts and synchronization concerns, if you try to maintain your own object map for caching and eviction.

Servant Locators

A servant locator gives you full control over servant creation and routing of CORBA requests to the appropriate servants. Servant locators also help you avoid thread-related blockages.

Overriding the Default POA

The issues that surround implicit activation of objects in an unexpected POA require careful consideration by anyone who works with Orbix 6.2. Orbix 6.2 genies offer several options to override `_default_POA()` that your own code can emulate.

Activation Modes

In This Section

This subsection describes migration issues relating to activation modes. It discusses the following topics:

- [BOA Activation Modes](#)
 - [POA Shared Modes](#)
 - [Migration Impact](#)
 - [Orbix 6.2 Enterprise Edition](#)
-

BOA Activation Modes

BOA activation modes—Shared, Unshared, Per-method and Persistent—are used for a variety of reasons: to achieve multi-threaded behavior in a single-threaded environment, to increase server reliability, and so on. All Orbix 2.3.x activation modes, except Shared, are typically used to achieve some form of load balancing that is transparent to the client. The two most popular modes are Shared and the Orbix-specific mode, Per-Client-Pid:

- Shared mode — enables all clients to communicate with the same server implementation.
 - Per-Client-Pid mode — enforces a one-to-one relationship between the client process and server process, and is sometimes used to maximize server availability.
-

POA Shared Modes

The POA provides three shared activation modes:

- always
 - on-demand
 - never
-

Migration Impact

The choice of activation mode has almost no impact on BOA-based or POA-based server code, so the migration path should be straightforward.

Orbix 6.2 Enterprise Edition

The Enterprise Edition of Orbix 6.2 includes transparent locator-based load balancing over a group of replica POAs. This should answer the needs currently addressed by most Orbix 2.3.x activation modes.

Object/Servant Lifecycles

Overview

This subsection summarizes the differences in object reference creation between BOAs and POAs. It discusses the following topics:

- [Creating Object References with POAs](#)
 - [BOA-based Implementation](#)
 - [POA-based Implementation](#)
 - [Migration Impact](#)
-

Creating Object References with POAs

Because the POA separates CORBA objects from servants, it offers markedly different approaches to the creation of object references. For example, the following IDL provides a factory object, `openNewAccount()`, for creating `Account` objects:

```
interface Account {...}
interface Bank {
    Account openNewAccount(in string owner);
};
```

BOA-based Implementation

A typical C++ BOA-based implementation of the `Bank::openNewAccount()` method looks like this:

```
Account_ptr Bank_i::openNewAccount(const char* owner)
{
    Account_i* newAccImpl = new Account_i(owner);
    StoreWithAllTheOtherAccounts(newAccImpl);
    return Account::_duplicate(newAccImpl);
}
```

POA-based Implementation

A POA-based implementation is slightly, but significantly, different:

```
Account_ptr Bank_i::openNewAccount(const char* owner)
{
    Account_i* newAccImpl = new Account_i(owner);
    StoreWithAllTheOtherAccounts(newAccImpl);
    return newAccImpl->_this();
}
```

Migration Impact

You do not need to manage the object reference. It is returned to the client and forgotten until a client makes an invocation on it. The server then determines which servant processes the request. You can delegate this work to the POA, or you use a servant manager to do it yourself.

Creating Object References Without Servants

Overview

This subsection summarizes the differences in the way that BOAs and POAs associate object references with servants. It discusses the following topics:

- [BOA-Based Servers](#)
 - [POA-Based Servers](#)
 - [Scalability of POA-Based Servers](#)
 - [Migration Impact](#)
-

BOA-Based Servers

In BOA-based servers, the `tie` approach helps to separate a CORBA object from its servant. Because the POA enforces this separation, there is usually no reason to use the `tie` approach. It is useful only on the rare occasion where a servant cannot inherit from third party classes, as mandated by some object-oriented databases. In general, the `tie` approach adds an extra layer of unnecessary functionality.

POA-Based Servers

A POA-based server lets you create CORBA object references without creating their servant implementations. When created you can send these references around your CORBA system and deal with processing invocations on them at a later stage.

Scalability of POA-Based Servers

Creating CORBA object references without creating their servant implementations lends itself to very scalable solutions. For example, you can distribute all `Account` object references in a CORBA system and use a default servant to process all the invocations on them, rather than implement a unique servant for each object. This is logical as there typically might be only several invocations on a given `Account` object each week.

Migration Impact

You do not need to manage object references. An object reference is returned to the client and forgotten until a client makes an invocation on it. The server then determines which servant processes the request. You can delegate this work to the POA, or you can use a servant manager to do it yourself.

Function Signatures

Changes to the signature

In Orbix 6.2, two significant changes have been made to C++ function signatures:

- The `CORBA::Environment` parameter has been dropped.
- New types are used for `out` parameters. An `out` parameter of `T` type is now passed as a `T_out` type.

Consequently, when migrating C++ implementation classes you must replace the function signatures that represent IDL operations and attributes.

Exception-Safe Servant Implementations

Overview

This subsection describes migration issues relating to the `_var` type. It discusses the following topics:

- [CORBA 2.1 and Behavior of the `_var` Type](#).
 - [Exception-Safe Use of `_var` Type](#).
-

CORBA 2.1 and Behavior of the `_var` Type

The CORBA 2.1 specifications and earlier versions failed to consider the behavior of the `_var` type during a servant method implementation that might require the `_var` to give up the memory that it owns (usually under exceptional circumstances).

Exception-Safe Use of `_var` Type

The CORBA 2.2 specification improved the C++ mapping by introducing the `_retn()` method on `_var` classes. This method ensures exception-safe usage of `_var` types and allows the `_var` to properly relinquish ownership of its data.

For example:

```
// C++
char* FooImpl::get_string() throw(CORBA::SystemException) {
    CORBA::String_var result = CORBA::string_dup("foo");
    // Now do something that might throw a SystemException,
    // for instance, make another CORBA call.
    // This is safe since result is a _var and cleans
    // up when it goes out of scope
    return result._retn(); // Give up ownership to return
}
```

Migrating Proprietary 2.3 Features

Overview

This section discusses the issues that relate to migrating proprietary Orbix 2.3.x features to Orbix 6.2.

In this section

This section discusses the following topics:

Orbix Filters and CORBA 2.3 Alternatives	page 52
Transformers	page 56
Orbix-Specific APIs	page 57
Connection Management	page 58
Callbacks and Bidirectional GIOP	page 60

Orbix Filters and CORBA 2.3 Alternatives

Overview

This subsection summarizes, from the point of view of their purpose, the CORBA 2.3 alternatives in Orbix 6.2 to Orbix filters. It discusses the following topics:

- [Orbix Filter Functions](#)
 - [Request Logging](#)
 - [Accessing a Client's TCP/IP Information](#)
 - [Piggybacking Extra Data](#)
 - [Multi-Threaded Request Processing](#)
 - [Thread Pools](#)
 - [Thread Pool Configuration Settings](#)
 - [WorkQueue Policies](#)
-

Orbix Filter Functions

Orbix proprietary filter mechanisms serve many purposes. These include:

- Request logging.
- Accessing the client's TCP/IP information using `Request::descriptor()`.
- Piggybacking extra data.
- Security using an `AuthenticationFilter`.
- Multi-threading using a `ThreadFilter`.

The following sections discuss Orbix 6.2 alternatives.

Request Logging

To achieve request logging capabilities, use `PortableInterceptor` interfaces to obtain access to a CORBA request at any stage of the marshalling process. These interfaces offer much more than Orbix filters. You can use them to add and examine service contexts. You can also use them to examine the actual arguments to the request.

Note: The `PortableInterceptor` draft specification is still undergoing review and might be subject to changes before final ratification.

Accessing a Client's TCP/IP Information

Some clients use Orbix-specific extensions to access socket-level information, such as the caller's IP address, to implement proprietary security features. Methods such as `CORBA::Request::descriptor()`, however, are not available in Orbix 6.2, so alternatives must be found.

To provide security for your applications, it is recommended that you use an implementation of the security service provided with the Orbix 6.2 Enterprise Edition off-host instead. See the *Mainframe Security Guide* for more details.

Note: File descriptors are not exposed, because Orbix 6.2 transparently supports protocols such as shared memory or multicast, which do not necessarily have a concept of a file descriptor. Exposing a file descriptor breaks this transparency and greatly constrains the flexibility of the ORB and the application.

Security using an authentication filter

Some Orbix 2.3.x applications use authentication filters to implement security features. In Orbix 6.2, it is recommended that you use the security service that is made available with the Orbix 6.2 Enterprise Edition off-host. See the *Mainframe Security Guide* for more details.

Piggybacking Extra Data

Piggybacking is a feature in Orbix 2.3.x that enables you to add and remove extra arguments to a request message. Piggybacking extra data from client to server should be changed to the CORBA 2.3-compliant approach of using `ServiceContexts`.

Multi-Threaded Request Processing

Orbix 2.3.x supports the Orbix `ThreadFilters` mechanism, which offers multi-threading capabilities.

In Orbix 6.2, request processing conforms to the CORBA 2.4 specification. Each POA can have its own threading policy:

- `SINGLE_THREAD_MODEL` ensures that all servant objects in that POA have their functions called in a serial manner. In Orbix 6.2, servant code is called only by the main thread; therefore, no locking or concurrency-protection mechanisms need to be used.
- `ORB_CTRL_MODEL` leaves the ORB free to dispatch CORBA invocations to servants in any order and from any thread it chooses.

Because the CORBA 2.4 specification does not specify exactly what happens when the `ORB_CTRL_MODEL` policy is chosen, Orbix 6.2 makes some proprietary extensions to the threading model.

The multi-threaded processing of requests is controlled using the Orbix 6.2 work queue feature. Two kinds of work queue are provided by Orbix 6.2:

- *Automatic Work Queue*: A work queue that feeds a thread pool. When a POA uses an automatic work queue, request events are automatically dequeued and processed by threads. The size of the thread pool is configurable.
- *Manual Work Queue*: A work queue that requires the developer to explicitly dequeue and process events.

Manual work queues give developers greater flexibility when it comes to multi-threaded request processing. For example, prioritized processing of requests could be implemented by assigning high-priority CORBA objects to one POA instance and low-priority CORBA objects to a second POA instance. Given that both POAs are associated with manual work queues, the developer can write threading code that preferentially processes requests from the high-priority POA.

Thread Pools

Thread pools are created and controlled through the ORB configuration. All POAs with a policy of `ORB_CTRL_MODEL` share a thread pool within the ORB. By default, the thread pool starts with five threads, and adds new threads when the number of outstanding requests exceeds the number of threads. By default, there is no limit to the maximum number of threads.

Thread Pool Configuration Settings

The configuration settings for the thread pool are:

- `thread_pool:high_water_mark`
- `thread_pool:low_water_mark`
- `thread_pool:initial_threads`
- `thread_pool:max_queue_size`

These settings can be controlled through the Orbix 6.2 configuration.

WorkQueue Policies

Orbix 6.2 also provides a proprietary `WorkQueue` policy, which you can associate with a POA and thereby control the flow of incoming requests for that POA. You can implement your own `WorkQueue` interface, or use IONA-supplied `WorkQueue` factories to create one of two `WorkQueue` types:

- A `ManualWorkQueue`, which requires the developer to explicitly dequeue and process events.
- An `AutomaticWorkQueue`, which feeds a thread pool.

When a POA uses an `AutomaticWorkQueue`, request events are automatically dequeued and processed by threads. Use one of the preceding thread pool configuration settings listed to configure the size of the thread pool.

Transformers

Orbix 2.3.x

Transformers are a deprecated feature of Orbix 2.3.x that allow you to apply customized encryption to CORBA request messages. This could be used to implement a primitive substitute for a security service.

Orbix 6.2

In Orbix 6.2, transformers are not supported. It is recommended, instead, that you use the security service that is made available with the enterprise edition of Orbix 6.2. See the *Mainframe Security Guide* for more details.

Orbix-Specific APIs

In This Section

This subsection describes migration issues relating to Orbix-specific APIs. It discusses the following topics:

- [Availability of ORB Classes in Orbix 2.3.x.](#)
- [Availability of ORB Classes in Orbix 6.2.](#)
- [Migration Impact.](#)

Availability of ORB Classes in Orbix 2.3.x

The Orbix ORB class has many proprietary configuration Application Programming Interfaces (APIs) and extensions, such as `maxConnectRetries()` and `bindUsingIIOP()`.

Availability of ORB Classes in Orbix 6.2

Proprietary Orbix ORB class APIs are not available in the Orbix 6.2 ORB class.

Migration Impact

In general, these calls are no longer necessary, or their functionality is available through configuration.

Connection Management

Overview

Orbix 6.2 provides an active connection manager that allows the ORB to reclaim connections automatically, and thereby increases the number of clients that can concurrently use a server beyond the limit of available file descriptors.

This subsection discusses the following topics:

- [IIOp Configuration Variables](#)
- [ORBs and IIOp Connections](#)
- [File Descriptor Limits](#)
- [File Descriptor Limits and Orbix 6.2](#)
- [TCP/IP Socket-Level Access](#)

IIOp Configuration Variables

IIOp connection management is controlled by four configuration variables:

- `plugins:iio:incoming_connections:hard_limit` sets the maximum number of incoming (server-side) connections allowed to IIOp. IIOp refuses new connections above this limit.
- `plugins:iio:incoming_connections:soft_limit` determines when IIOp starts to close incoming connections.
- `plugins:iio:outgoing_connections:hard_limit` sets the maximum number of outgoing (client-side) connections allowed to IIOp. IIOp refuses new outgoing connections above this limit.
- `plugins:iio:outgoing_connections:soft_limit` determines when IIOp starts to close outgoing connections.

ORBs and IIOp Connections

The ORB first tries to close idle connections in least-recently-used order. If there are no idle connections, the ORB closes busy connections in least-recently-opened order.

File Descriptor Limits

Active connection management effectively remedies file descriptor limits that have constrained previous Orbix applications. If a client is idle for a period of time and the server ORB reaches its connection limit, it sends a

GIOP `CloseConnection` message to the client and closes the connection. Later, the same client can transparently re-establish its connection, to send a request without throwing a CORBA exception.

File Descriptor Limits and Orbix 6.2

Orbix 6.2 is configured to use the largest upper file descriptor on each supported operating system (OS). On a UNIX OS it is possible to rebuild the OS kernel to obtain a larger number. However, active connection management should make this unnecessary.

File descriptors are not exposed, because Orbix 6.2 transparently supports protocols such as shared memory or multicast, which do not necessarily have a concept of a file descriptor. Exposing a file descriptor breaks this transparency and greatly constrains the flexibility of the ORB and the application.

Note: Orbix 2.3.x throws a `COMM_FAILURE` exception on the first attempt at re-connection; server code that anticipates this exception should be reevaluated against Orbix 6.2 functionality.

TCP/IP Socket-Level Access

Orbix 6.2 does not allow access to TCP/IP sockets or transport-level information, nor does it mandate a TCP/IP transport layer. You can specify a transport plug-in such as multicast, (which is connectionless), SOAP, HTTP, ATM, and so on. The shared memory transport (SIOP), for example, does not use file descriptors or sockets. Because Orbix 6.2 has no equivalent to the Orbix `IOCallback` functionality, you must migrate any code that uses it.

Callbacks and Bidirectional GIOP

Overview

Orbix 6.2 introduces support for bidirectional GIOP, based on an OMG standard. This is a new feature introduced since Orbix E2A ASP v 6.0. Previously (in Orbix E2A ASP v5.x and v6.0), bidirectional GIOP was not supported, or was not based on an OMG standard (Orbix 3.x and earlier).

Motivation for bi-directional IIOP

Bidirectional GIOP was introduced in Orbix in order to overcome the limitations of standard GIOP in relation to using callback objects through a firewall.

Features

IONA's implementation of bidirectional GIOP has the following features:

1. Compliant with the modified bidirectional GIOP approach described in the firewall submission.
 2. Compatible with GIOP 1.2 (that is, not dependent on GIOP 1.4 `NegotiateSession` messages).
 3. Decoupled from IIOP, so that it can be used over arbitrary connection-oriented transports (for example, SHMIOP).
 4. Supports weak `BiDirIds` initially.
 5. Supports bidirectional invocations on legacy Orbix 2.3.x callback object references in order to facilitate phased migration to Orbix 6.2.
-

References

For more details about the bidirectional GIOP support in Orbix 6.2, see the following references:

- *CORBA Programmer's Guide*
- *Administrator's Guide*

COBOL Migration Issues

This chapter describes the issues involved in migrating COBOL applications from an Orbix 2.3-based IONA mainframe solution to Orbix Mainframe 6.2.

In this Chapter

This chapter discusses the following topics:

Name Mapping Issues	page 63
Copybook Names Based on IDL Member Name	page 89
Name Scoping and the COBOL Compilers	page 97
Typecode Name and Length Identifiers	page 107
Reserved COBOL and OMG Keywords	page 116
Error Checking and Exceptions	page 122
Nested Unions in IDL	page 126
Mapping for Arrays	page 131
Working Storage data Items and Group Moves	page 133
Mapping for IDL type Any	page 135
CORBA Copybook Additions	page 137

Parameter Passing of Object References in IDL Operations	page 138
CORBA Object Location and Binding	page 139
API Migration Issues	page 145
COBOL IMS Server Migration Issues	page 151
COBOL IMS Client Migration Issues	page 165
COBOL CICS Server Migration Issues	page 171
COBOL CICS Client Migration Issues	page 178
Miscellaneous	page 181

Name Mapping Issues

In This Section

This section discusses the following topics:

Fully Qualified Level 01 Data Names	page 64
Operation and Level 88 Data Names	page 68
IDL Constant Definitions Mapped to Fully Qualified Names	page 72
Derived Interface Names and Fully Qualified Names	page 77
Numeric Suffixes for Data Names	page 80
160-Character Limit for String Literals	page 81
Maximum Length of COBOL Data Names	page 86

Fully Qualified Level 01 Data Names

Overview

This subsection summarizes the differences in the way that `gencb1` and the Orbix 6.2 Compiler generate level 01 data names. It discusses the following topics:

- [The `gencb1` Utility](#)
- [Orbix 6.2 IDL Compiler](#)
- [Sample IDL](#)
- [The `gencb1` Utility Output](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)
- [Example of Using the `-M` Argument](#)
- [In Summary](#)

The `gencb1` Utility

The `gencb1` utility uses only the interface name as a prefix for generated data names. The `gencb1` utility can only support interfaces that are defined within a single module. It can therefore not support multiple levels of nested modules and interfaces.

Orbix 6.2 IDL Compiler

The Orbix 6.2 IDL Compiler replaces the `gencb1` utility. The Orbix 6.2 IDL Compiler generates fully qualified names for COBOL 01 level data items. This means that it includes both module and interface names in COBOL data names. It can therefore support any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces).

The ability of the Orbix 6.2 IDL Compiler to generate fully qualified names ensures the uniqueness of each generated name when, for example, the same operation name or attribute is used at a different scope within an IDL member.

Sample IDL

Consider the following IDL sample called the `AMODULE` member:

```
module Mymod
{
  interface myinter
  {
    void myop(inout long mylong);
  };
};
```

The gencl Utility Output

The `gencl` utility outputs the following for the preceding IDL sample:

```
01 MYINTER-MYOP-ARGS.
   03 MYLONG                                PICTURE S9(09) BINARY.
```

The module name is omitted from the 01 level data name.

Orbix 6.2 IDL Compiler Output

Orbix 6.2 IDL Compiler outputs the following for the preceding IDL:

```
01 MYMOD-MYINTER-MYOP-ARGS.
   03 MYLONG                                PICTURE S9(10) BINARY.
```

The Orbix 6.2 IDL Compiler includes `Mymod` in the 01 level data name

Migration Impact

Use the `-M` argument that is provided with the Orbix 6.2 IDL Compiler to avoid having to make changes to your application source code. The `-M` argument allows you to generate a mapping member that you can then use to map alternative names to your fully qualified data names. You can set these alternative names in the mapping member to be the same as the COBOL data names that were originally generated by `gencl`.

You must run the Orbix 6.2 IDL Compiler twice, first with the `-McreateN` and then the `-Mprocess` argument. The first run generates the mapping member, complete with the fully qualified names and the alternative name mappings. The alternative name mappings generated are dependent on the argument given to the `-McreateN` where *N* can have an integer value of either 0, 1, or 2. At this point you can manually edit the mapping member (if necessary) to change the alternative names to the names you want to use.

Then run the `-Mprocess` argument again, this time to generate your COBOL copybooks complete with the alternative data names in the specified mapping member.

Refer to the *COBOL Programmer's Guide and Reference* for an example of how to use the `-M` argument.

Example of Using the -M Argument

The `-M` argument can be used to make the Orbix 6.2 compiler output the same as the `genctl` output for the preceding IDL. The steps to do this are as follows:

Step	Action
1	<p>Create a mapping member for the IDL by running the mapping member as follows:</p> <pre data-bbox="582 701 1177 961"> //IDLCBL EXEC ORXIDL, // SOURCE=AMODULE, // IDL=&ORBIX..DEMOS.IDL, // COPYLIB=&ORBIX..DEMOS.COBOL.COPYLIB, // IMPL=&ORBIX..DEMOS.COBOL.SRC, // IDLPARAM=' -cobol:-Mcreate1MYMAP ' //IDLMAP DD DISP=SHR,DSN=&ORBIX..DEMOS.COBOL.MAP </pre> <p>This produces the following in the mapping member:</p> <pre data-bbox="582 1025 943 1124"> Mymod Mymod Mymod/myinter myinter Mymod/myinter/myop myinter-myop </pre>

Step	Action
2	<p>Using the mapping member in step 1 and run the IDL compiler again as follows:</p> <pre data-bbox="617 361 1209 626"> //IDLCBL EXEC ORXIDL, // SOURCE=AMODULE, // IDL=&ORBIX..DEMOS.IDL, // COPYLIB=&ORBIX..DEMOS.COBOL.COPYLIB, // IMPL=&ORBIX..DEMOS.COBOL.SRC, // IDLPARAM='-cobol:-MprocessMYMAP' //IDLMAP DD DISP=SHR,DSN=&ORBIX..DEMOS.COBOL.MAP </pre> <p>This produces output which is the same as that generated by <code>gencl</code> for this operation section:</p> <pre data-bbox="617 713 1275 774"> 01 MYINTER-MYOP-ARGS. 03 MYLONG PICTURE S9(10) BINARY. </pre>

In Summary

Affects both clients and servers. Requires use of the `-M` argument, and if necessary, code changes.

Operation and Level 88 Data Names

Overview

This subsection summarizes the differences in the way that `genctl` and the Orbix 6.2 IDL Compiler generate level 88 and level 01 data names for operations and attributes defined in IDL. It discusses the following topics:

- [The `genctl` approach](#)
 - [Orbix 6.2 IDL Compiler](#)
 - [Migration Impact](#)
 - [Sample IDL](#)
 - [The `genctl` Utility Output](#)
 - [Orbix 6.2 IDL Compiler Output](#)
 - [Example of Using the `-M` Argument](#)
 - [In Summary](#)
-

The `genctl` approach

The `genctl` utility does not use the fully qualified name, instead it uses the interface name only as the first qualifier. You can use the `-M` argument with the Orbix 6.2 IDL Compiler to mimic `genctl` output.

Orbix 6.2 IDL Compiler

Operation identifier names and associated level 88 data names are generated with fully qualified names by default, because of the multiple levels of nesting in IDL members that the Orbix 6.2 IDL Compiler supports. The issue is similar to that discussed in [“Fully Qualified Level 01 Data Names” on page 64](#).

Migration Impact

There is only a migration impact if the IDL contains modules.

Use the `-M` argument that is provided with the Orbix 6.2 IDL Compiler to resolve the migration impact. The `-M` argument can be used to map the fully qualified generated names (based on the IDL member name) to alternative names that match those generated by `genctl`.

Refer to the *COBOL Programmer's Guide and Reference* for an example of how to use the `-M` argument.

Sample IDL

Consider the following IDL, called the `MYMOD` member:

```
module amodule
{
    interface fred
    {
        void myop(in long along,inout short ashort);
    };
};
```

The gencl Utility Output

Based on the preceding IDL, `gencl` outputs the following:

```
01 FRED-OPERATION          PICTURE X(26) .
      88 FRED-MYOP          VALUE "myop:IDL:amodule/fred:1."
01 FRED-OPERATION-LENGTH  PICTURE 9(09)BINARY VALUE 26.
```

Orbix 6.2 IDL Compiler Output

Based on the preceding IDL, the Orbix 6.2 IDL Compiler outputs the following:

```
01 AMODULE-FRED-OPERATION PICTURE X(26) .
      88 AMODULE-FRED-MYOP
      VALUE "myop:IDL:amodule/fred:1.0" .
01 AMODULE-FRED-OPERATION-LENGTH PICTURE 9(09) BINARY
      VALUE 26.
```

Example of Using the -M Argument

The `-M` argument be used can to make the Orbix 6.2 compiler output the same as the `genctl` output for the preceding IDL by following the steps below:

Step	Action
1	<p>Create a mapping member for the IDL by running the mapping member as follows:</p> <pre data-bbox="585 517 1155 777">//IDLCBL EXEC ORXIDL, // SOURCE=MYPMOD, // IDL=&ORBIX..DEMOS.IDL, // COPYLIB=&ORBIX..DEMOS.COBOL.COPYLIB, // IMPL=&ORBIX..DEMOS.COBOL.SRC, // IDLPARAM=' -cobol:-Mcreate1MYMAP1' //IDLMAP DD DISP=SHR,DSN=&ORBIX..DEMOS.COBOL.MAP</pre> <p>This produces the following in the mapping member:</p> <pre data-bbox="585 841 916 942">amodule amodule amodule/fred fred amodule/fred/myop/ fred-myop</pre>

Step	Action
2	<p>Use the mapping member in step 1 and run the IDL compiler again as follows:</p> <pre data-bbox="617 369 1201 631"> //IDL CBL EXEC ORXIDL, // SOURCE=MYMOD, // IDL=&ORBIX..DEMOS.IDL, // COPYLIB=&ORBIX..DEMOS.COBOL.COPYLIB, // IMPL=&ORBIX..DEMOS.COBOL.SRC, // IDLPARAM=' -cobol: -MprocessMYMAP1 ' //IDL MAP DD DISP=SHR,DSN=&ORBIX..DEMOS.COBOL.MAP </pre> <p>This produces output which is the same as that generated by <code>gencl</code> for this operation section:</p> <pre data-bbox="617 725 1271 868"> 01 FRED-OPERATION PICTURE X(26). 88 FRED-MYOP VALUE "myop:IDL:amodule/fred:1.0". 01 FRED-OPERATION-LENGTH PICTURE 9(09) BINARY VALUE 26. </pre>

In Summary

Affects clients and servers. Requires code change or use of the described workaround.

IDL Constant Definitions Mapped to Fully Qualified Names

Overview

This subsection summarizes the differences in the way that `genctl` and the Orbix 6.2 IDL Compiler generate COBOL data names for IDL constant definitions. It discusses the following topics:

- [Mapping for Constants Comparison](#)
- [The `genctl` Utility](#)
- [Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)
- [Sample IDL](#)
- [Orbix 6.2 Generated Data Names](#)
- [Legacy Support](#)
- [In Summary](#)

Mapping for Constants Comparison

The following are the differences between the Orbix 6.2 IDL Compiler and `genctl` mapping for constants:

Table 6: *COBOL Compiler Output for IDL Constant Definitions*

	Orbix 6.2 IDL Compiler	<code>genctl</code> Utility
Global constant at IDL member level	01 GLOBAL- <i>idlmembername</i> -CONSTS 03 <i>localname</i> ...	01 <i>interfacename</i> -GLOBAL-CONSTS 03 <i>interfacename-localaname</i> ...
Global constant at module level	01 <i>FQN</i> -CONSTS 03 <i>localname</i> ...	01 <i>interfacename</i> -MODULE-CONSTS 03 <i>interfacename-localname</i> ...
Constant at interface level	01 <i>FQN</i> -CONSTS 03 <i>localname</i> ...	01 <i>interfacename</i> -CONSTANTS 03 <i>interfacename-localname</i> ...

In the preceding table, *FQN* represents the fully qualified name for the module or interface where the constant is defined.

The `genctl` Utility

The `genctl` utility uses only the interface name to map IDL constant definitions to data names, because it only supports only one level of nesting of modules in IDL.

Orbix 6.2 IDL Compiler

IDL constant definitions are mapped to fully qualified data names in Orbix 6.2, because the Orbix 6.2 IDL Compiler can process any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces). Therefore, the same constant names can be used at different scopes, and uniqueness of data names is imperative.

Migration Impact

The `MODULE` keyword that is generated by `gencb1` is not used in Orbix 6.2, because there is support for more than one level of module. With `gencb1`, only one level of module is supported. .

Note: The `GLOBAL` keyword is still used, but in the case of `gencb1`, refers to all constant definitions defined in the Interface Repository. In the case of Orbix 6.2 it refers to all constants defined at global scope in the IDL member being processed.

Note: The Interface Repository server is not required by the Orbix 6.2 IDL Compiler when generating COBOL definitions from IDL. For further details refer to [“Interface Repository Server” on page 181](#).

Sample IDL

Consider the following IDL member, called `TEST`, which defines four constants with the same name — `myconstant` — at different levels:

```
//test.idl
const long myconstant = 1;
module m1
{
    const long myconstant = 1;
    interface fred
    {
        const long myconstant = 1;
        void myop();
    };
    module m2
    {
        interface fred
        {
            const long myconstant = 1;
            void myop();
        };
    };
};
```

Orbix 6.2 Generated Data Names

Based on the preceding IDL, the Orbix 6.2 IDL Compiler generates the following data names:

```
*****
* Constants in root scope:
*****
01 GLOBAL-TEST-CONSTS.
   03 MYCONSTANT                PICTURE S9(10) BINARY VALUE 1.
*****
* Constants in m1:
*****
01 M1-CONSTS.
   03 MYCONSTANT                PICTURE S9(10) BINARY
                                VALUE 1.
*****
* Constants in m1/fred:
*****
01 M1-FRED-CONSTS.
   03 MYCONSTANT                PICTURE S9(10) BINARY
                                VALUE 1.
*****
* Constants in m1/m2/fred:
*****
01 M1-M2-FRED-CONSTS.
   03 MYCONSTANT                PICTURE S9(10) BINARY
                                VALUE 1.
```

Legacy Support

It is not feasible to provide full legacy support in this case. However, you can use the `-M` argument with the Orbix 6.2 IDL Compiler to control the `FQN` name shown in the preceding example. You can also use the `-o` argument with the Orbix 6.2 IDL Compiler to determine the name of the generated copybook, which defaults to the IDL member name. This only affects the level 01 data name for Global constants; for example, if the `-o` argument is used with the name `TESTS`, that is, `-OTESTS`, the IDL compiler output changes from:

```
01 GLOBAL-TEST-CONSTS.
   03 MYCONSTANT                PICTURE S9(09) BINARY VALUE 1.
```

to:

```
01 GLOBAL-TESTS-CONSTS.
   03 MYCONSTANT                PICTURE S9(09) BINARY VALUE 1.
```

In Summary

Affects clients and servers. Requires code changes where constants are used.

Derived Interface Names and Fully Qualified Names

Overview

This subsection summarizes the differences in the way that version v2r3m5 (or higher) of `gencb1` and the Orbix 6.2 IDL Compiler generate level 88 entries for IDL operation names to process remote derived objects on the client side.

Note: For users of a `gencb1` version earlier than version v2r3m5 no changes are required, because the extra level 88 entry for each operation name (incorporating the fully qualified name) is not included.

This subsection discusses the following topics:

- [Migration Impact](#)
- [Sample IDL](#)
- [Main Copybook Sample for GRID using version v2r3m5 \(or higher\)](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Changes on the Client-Side](#)
- [In Summary](#)

Migration Impact

For users of `gencb1` version v2r3m5 (or higher) which generates a main copybook that includes an extra level 88 entry for each operation name (incorporating the fully qualified name) changes are required.

Applications that use fully qualified data names require changes to use the original name. For the `grid` example this would mean changing `set fq-grid-get-height` to `set grid-get-height`. The Orbix 6.2 IDL Compiler does not generate the fully qualified data name, therefore client code that references these fully qualified names needs to be changed to use the original names.

Sample IDL

Consider the following sample IDL, with an interface called `grid`

```
interface grid {
    readonly attribute short height; // height of the grid
    readonly attribute short width; // width of the grid

    // IDL operations

    // set the element [n,m] of the grid, to value:
    void set(in short n, in short m, in long value);

    // return element [n,m] of the grid:
    long get(in short n, in short m);
};
```

Main Copybook Sample for GRID using version v2r3m5 (or higher)

The `gencl` version `v2r3m5` (or higher) outputs the following for the preceding IDL:

```
01 GRID-OPERATION                PICTURE X(17).
   88 GRID-GET-HEIGHT             VALUE "_get_height".
   88 FQ-GRID-GET-HEIGHT          VALUE "_get_height:grid".
   88 GRID-GET-WIDTH              VALUE "_get_width".
   88 FQ-GRID-GET-WIDTH           VALUE "_get_width:grid".
   88 GRID-IDL-SET                VALUE "set".
   88 FQ-GRID-IDL-SET             VALUE "set:grid".
   88 GRID-IDL-GET                VALUE "get".
   88 GRID-IDL-GET                VALUE "get".
   88 FQ-GRID-IDL-GET             VALUE "get:grid".
```

Note the extra entry per operation.

Orbix 6.2 IDL Compiler Output

The Orbix 6.2 IDL Compiler generates the following output for the `grid` interface:

```

01 GRID-OPERATION                                PICTURE X(25).
  88 GRID-GET-HEIGHT                             VALUE
      "_get_height:IDL:grid:1.0".
  88 GRID-GET-WIDTH                             VALUE
      "_get_width:IDL:grid:1.0".
  88 GRID-IDL-SET                               VALUE
      "set:IDL:grid:1.0".
  88 GRID-IDL-GET                               VALUE
      "get:IDL:grid:1.0".
01 GRID-OPERATION-LENGTH                       PICTURE 9(09) BINARY
      VALUE 25.

```

There is no extra entry per operation, and each entry contains all the necessary information in the level 88 string, that is, the operation name (and the module and interface name) it relates to.

Changes on the Client-Side

The following client code needs to be changed for the preceding IDL:

```

* Try to read the height and width of the grid.
  set fq-grid-get-height    to true
  call "ORBEXEC"           using grid-obj
                           grid-operation
                           grid-height-args

```

to:

```

* Try to read the height and width of the grid.
  set grid-get-height      to true
  call "ORBEXEC"           using grid-obj
                           grid-operation
                           grid-height-args

```

In Summary

Affects clients and requires minor code changes.

Numeric Suffixes for Data Names

Overview

This subsection summarizes the differences in the way that `gencbl` and the Orbix 6.2 IDL Compiler add numeric suffixes to generate unique data names for IDL identifier names. It discusses the following topics:

- [The `gencbl` utility](#)
- [Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

The `gencbl` utility

The `gencbl` utility generates unique data names by attaching numeric suffixes to them (starting at -1). It used this method regardless of whether the number was ever used. Therefore, in nested levels of IDL, some of the generated data names appeared to skip numbers.

Refer to [“Name Scoping and the COBOL Compilers” on page 97](#) for an example of how this works.

Orbix 6.2 IDL Compiler

The Orbix 6.2 IDL Compiler does not skip numbers in this way. Therefore, some of the data names that it generates (especially where nested sequences are used) are different from the names generated by `gencbl`.

Migration Impact

Affects source code where nesting of sequences and other complex types occurs.

160-Character Limit for String Literals

Overview

IDL typecodes are mapped to string literals in COBOL using a `level 01` data name and within it the typecodes as `level 88` data names. However, the IBM COBOL compiler does not allow string literals that exceed 160 characters.

This subsection discusses the following topics:

- [The `gencl` Utility Solution](#)
- [The Orbix 6.2 IDL Compiler Solution](#)
- [Sample IDL](#)
- [The `gencl` Output](#)
- [The Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)
- [In Summary](#)

The `gencl` Utility Solution

To get around this problem, an extra undocumented argument was supplied (the `-D` argument) with `gencl` (version 2.3.1 and later), to generate typecodes in a non-OMG-compliant manner. To use these typecodes, some minor changes were required to application source code for passing sequences.

The Orbix 6.2 IDL Compiler Solution

The Orbix 6.2 IDL Compiler resolves this issue by ensuring that the typecode representations produced rarely exceed 160 characters, and thus can always be defined as a `level 88` item. The `level 88` items produced are not actually typecodes; they are unique strings representing the keys which the COBOL runtime interprets to derive the typecode using the `idlmembernameX` copybook at runtime.

Sample IDL

Consider the following IDL sample, called the `SOLUTION` member:

```
interface solution {
    struct PersonInfo {
        string  FirstName;
        string  MiddleName;
        string  SurName;
        boolean Married;
        unsigned long
            Age;
        char    Sex;
        unsigned long
            NoChildren;
    };
    struct WorkInfo {
        string  JobTitle;
        string  Department;
        string  CompanyName;
        char    Grade;
        float   Salary;
        boolean HealthIns;
        boolean Overtime;
        boolean CompanyCar;
        boolean Expenses;
        unsigned
            long    YearsService;
        string  Miscdetls;
    };
    struct AddressInfo {
        short HouseNumber;
        string AddressLine1;
        string AddressLine2;
        string AddressLine3;
        string AddressLine4;
        string PostalCode;
        string City;
        string State;
        string Country;
        string Continent;
    };
    struct CustInfo {
        PersonInfo  PersonDetls;
        AddressInfo AddressDetls;
        WorkInfo    WorkDetls;
    };
};
```


The Orbix 6.2 IDL Compiler Output

For the preceding IDL, the Orbix 6.2 IDL Compiler generates the following typecode section in the main copybook:

```
*****
* Typecode section
* This contains CDR encodings of necessary typecodes.
*****
01 SOLUTION-TYPE                                PICTURE X(28).
COPY CORBATYP.
    88 SOLUTION-ADDRESSINFO                    VALUE
        "IDL:solution/AddressInfo:1.0".
    88 SOLUTION-CUSTDETLs                      VALUE
        "IDL:solution/CustDetls:1.0".
    88 SOLUTION-CUSTINFO                       VALUE
        "IDL:solution/CustInfo:1.0".
    88 SOLUTION                                VALUE
        "IDL:solution:1.0".
    88 SOLUTION-WORKINFO                       VALUE
        "IDL:solution/WorkInfo:1.0".
    88 SOLUTION-PERSONINFO                     VALUE
        "IDL:solution/PersonInfo:1.0".
01 SOLUTION-TYPE-LENGTH                        PICTURE S9(09) BINARY
                                                VALUE 28.
```

Migration Impact

Customers that used a non-OMG-compliant version of `gencb1` with the alternative typecode mapping must now revert back to the OMG way of coding their applications.

From the `gencb1` output which uses the `-D` argument, the code to set the type in a sequence for the preceding IDL is:

```
CALL "STRSET" USING SEQUENCE-TYPE OF ...my-sequence...
TC-CUSTDETLs-TYPE-LENGTH
TC-CUSTDETLs-TYPE.
```

From the Orbix 6.2 IDL Compiler output which is OMG compliant the code to set the type in a sequence for the preceding IDL is:

```
SET SOLUTION-CUSTDETLs TO TRUE
CALL "STRSET" USING SEQUENCE-TYPE OF ...my-sequence...
SOLUTION-TYPE-LENGTH
SOLUTION-TYPE.
```

In Summary

Requires code changes to application source code using sequences.

Maximum Length of COBOL Data Names

Overview

This subsection summarizes the differences in the way that the `gencb1` utility and the Orbix 6.2 IDL Compiler process IDL identifier names that exceed 30 characters. It discusses the following topics:

- [The `gencb1` Utility Approach](#)
 - [Problems with the `gencb1` Utility Approach](#)
 - [Orbix 6.2 IDL Compiler Approach](#)
 - [Sample IDL](#)
 - [Data Names Generated by `gencb1`](#)
 - [Data Names Generated by the Orbix 6.2 IDL Compiler](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

The `gencb1` Utility Approach

Because COBOL places a 30-character restriction on the length of data names, a method to resolve this issue is provided with the `gencb1` utility. For any identifiers exceeding 30 characters, this method truncates the identifier name to the first 27 characters and attaches a three-character numeric suffix.

Problems with the `gencb1` Utility Approach

This method is prone to problems if the original IDL for a completed application has to be subsequently modified, and the modifications involve IDL identifiers exceeding 30 characters being added before existing operations or arguments. In this case, the regenerated suffixes for the various data names do not match the original suffixes generated. This results in customers having to make undesirable source code changes.

Orbix 6.2 IDL Compiler Approach

To avoid this problem, a new method has been implemented with the Orbix 6.2 IDL Compiler. This new method ensures that the same suffix is always regenerated for a particular data name.

Sample IDL

Consider the following IDL:

```
interface longname{
struct complex {
long
    thisIsAreallyLongFeatureNamewithAnotherReallyLongFeatureExtensionAtTheEnd;
long
    yetAnotherReallyLongFeatureNamewithAnotherReallyLongFeatureExtension;
long
    ThirdLastYetAnotherReallyLongFeatureNamewithAnotherReallyLongFeatureExtension;
};
void initialise();
void op1(in complex ii);
complex op2(in complex ii, inout complex io, out complex oo);
};
```

Data Names Generated by genclb

The genclb utility generated data names as follows, based on the preceding IDL:

```
01 LONGNAME-OP1-ARGS.
03 II.
05 THISISAREALLYLONGFEATURENAMEWI    PICTURE S9(09) BINARY.
05 YETANOTHERREALLYLONGFEATURENAM    PICTURE S9(09) BINARY.
05 THIRDLASTYETANOTHERREALLYLONGF    PICTURE S9(09) BINARY.

01 LONGNAME-OP2-ARGS.
03 II.
05 THISISAREALLYLONGFEATURENAM000    PICTURE S9(09) BINARY.
05 YETANOTHERREALLYLONGFEATURE001    PICTURE S9(09) BINARY.
05 THIRDLASTYETANOTHERREALLYLO002    PICTURE S9(09) BINARY.
03 IO.
05 THISISAREALLYLONGFEATURENAM003    PICTURE S9(09) BINARY.
05 YETANOTHERREALLYLONGFEATURE004    PICTURE S9(09) BINARY.
05 THIRDLASTYETANOTHERREALLYLO005    PICTURE S9(09) BINARY.
03 OO.
05 THISISAREALLYLONGFEATURENAM006    PICTURE S9(09) BINARY.
05 YETANOTHERREALLYLONGFEATURE007    PICTURE S9(09) BINARY.
05 THIRDLASTYETANOTHERREALLYLO008    PICTURE S9(09) BINARY.
```

Data Names Generated by the Orbix 6.2 IDL Compiler

The Orbix 6.2 IDL Compiler generates data names as follows, based on the preceding IDL:

```

01 LONGNAME-OP1-ARGS.
  03 II.
    05 THISISAREALLYLONGFEATUREN-E658      PICTURE S9(10) BINARY.
    05 YETANOTHERREALLYLONGFEATU-7628     PICTURE S9(10) BINARY.
    05 THIRDLASTYETANOTHERREALLY-E278     PICTURE S9(10) BINARY.
01 LONGNAME-OP2-ARGS.
  03 II.
    05 THISISAREALLYLONGFEATUREN-E658      PICTURE S9(10) BINARY.
    05 YETANOTHERREALLYLONGFEATU-7628     PICTURE S9(10) BINARY.
    05 THIRDLASTYETANOTHERREALLY-E278     PICTURE S9(10) BINARY.
  03 IO.
    05 THISISAREALLYLONGFEATUREN-E658      PICTURE S9(10) BINARY.
    05 YETANOTHERREALLYLONGFEATU-7628     PICTURE S9(10) BINARY.
    05 THIRDLASTYETANOTHERREALLY-E278     PICTURE S9(10) BINARY.
  03 OO.
    05 THISISAREALLYLONGFEATUREN-E658      PICTURE S9(10) BINARY.
    05 YETANOTHERREALLYLONGFEATU-7628     PICTURE S9(10) BINARY.
    05 THIRDLASTYETANOTHERREALLY-E278     PICTURE S9(10) BINARY.
  03 RESULT.
    05 THISISAREALLYLONGFEATUREN-E658      PICTURE S9(10) BINARY.
    05 YETANOTHERREALLYLONGFEATU-7628     PICTURE S9(10) BINARY.
    05 THIRDLASTYETANOTHERREALLY-E278     PICTURE S9(10) BINARY.

```

Migration Impact

This change means that completely different suffixes are generated where this scenario applies with the result that any application code that references these data names has to be changed to reference the data names with the Orbix 6.2 suffixes.

In Summary

Affects clients and servers where IDL identifiers exceed 30 characters.
Requires code changes.

Copybook Names Based on IDL Member Name

Overview

Copybook names in Orbix 6.2 are generated based on the IDL member name, instead of being based on the interface name, as is the case with `gencb1`. The reason for this change is because the Orbix 6.2 IDL Compiler can process any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces). If the same interface name is defined at different levels within the same IDL member, it is impossible to base copybook names on interface names.

In this section

This section discusses the following topics:

Introduction to IDL Member Name Migration Issues	page 90
IDL Member Name Different from its Interface Names	page 92
More than One Interface in an IDL Member	page 94
Length of IDL Member Names	page 96

Introduction to IDL Member Name Migration Issues

Overview

This subsection describes migration issues relating to IDL member names. It discusses the following topics:

- [Sample IDL](#)
- [The genctl utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

Sample IDL

For example, consider the following IDL member called `myidl`:

```
//myidl
module m1
{
    interface fred
    {
        void myop();
    };
    module m2
    {
        interface fred
        {
            void myop();
        };
    };
};
```

The genctl utility

The `genctl` utility cannot correctly process the preceding IDL, because it contains more than one level of module.

Because both interfaces share the same name, which is `fred` in the preceding example, the generation of one copybook would overwrite the other.

The Orbix 6.2 IDL Compiler

The Orbix 6.2 IDL Compiler instead generates COBOL copybooks whose names are based on the IDL member name, which is `myidl` in the preceding example. Therefore, the definitions for all the interfaces contained within this IDL member are produced in the `MYIDL` copybooks. (This is also how the IDL compiler generates C++ and Java files.)

Migration Impact

This has a migration impact if either of the following apply:

- IDL member names are different from the interface names they contain.
- More than one interface is defined in an IDL member.

The migration impact for each of these situations is described in the following subsections.

IDL Member Name Different from its Interface Names

Overview

This subsection summarizes the different outputs for `gencb1` and the Orbix 6.2 IDL Compiler for an IDL member that has one interface which has a name different from the member name. It discusses the following topics:

- [Sample IDL](#)
- [The `gencb1` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Workaround](#)
- [In Summary](#)

Sample IDL

Consider the following IDL member, `GRID`, which defines an interface called `fred`:

```
//grid.idl
interface fred
{
    void myop(in long mylong);
};
```

The `gencb1` Utility

In the case of the `gencb1` utility, the generated copybook names are based on the interface name, which is `fred` in the preceding example.

The Orbix 6.2 IDL Compiler

In the case of the Orbix 6.2 IDL Compiler, the generated copybook names are based on the IDL member name, which is `grid` in the preceding example.

Workaround

If your IDL member name is not the same as the interface name it contains, you can use the `-o` argument with the Orbix 6.2 IDL Compiler to map the names of the generated COBOL copybooks (which in Orbix 6.2 is based by default on the IDL member name) to alternative names. This means you can change the Orbix 6.2 default names to the `gencb1` generated names, and thus avoid having to change the `COPY` statements (for example, from `COPY FRED` to `COPY GRID`) in your application source code. The names of the generated COBOL copybooks are then automatically changed to the

alternative name that you specify with the `-o` argument. Refer to the *COBOL Programmer's Guide and Reference* for an example of how to use the `-o` argument.

In Summary

Affects clients and servers. Requires minor code change or use of the described workaround.

More than One Interface in an IDL Member

Overview

This subsection summarizes the different outputs for `gencb1` and the Orbix 6.2 IDL Compiler for an IDL member that has more than one interface, each with different names. It discusses the following topics:

- [The `gencb1` Utility](#)
 - [The Orbix 6.2 IDL Compiler](#)
 - [Sample IDL](#)
 - [Compiler Output](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

The `gencb1` Utility

The `gencb1` utility generates a set of copybooks for each interface definition, and bases the name for each set of copybooks on the associated interface name.

The Orbix 6.2 IDL Compiler

The Orbix 6.2 IDL Compiler generates only one set of COBOL copybooks for an IDL member, and it bases the name for that set of copybooks on the IDL member name.

If an IDL member contains N interfaces (where N is greater than one), your existing application code now contains $N-1$ redundant `COPY` statements.

Sample IDL

Consider the following IDL member, called `GRID`, which contains the following two interfaces:

```
interface grid
{
    void sizeofgrid(in long mysize1, in long
        mysize2);
};

interface block
{
    void area(in long myarea);
};
```

Compiler Output

The differences in the way `gencb1` and the Orbix 6.2 IDL Compiler process the preceding IDL can be outlined as follows:

Table 7: *COBOL Compiler Output for GRID IDL Member*

The Orbix 6.2 IDL Compiler	The gencb1 Utility
<p>Generates only one set of copybooks that contain all the definitions for all interfaces contained within the IDL member. The copybook names are based on the IDL member name. For example:</p> <p>GRID GRIDX GRIDD</p>	<p>Generates a set of copybooks for each interface, based on each interface name. For example:</p> <p>GRID GRIDX GRIDD BLOCK BLOCKX BLOCKD</p>

Migration Impact

Based on the preceding example, the `BLOCK` copybooks are redundant with the Orbix 6.2 IDL Compiler. Therefore, the `COPY` statements for the `BLOCK` copybook must be removed from the application code.

In Summary

Affects clients and servers. Requires minor code change.

Length of IDL Member Names

Overview

This subsection summarizes the different ways that `gencbl` and the Orbix 6.2 IDL compiler generate member names from IDL member names. It discusses the following topics:

- [The `gencbl` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

The `gencbl` Utility

The `gencbl` utility bases generated member names on the interface name. It ensures that generated member names have a maximum of eight characters including one of the following suffixes: `SV`, `X`, `D`, or `Z`.

The Orbix 6.2 IDL Compiler

Generated member names are based on the IDL member name and are restricted to a maximum of eight characters, including the suffix, which can be one of the following: `SV`, `X`, `D`, or `S`.

Migration Impact

If the IDL member name is longer than six characters, only the first six are used for prefixes for the generated copybook member or source code member.

Name Scoping and the COBOL Compilers

Overview

This section summarizes the differences between how `gencbl` and the Orbix 6.2 IDL Compiler handle a situation where the same data names are referenced within the same 01 level, even if the data names are fully qualified.

IBM Error Code

The IBM COBOL and Enterprise COBOL compilers produce an error message similar to the following if the same data names are referenced within the same 01 level, even if the data names are fully qualified:

```
IGYPS0037-S XXX was not a uniquely defined name. The definition
to be used could not be determined from the context. The
reference to the name was discarded.
```

Problem Scenarios

The problem can arise in either of the following scenarios:

- If the same container name is used more than once.
- If the same fieldname is used more than once.

In This Section

This section discusses the following topics:

Same Container Name Used More than Once	page 98
Same Fieldname Used More than Once	page 105

Same Container Name Used More than Once

In This Section

This subsection discusses migration issues relating to the IBM COBOL and Enterprise COBOL compilers and container names. It discusses the following topics:

- [Sample IDL](#)
- [The genctl Utility Output](#)
- [COBOL Compiler Problem](#)
- [Orbix 6.2 IDL Compiler Solution](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)
- [In Summary](#)

Sample IDL

Consider how `CBObjectInfo` is used in the following IDL:

Example 1: IDL Example for use of Structs (Sheet 1 of 2)

```
//IDL
module contain {

// CB Object

struct CBObjectInfo {
    string id;
    string lastChangedDateTime;
    string lastChangedUserID;
};

// Email Info Record

struct EmailAddressInfo {
    CBObjectInfo info;
    short addressType;
    string emailAddress;
    string availability;
};

typedef sequence <EmailAddressInfo> EmailAddressInfos;
```

Example 1: IDL Example for use of Structs (Sheet 2 of 2)

```

// Phone Number Info Record

struct PhoneNumberInfo {
    CObjectInfo info;
    short addressType;
    string phoneNumber;
    string availability;
};

typedef sequence <PhoneNumberInfo> PhoneNumberInfos;

// Street Address Info Record

struct StreetAddressInfo {
    CObjectInfo info;
    short addressType;
    string addressString1;
    string addressString2;
    string addressString3;
    string city;
    string stateProvince;
    string country;
    string postalCode;
    string availability;
};

typedef sequence <StreetAddressInfo> StreetAddressInfos;

struct ContactPointInfo {
    CObjectInfo info;
    string contactPointName;
    string timeZone;
    string description;
    string notes;
    EmailAddressInfos emailAddressList;
    PhoneNumberInfos phoneNumberList;
    StreetAddressInfos streetAddressList;
};

typedef sequence <ContactPointInfo> ContactPointInfos;

interface ContactPointInterface {
void    createContactPoint (inout ContactPointInfo cpInfo);

};
};

```

The gencl Utility Output

The `gencl` utility generates the following based on the preceding IDL:

Example 2: gencl output for IDL for use of Structs (Sheet 1 of 2)

```

*
* Operation : createContactPoint
* Parameters : inout struct ContactPointInfo cpInfo
*
01 CONTACTPOINTINTERFACE-CRE-ARGS.
  03 CPINFO.
    05 INFO.
      07 IDL-ID                                POINTER.
      07 LASTCHANGEDDATETIME                  POINTER.
      07 LASTCHANGEDUSERID                    POINTER.
    05 CONTACTPOINTNAME                        POINTER.
    05 TIMEZONE                                POINTER.
    05 DESCRIPTION                             POINTER.
    05 NOTES                                   POINTER.
    05 EMAILADDRESSLIST-2.
      07 EMAILADDRESSLIST.
        09 INFO.
          11 IDL-ID                            POINTER.
          11 LASTCHANGEDDATETIME              POINTER.
          11 LASTCHANGEDUSERID                POINTER.
        09 ADDRESSTYPE                          PICTURE S9(04) BINARY.
        09 EMAILADDRESS                         POINTER.
        09 AVAILABILITY                         POINTER.
    05 EMAILADDRESSLIST-2-SEQUENCE.
      07 SEQUENCE-MAXIMUM                       PICTURE 9(09) BINARY.
      07 SEQUENCE-LENGTH                       PICTURE 9(09) BINARY.
      07 SEQUENCE-BUFFER                       POINTER.
      07 SEQUENCE-TYPE                         POINTER.
    05 PHONENUMBERLIST-2.
      07 PHONENUMBERLIST.
        09 INFO.
          11 IDL-ID                            POINTER.
          11 LASTCHANGEDDATETIME              POINTER.
          11 LASTCHANGEDUSERID                POINTER.
        09 ADDRESSTYPE                          PICTURE S9(04) BINARY.
        09 PHONENUMBER                         POINTER.
        09 AVAILABILITY                         POINTER.
    05 PHONENUMBERLIST-2-SEQUENCE.
      07 SEQUENCE-MAXIMUM                       PICTURE 9(09) BINARY.
      07 SEQUENCE-LENGTH                       PICTURE 9(09) BINARY.
      07 SEQUENCE-BUFFER                       POINTER.
      07 SEQUENCE-TYPE                         POINTER.

```

Example 2: *gencbl* output for IDL for use of Structs (Sheet 2 of 2)

```

05 STREETADDRESSLIST-2.
  07 STREETADDRESSLIST.
    09 INFO.
      11 IDL-ID                      POINTER.
      11 LASTCHANGEDDATETIME        POINTER.
      11 LASTCHANGEDUSERID          POINTER.
    09 ADDRESSTYPE                    PICTURE S9(04) BINARY.
    09 ADDRESSSTRING1                POINTER.
    09 ADDRESSSTRING2                POINTER.
    09 ADDRESSSTRING3                POINTER.
    09 CITY                          POINTER.
    09 STATEPROVINCE                 POINTER.
    09 COUNTRY                       POINTER.
    09 POSTALCODE                    POINTER.
    09 AVAILABILITY                  POINTER.
  05 STREETADDRESSLIST-2-SEQUENCE.
    07 SEQUENCE-MAXIMUM               PICTURE 9(09) BINARY.
    07 SEQUENCE-LENGTH               PICTURE 9(09) BINARY.
    07 SEQUENCE-BUFFER               POINTER.
    07 SEQUENCE-TYPE                 POINTER.

```

COBOL Compiler Problem

In the preceding example, the `IDL-ID` under `INFO` under `CPINFO` is treated as ambiguous by the IBM COBOL and Enterprise COBOL compilers, because of the presence of other group levels under the same `01` level that are also called `INFO`.

Orbix 6.2 IDL Compiler Solution

The Orbix 6.2 IDL Compiler provides a solution to this problem, whereby it attaches a numeric suffix (starting at -1, that is, 1 with a hyphen) to any group level reference that is used more than once under the same `01` level.

Orbix 6.2 IDL Compiler Output

The Orbix 6.2 IDL Compiler generates the following COBOL code, based on the preceding IDL:

Example 3: *Orbix 6.2 Compiler output for Structs IDL (Sheet 1 of 3)*

```
*****
* Operation:      createContactPoint
* Mapped name:   createContactPoint
* Arguments:     <inout> contain/ContactPointInfo cpInfo
* Returns:       void
* User Exceptions: none
*****
01 IDL-CONTAIN-CONTACTP-E3BE-ARGS.
   03 CPINFO.
      05 INFO.
         07 IDL-ID                                POINTER
                                                VALUE NULL.
         07 LASTCHANGEDDATETIME                 POINTER
                                                VALUE NULL.
         07 LASTCHANGEDUSERID                   POINTER
                                                VALUE NULL.
      05 CONTACTPOINTNAME                       POINTER
                                                VALUE NULL.
      05 TIMEZONE                               POINTER
                                                VALUE NULL.
      05 DESCRIPTION                            POINTER
                                                VALUE NULL.
      05 NOTES                                  POINTER
                                                VALUE NULL.
      05 EMAILADDRESSLIST-1.
         07 EMAILADDRESSLIST.
            09 INFO-1.
               11 IDL-ID                        POINTER
                                                VALUE NULL.
               11 LASTCHANGEDDATETIME           POINTER
                                                VALUE NULL.
               11 LASTCHANGEDUSERID             POINTER
                                                VALUE NULL.
            09 ADDRESSTYPE                       PICTURE S9(05)BINARY.
            09 EMAILADDRESS                     POINTER
                                                VALUE NULL.
            09 AVAILABILITY                      POINTER
                                                VALUE NULL.
```

Example 3: *Orbis 6.2 Compiler output for Structs IDL (Sheet 2 of 3)*

```

05 EMAILADDRESSLIST-SEQUENCE.
    07 SEQUENCE-MAXIMUM          PICTURE 9(09) BINARY
                                VALUE 0.
    07 SEQUENCE-LENGTH          PICTURE 9(09) BINARY
                                VALUE 0.
    07 SEQUENCE-BUFFER          POINTER
                                VALUE NULL.
    07 SEQUENCE-TYPE            POINTER
                                VALUE NULL.
05 PHONENUMBERLIST-1.
    07 PHONENUMBERLIST.
        09 INFO-2.
            11 IDL-ID           POINTER
                                VALUE NULL.
            11 LASTCHANGEDDATETIME  POINTER
                                VALUE NULL.
            11 LASTCHANGEDUSERID  POINTER
                                VALUE NULL.
        09 ADDRESSTYPE          PICTURE S9(05) BINARY.
        09 PHONENUMBER          POINTER
                                VALUE NULL.
        09 AVAILABILITY         POINTER
                                VALUE NULL.
05 PHONENUMBERLIST-SEQUENCE.
    07 SEQUENCE-MAXIMUM          PICTURE 9(09) BINARY
                                VALUE 0.
    07 SEQUENCE-LENGTH          PICTURE 9(09) BINARY
                                VALUE 0.
    07 SEQUENCE-BUFFER          POINTER NULL.
    07 SEQUENCE-TYPE            POINTER
                                VALUE NULL.
05 STREETADDRESSLIST-1.
    07 STREETADDRESSLIST.
        09 INFO-3.
            11 IDL-ID           POINTER
                                VALUE NULL.
            11 LASTCHANGEDDATETIME  POINTER
                                VALUE NULL.
            11 LASTCHANGEDUSERID  POINTER
                                VALUE NULL.
        09 ADDRESSTYPE          PICTURE S9(05) BINARY.

```

Example 3: *Orbix 6.2 Compiler output for Structs IDL (Sheet 3 of 3)*

09 ADDRESSSTRING1	POINTER
	VALUE NULL.
09 ADDRESSSTRING2	POINTER
	VALUE NULL.
09 ADDRESSSTRING3	POINTER
	VALUE NULL.
09 CITY	POINTER
	VALUE NULL.
09 STATEPROVINCE	POINTER
	VALUE NULL.
09 COUNTRY	POINTER
	VALUE NULL.
09 POSTALCODE	POINTER
	VALUE NULL.
09 AVAILABILITY	POINTER
	VALUE NULL.
05 STREETADDRESSLIST-SEQUENCE.	
07 SEQUENCE-MAXIMUM	PICTURE 9(09) BINARY
	VALUE 0.
07 SEQUENCE-LENGTH	PICTURE 9(09) BINARY
	VALUE 0.
07 SEQUENCE-BUFFER	POINTER
	VALUE NULL.
07 SEQUENCE-TYPE	POINTER
	VALUE NULL.

Migration Impact

This change means that completely different suffixes are generated where this scenario applies, with the result that any application code that references these data names has to be changed to reference the data names with the new suffixes.

In Summary

Affects both client and server application code.

Same Fieldname Used More than Once

In This Section

This subsection describes migration issues relating to the IBM COBOL and Enterprise COBOL compilers and fieldnames. It discusses the following topics:

- [Sample IDL](#)
- [Orbix 6.2 COBOL IDL Compiler Output](#)
- [Migration Impact](#)

Sample IDL

Consider the following IDL:

```
//IDL

interface sample
{
  struct ClmSum {
  short  int_div_id;
  };

  typedef sequence<ClmSum,30> ClmSumSeq;

  struct MemClmRsp {
  string more_data_sw;
  short  int_div_id;
  long  claim_micro_sec_id;
  ClmSumSeq MemClmList;
  };

  short getSummary(
  out MemClmRsp MemClaimList);
}
```

Orbix 6.2 COBOL IDL Compiler Output

For the preceding IDL sample, the relevant COBOL output is the main copybook:

```
*****
* Operation:      getSummary
* Mapped name:    getSummary
* Arguments:      <out> sample/MemClmRsp MemClaimList
* Returns:        short
* User Exceptions: none
*****
01 SAMPLE-GETSUMMARY-ARGS.
  03 MEMCLAIMLIST.
    05 MORE-DATA-SW                                POINTER
                                                VALUE NULL.
    05 INT-DIV-ID                                  PICTURE S9(05) BINARY.
    05 CLAIM-MICRO-SEC-ID                          PICTURE S9(10) BINARY.
    05 MEMCLMLIST-1                                OCCURS 30 TIMES.
      07 MEMCLMLIST.
        09 INT-DIV-ID                              PICTURE S9(05) BINARY.
    05 MEMCLMLIST-SEQUENCE.
      07 SEQUENCE-MAXIMUM                          PICTURE 9(09) BINARY
                                                VALUE 30.
      07 SEQUENCE-LENGTH                          PICTURE 9(09) BINARY
                                                VALUE 0.
      07 SEQUENCE-BUFFER                          POINTER
                                                VALUE NULL.
      07 SEQUENCE-TYPE                             POINTER
                                                VALUE NULL.
    03 RESULT                                      PICTURE S9(05) BINARY.
```

Migration Impact

The copybook that is generated, based on the preceding IDL, has two references to `int_div_id`, but only one is accessible because of COBOL name scoping rules.

This problem remains unresolved.

Typecode Name and Length Identifiers

Overview

This section summarizes the different output for `gencb1` and the Orbix 6.2 IDL Compiler for typecode and typecode length data names.

In this section

This section discusses the following topics:

Comparing Compiler Output	page 108
IDL Member Name Different from its Interface Name	page 109
More than One Interface in an IDL Member	page 112

Comparing Compiler Output

Overview

This subsection describes the migration issues relating to compiler outputs for typecode and typecode length data names. It discusses the following topics:

- [The genctl utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

The genctl utility

The typecode and typecode length data names generated by `genctl` use the names `interfacename-TYPE` and `interfacename-TYPE-LENGTH`. This is not suitable for a situation where an IDL member contains multiple nested levels of modules and interfaces, because unique data names cannot be generated in this case.

The Orbix 6.2 IDL Compiler

Because the Orbix 6.2 IDL Compiler can process any level of scoping in an IDL member, the generated data names are of the form `idlmembername-TYPE` and `idlmembername-TYPE-LENGTH`. This ensures the uniqueness of the data names.

Migration Impact

However, this has a migration impact if either of the following apply:

- IDL member name is different from the interface name it contains.
- More than one interface is defined in an IDL member.

The migration impact for each of these situations is described in the following subsections.

IDL Member Name Different from its Interface Name

Overview

With `genidl` the 01 typecode name and length fields are based on the interface name. With the Orbix 6.2 IDL Compiler, 01 typecode name and length fields are based on the IDL member name.

This subsection discusses the following topics:

- [Sample IDL](#)
- [The `genidl` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)
- [In Summary](#)

Sample IDL

Consider the following IDL member, called `TEST`, with an interface named `sample`:

```
//idl member is test.idl
interface sample
{
    typedef short House_Num;
    struct Address
    {
        string name;
        House_Num number;
        string address1;
        string address2;
    };
    typedef sequence<Address,30> AddressList;
    void myop(inout AddressList alladdresses);
};
```

The gencl Utility

With `gencl`, the 01 typecode name and length fields are based on the interface name, that is, `sample-TYPE` and `01 sample-TYPE-LENGTH` where `sample` is the interface name. The `gencl` output for the preceding IDL is as follows:

```
*Typecode definitions used in the interface sample
*Use this data item for retrieving or setting the type
*information for ANYs or SEQUENCES.
*
01 SAMPLE-TYPE                                PICTURE X(87).
COPY CORBATYP.
    88 SAMPLE-HOUSE-NUM                        VALUE "s".
    88 SAMPLE-ADDRESSLIST VALUE
    "S{R~sample::Address~name{0},number{
    -
    "L~sample::House_Num~{s}},address1{0},address2{0}},30".
    88 SAMPLE-ADDRESS VALUE
    "R~sample::Address~name{0},number{L~samp
    -
    "le::House_Num~{s}},address1{0},address2{0}".

01 SAMPLE-TYPE-LENGTH                        PICTURE 9(09) BINARY
                                                VALUE 87.
```

The Orbix 6.2 IDL Compiler

With the Orbix 6.2 IDL Compiler 01 typecode name and length fields are based on the IDL member name, that is `test-TYPE` and 01 `test-TYPE-LENGTH`, where `test` is the IDL member name. The Orbix 6.2 output in the main copybook by default for the preceding IDL is as follows:

```
*****
* Typecode section
* This contains CDR encodings of necessary typecodes.
*****
01 TEST-TYPE                                PICTURE X(26).
      COPY CORBATYP.
      88 SAMPLE-HOUSE-NUM                    VALUE
         "IDL:sample/House_Num:1.0".
      88 SAMPLE-ADDRESS                      VALUE
         "IDL:sample/Address:1.0".
      88 SAMPLE                              VALUE
         "IDL:sample:1.0".
      88 SAMPLE-ADDRESSLIST                 VALUE
         "IDL:sample/AddressList:1.0".
01 TEST-TYPE-LENGTH                        PICTURE S9(09) BINARY
                                           VALUE 26.
```

Because `TEST` is the IDL member name, the 01 levels are prefixed with `TEST`. The main copybook name is based on the IDL member name and cannot exceed six characters, and in this case is called `TEST`.

Migration Impact

If your IDL member name is not the same as the interface name it contains, you can use the `-o` argument with the Orbix 6.2 IDL Compiler to make both names the same and thereby avoid application code changes. The `-o` argument allows you to change, for example, `XXXX` in `XXXX-TYPE` and `XXXX` in `XXXX-TYPE-LENGTH`. For the preceding Orbix 6.2 IDL Compiler output to avoid source code changes would mean changing `TEST` in `TEST-TYPE` and `TEST` in `TEST-TYPE-LENGTH` to `SAMPLE-TYPE` and `SAMPLE-TYPE-LENGTH`. The `-o` argument does not restrict you the use of either the interface name or the IDL member name.

Refer to the *COBOL Programmer's Guide and Reference* for an example of how to use the `-o` argument.

In Summary

Affects clients and servers. Requires code change or use of the `-o` argument.

More than One Interface in an IDL Member

In This Section

This subsection describes the migration issues for typecode and typecode length data names where there is more than one interface in an IDL member. It discusses the following topics:

- [The genctl Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Sample IDL](#)
- [The genctl output](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)
- [In Summary](#)

The genctl Utility

With `genctl`, the 01 typecode name and length fields are based on the interface name, that is, `sample-TYPE` and `sample-TYPE-LENGTH` where `sample` is the interface name.

The Orbix 6.2 IDL Compiler

With the Orbix 6.2 IDL Compiler, 01 typecode name and length fields are based on the IDL member name, that is `test-TYPE` and 01 `test-TYPE-LENGTH`, where `test` is the IDL member name.

Sample IDL

For example, consider the following IDL member, called `TEST`, which contains the two interfaces called `sample` and `example` respectively:

```
//idl member is test.idl test
interface sample
{
    typedef short House_Num;
    struct Address
    {
        string name;
        House_Num number;
        string address1;
        string address2;
    };
    typedef sequence<Address,30> AddressList;
    void myop(inout AddressList alladdresses);
};

interface example
{
    typedef long Account_Num;
    struct Account_Details
    {
        string name;
        Account_Num number;
        string address1;
        string address2;
    };
    typedef sequence<Account_Details,30> AccountList;
    void myop(inout AccountList allaccounts);
};
```

The gencl output

The gencl output for the `example` interface in `TEST` is as follows:

```

** Typecode definitions used in the interface xample
* Use this data item for retrieving or setting the type
* information for ANYs or SEQUENCES.
*
01 EXAMPLE-TYPE                                PICTURE X(90).
   COPY CORBATYP.
   88 EXAMPLE-ACCOUNT-NUM                      VALUE "1".
   88 EXAMPLE-ACCOUNTLIST VALUE
      "S{R~Account_Details~name{0},number
      -" {L~example::Account_Num~{1}},address1{0},address2{0}},30".
   88 EXAMPLE-ACCOUNT-DETAILS VALUE
      "R~Account_Details~name{0},numb
      -"er{L~example::Account_Num~{1}},address1{0},address2{0}".

01 EXAMPLE-TYPE-LENGTH                        PICTURE 9(09) BINARY
   VALUE 90.

```

The gencl output for the `sample` interface in `TEST` is as follows:

```

* Typecode definitions used in the interface sample
* Use this data item for retrieving or setting the type
* information for ANYs or SEQUENCES.
*
01 SAMPLE-TYPE                                PICTURE X(79).
   COPY CORBATYP.
   88 SAMPLE-HOUSE-NUM                          VALUE "s".
   88 SAMPLE-ADDRESSLIST VALUE
      "S{R~Address~name{0},number{L~sample
      -" :~House_Num~{s}},address1{0},address2{0}},30".
   88 SAMPLE-ADDRESS VALUE
      "R~Address~name{0},number{L~sample::Hous
      -"e_Num~{s}},address1{0},address2{0}".

01 SAMPLE-TYPE-LENGTH                        PICTURE 9(09) BINARY
   VALUE 79.

```

Orbix 6.2 IDL Compiler Output

The Orbix 6.2 output in the main copybook (by default) for the preceding IDL is as follows:

```
*****
* Typecode section
* This contains CDR encodings of necessary typecodes.
*****
01 TEST-TYPE                                PICTURE X(31).
      COPY CORBATYP.
      88 SAMPLE-HOUSE-NUM                    VALUE
         "IDL:sample/House_Num:1.0".
      88 SAMPLE-ADDRESS                      VALUE
         "IDL:sample/Address:1.0".
      88 EXAMPLE-ACCOUNTLIST                VALUE
         "IDL:example/AccountList:1.0".
      88 EXAMPLE-ACCOUNT-NUM                VALUE
         "IDL:example/Account_Num:1.0".
      88 EXAMPLE-ACCOUNT-DETAILS           VALUE
         "IDL:example/Account_Details:1.0".
      88 SAMPLE                              VALUE
         "IDL:sample:1.0".
      88 EXAMPLE                              VALUE
         "IDL:example:1.0".
      88 SAMPLE-ADDRESSLIST                 VALUE
         "IDL:sample/AddressList:1.0".
01 TEST-TYPE-LENGTH                        PICTURE S9(09)BINARY
                                             VALUE 31.
```

All the typecodes for the complete IDL member are represented under a single 01 level.

Migration Impact

Any references in application code to the `type` and `type-length` data names must be changed to reflect the IDL compiler output in the main copybook. The `-M` and `-O` arguments can assist in migration. Refer to the *COBOL Programmer's Guide and Reference* for an example of how to use the `-M` and `-O` arguments.

In Summary

Affects clients and servers using sequences or anys. Requires code changes.

Reserved COBOL and OMG Keywords

In This Section

This section discusses the following topics:

Reserved COBOL Keywords for Module or Interface Names	page 117
Use of Result as an Argument Name in IDL	page 118
OMG Mapping Standard for Unions and Exceptions	page 120

Note: The Orbix 6.2 IDL compiler supports the COBOL reserved word list, pertaining to the Enterprise COBOL Compiler and the IBM OS/390 Compiler.

Reserved COBOL Keywords for Module or Interface Names

Overview

This subsection describes the different ways that `gencbl` and the Orbix 6.2 IDL Compiler treat COBOL keywords used as module or interface names. It discusses the following topics:

- [The `gencbl` utility](#)
 - [The Orbix 6.2 IDL Compiler](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

The `gencbl` utility

The `gencbl` utility does not apply special treatment to a reserved COBOL keyword used as an IDL interface or module name.

The Orbix 6.2 IDL Compiler

In Orbix 6.2, if a reserved COBOL keyword is used as an IDL interface or module name, the Orbix 6.2 IDL Compiler prefixes it with `IDL-`.

Migration Impact

This has a migration impact for any customers that use reserved COBOL keywords as IDL interface or module names. If any customers are using reserved COBOL keywords, source code changes are required to their applications to cater for `IDL-` prefixed names that are generated for identifiers in Orbix 6.2.

In Summary

Affects clients and servers where module or interface names are reserved COBOL keywords.

Use of Result as an Argument Name in IDL

Overview

If your IDL uses `RESULT` as an argument name to an operation, and it also returns a parameter, each has a data name generated at the 03 level, but both data names are `RESULT`. These are not valid in COBOL, because two 03 level entries under the same 01 level entry cannot share the same name. Refer to [“Name Scoping and the COBOL Compilers” on page 97](#) for more details.

This subsection discusses the following topics:

- [The gencl Solution](#)
- [Orbix 6.2 IDL Compiler Solution](#)
- [Migration Impact](#)
- [Sample IDL](#)
- [Orbix 6.2 IDL Compiler Data Names](#)
- [In Summary](#)

The gencl Solution

Version 2.3.2 of `gencl` resolved this issue by making `RESULT` a reserved COBOL keyword for IDL argument names and prefixing the resulting generated names with `IDL-`.

Orbix 6.2 IDL Compiler Solution

The current Orbix 6.2 IDL Compiler treats `RESULT` as a reserved COBOL keyword in all cases.

Migration Impact

There is a possible, but small, migration impact involved for any customer applications where IDL definitions are defined in the manner described at the start of this section, and the latest `gencl` version is not being used. There is also a possible migration impact if the word `RESULT` is used as any identifier in an IDL member.

Sample IDL

Consider the following IDL called `grid`:

```
//IDL
interface grid {

    long myop(inout long result);
};
```

Orbix 6.2 IDL Compiler Data Names

Based on the preceding IDL, the Orbix 6.2 IDL Compiler generates the following data names for the operation:

```
01 GRID-MYOP-ARGS.
   03 IDL-RESULT          PICTURE S9(10) BINARY.
   03 RESULT              PICTURE S9(10) BINARY.
```

In Summary

Affects any application where the IDL uses `result` as described. Require minor code change if latest `gencl` version is not being used, or if the word `result` is used as any identifier in an IDL member.

OMG Mapping Standard for Unions and Exceptions

Overview

The OMG mapping standard uses the letters `U` and `D` as identifier names for union and exception mappings (it uses both letters for each). There are two possible implications if these letters are used as identifier names in IDL:

- It might lead to problems similar to the one described in [“Name Scoping and the COBOL Compilers” on page 97](#).
- These identifiers are treated as reserved keywords by the Orbix 6.2 IDL Compiler and therefore prefixed by `IDL-` in the Orbix 6.2 IDL Compiler output. Any application code that references these must be changed to account for the new compiler output.

This subsection discusses the following topics:

- [IDL Fieldname and Container Names](#)
- [Sample IDL](#)
- [The `gencl` Utility](#)
- [The `gencl` Utility Output](#)
- [Orbix 6.2 IDL Compiler Solution](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)

IDL Fieldname and Container Names

It is strongly recommended that an IDL field name or IDL container name is not called `U` or `D` in conjunction with a union and exception respectively.

Sample IDL

The following IDL sample illustrates the use of `U` and `D` as identifier names:

```
interface example
{
    void myop(inout long d,inout long u);
};
```

The `gencl` Utility

The `gencl` utility does not treat the IDL identifier names `D` and `U` as reserved COBOL keywords.

The gencl Utility Output

Based on the preceding sample IDL, `gencl` produces the following:

```
01 EXAMPLE-MYOP-ARGS.  
03 D PICTURE S9(09) BINARY.  
03 U PICTURE S9(09) BINARY.
```

Orbix 6.2 IDL Compiler Solution

The Orbix 6.2 IDL Compiler treats `U` and `D` as COBOL reserved words and therefore they are prefixed with `IDL-` in the compiler output.

Orbix 6.2 IDL Compiler Output

For the preceding IDL the Orbix 6.2 IDL Compiler produces:

```
01 EXAMPLE-MYOP-ARGS.  
03 IDL-D PICTURE S9(10) BINARY.  
03 IDL-U PICTURE S9(10) BINARY.
```

Migration Impact

Application code that references the Orbix 2.3.x `D` and `U` data names must change to reflect the Orbix 6.2 (`IDL-` prefixed) data names.

Note: The Orbix 6.2 IDL compiler supports the COBOL reserved word list, pertaining to the Enterprise COBOL Compiler and the IBM OS/390 Compiler.

Error Checking and Exceptions

In This Section

This section discusses the following discusses:

COBOL-Specific Issue Relating to Error Checking	page 123
Error Checking Generation at Runtime for Batch Servers	page 125

COBOL-Specific Issue Relating to Error Checking

Overview

This subsection summarizes the differences between `genctl` and the Orbix 6.2 IDL Compiler in regard to error checking. It discusses the following topics:

- [The `genctl` Utility Error Checking Code](#)
- [Orbix 6.2 IDL Compiler Error Checking Code](#)
- [Migration Impact](#)

The `genctl` Utility Error Checking Code

The `genctl` utility provides an `-E` argument to generate error-checking code in the generated server mainline and implementation code. The generated error-checking code is used, for example, after each API call as follows:

```
MOVE "ORBGET" TO WS-ERROR-FUNC.  
PERFORM CHECK-STATUS.
```

Orbix 6.2 IDL Compiler Error Checking Code

The Orbix 6.2 IDL Compiler generates this error-checking code slightly differently in the generated server mainline and implementation code. For example:

```
SET WS-ORBGET TO TRUE.  
PERFORM CHECK-STATUS.
```

Note: The Orbix 6.2 IDL Compiler generates error checking code by default.

A `MOVE` statement is not required in the preceding code example, because the supplied `CORBA` static copybook contains entries such as the following for all the APIs supplied with the product:

```
01 WS-API-CALLED PICTURE X(09) VALUE SPACES .
   88 WS-ANYFREE VALUE "ANYFREE" .
   88 WS-ANYGET VALUE "ANYGET" .
   88 WS-ANYSET VALUE "ANYSET" .
   88 WS-COAERR VALUE "COAERR" .
   88 WS-COAGET VALUE "COAGET" .
   88 WS-COARUN VALUE "COARUN" .
   88 WS-COAPUT VALUE "COAPUT" .
   88 WS-COAREQ VALUE "COAREQ" .
   88 WS-MEMALLOC VALUE "MEMALLOC" .
   88 WS-MEMFREE VALUE "MEMFREE" .
```

Migration Impact

This change has no migration impact and only affects newly generated server implementation and mainline code.

Error Checking Generation at Runtime for Batch Servers

Overview

This subsection summarizes the differences between `gencl` and the Orbix 6.2 IDL Compiler in relation to the `CHECK-STATUS` paragraph used for error checking. It discusses the following topics:

- [The `gencl` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

The `gencl` Utility

The `CHECK-STATUS` paragraph is generated by `gencl` for each server.

The Orbix 6.2 IDL Compiler

The `CHECK-STATUS` paragraph is shipped as a static `CHKERRS` copybook, in the `orbixhlq.INCLUDE.COPYLIB` in Orbix 6.2. The reason that the Orbix 6.2 IDL Compiler doesn't generate this procedure is that, regardless of the IDL, the procedure code is unchanged.

Migration Impact

There is no migration impact, because all newly generated code uses the static `CHKERRS` copybook and current customer applications use the old method which is completely transparent to customers. However IONA recommend you use the `CHKERRS` copybook which shows the system exception encountered in a more user-friendly format.

Nested Unions in IDL

Overview

The Orbix 6.2 IDL Compiler can support any level of nested unions in IDL. This subsection shows the Orbix 6.2 IDL Compiler output for sample IDL with nested unions.

This section discusses the following topics:

- [Sample IDL](#)
- [The gencl utility output](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)

Sample IDL

The following sample IDL member, called `NESTUNIN`, contains nested unions:

```
interface nestunin {  
  
    struct no_constr {  
        long along;  
    };  
    struct has_constr {  
        string astring;  
    };  
    struct has_constr2 {  
        has_constr astrstr;  
    };  
    union innerunion switch(long) {  
        case 1 : no_constr a;  
        case 3: has_constr b;  
        case 9: has_constr2 c;  
        default: string f;  
    };  
    union outerunion switch(long) {  
        case 1 : no_constr a;  
        case 3: has_constr b;  
        case 9: has_constr2 c;  
        case 30: innerunion myu;  
        default: string f;  
    };  
    void opNoC (in outerunion arg);  
  
};
```

The genctl utility output

The genctl utility outputs the following based on the preceding IDL:

```

01 NESTUNIN-OPNOC-ARGS.
  03 ARG.
    05 D                                PICTURE S9(09) BINARY.
    05 U.
      07 FILLER                          PICTURE X(04).
    05 FILLER REDEFINES U.
      07 A.
        09 ALONG                          PICTURE S9(09) BINARY.
    05 FILLER REDEFINES U.
      07 B.
        09 ASTRING                         POINTER.
    05 FILLER REDEFINES U.
      07 C.
        09 ASTRSTR.
          11 ASTRING                       POINTER.
    05 FILLER REDEFINES U.
      07 MYU.
        09 D                                PICTURE S9(09) BINARY.
        09 U.
          11 FILLER                         PICTURE X(04).
        09 FILLER REDEFINES U.
          11 A.
            13 ALONG                        PICTURE S9(09) BINARY.
        09 FILLER REDEFINES U.
          11 B.
            13 ASTRING                       POINTER.
        09 FILLER REDEFINES U.
          11 C.
            13 ASTRSTR.
              15 ASTRING                     POINTER.
        09 FILLER REDEFINES U.
          11 F                                POINTER.
    05 FILLER REDEFINES U.
      07 F                                POINTER.

```

Orbix 6.2 IDL Compiler Output

The Orbix 6.2 IDL Compiler outputs the following based on the preceding IDL:

```

01 NESTUNIN-OPNOC-ARGS.
  03 ARG.
    05 D                               PICTURE S9(10) BINARY.
    05 U.
      07 FILLER                         PICTURE X(16)
                                         VALUE LOW-VALUES.

    05 FILLER REDEFINES U.
      07 A.
        09 ALONG                        PICTURE S9(10) BINARY.

    05 FILLER REDEFINES U.
      07 B.
        09 ASTRING                       POINTER.

    05 FILLER REDEFINES U.
      07 C.
        09 ASTRSTR.
          11 ASTRING                     POINTER.

    05 FILLER REDEFINES U.
      07 MYU.
        09 D-1                           PICTURE S9(10) BINARY.
        09 U-1.
          11 FILLER                       PICTURE X(08).
        09 FILLER REDEFINES U-1.
          11 A-1.
            13 ALONG                      PICTURE S9(10) BINARY.
        09 FILLER REDEFINES U-1.
          11 B-1.
            13 ASTRING                     POINTER.
        09 FILLER REDEFINES U-1.
          11 C-1.
            13 ASTRSTR-1.
              15 ASTRING                   POINTER.
        09 FILLER REDEFINES U-1.
          11 F                             POINTER.

    05 FILLER REDEFINES U.
      07 F                             POINTER.

```

The OMG-reserved letters, `u` and `D`, are used by the Orbix 6.2 IDL Compiler, in the preceding example. In the first level of nesting, `u` and `D` are suffixed by `-1` by the Orbix 6.2 IDL Compiler.

Migration Impact

The `genctl` utility output for nested unions does not cater for the situation where the same container name is used more than once in an IDL member. For problems that arise in this scenario refer to [“Same Container Name Used More than Once” on page 98](#). Customers using nested unions in their IDL are required to change the nested `D` and `U` data names generated by `genctl` to make them unique.

From the preceding example, the Orbix 6.2 IDL Compiler output for nested `D` and `U` data names are unique. If your workaround is not the same as the Orbix 6.2 IDL Compiler solution, that is, adding a suffix `-n` where `n` is an integer beginning at `1` for each level of nesting (the first nested union is prefixed by `-1` and so on), there is a migration impact.

Changes are required to application code that references identifier names in nested unions to take into account the Orbix 6.2 IDL Compiler solution.

Mapping for Arrays

Overview

This section illustrates the differences between how `gencb1` and the Orbix 6.2 IDL Compiler treats arrays in IDL. It discusses the following topics:

- [Sample IDL](#)
- [The `gencb1` Utility](#)
- [The `gencb1` Utility Output](#)
- [Orbix 6.2 IDL Compiler](#)
- [Orbix 6.2 IDL Compiler Output](#)

Sample IDL

Consider the following IDL member, called `ARRAY`:

```
interface jack
{
    typedef long arr1[5][4];
    typedef arr1 arr2[10][6];
    void op1(in arr2 p1);
};
```

The `gencb1` Utility

The `gencb1` does not generate unique names at each level for multiple nested arrays.

The `gencb1` Utility Output

The `gencb1` utility outputs the following based on the preceding IDL:

```
01 JACK-OP1-ARGS.
   03 P1-1                                OCCURS 10 TIMES.
       05 P1-2                            OCCURS 6 TIMES.
           07 P1-1                        OCCURS 5 TIMES.
               09 P1-2                    OCCURS 4 TIMES.
                   11 P1                  PICTURE S9(09) BINARY.
```

Note: The `gencb1` utility does not generate unique names at each level. This might lead to problems similar to those described in [“Name Scoping and the COBOL Compilers”](#) on page 97.

Orbix 6.2 IDL Compiler

These issues are fully resolved with the Orbix 6.2 IDL Compiler, which generates unique names for array data items.

Orbix 6.2 IDL Compiler Output

The Orbix 6.2 IDL Compiler outputs the following based on the preceding IDL:

```
01 JACK-OP1-ARGS.  
   03 P1-1 OCCURS 10 TIMES.  
     05 P1-2 OCCURS 6 TIMES.  
       07 P1-1-2 OCCURS 5 TIMES.  
         09 P1-2-2 OCCURS 4 TIMES.  
           11 P1 PICTURE S9(10) BINARY.
```

The Orbix 6.2 IDL Compiler generates unique names at each level.

Working Storage data Items and Group Moves

Overview

The Orbix 6.2 IDL Compiler has a new mapping for the IDL data types `long`, `short`, `unsigned long`, and `unsigned short`. Working storage data item definitions that use these data types are affected by this new mapping. This change might affect group moves that use these Working Storage data item definitions.

This section discusses the following topics:

- [Mapping Changes](#)
- [Reason for Mapping Changes](#)
- [Sample IDL](#)
- [Orbix 2.3.x IDL to COBOL Mapping](#)
- [Orbix 6.2 IDL to COBOL Mapping](#)
- [Migration Impact](#)

Mapping Changes

The following table represents the changes to the Working Storage data item definitions for the appropriate IDL data types:

Table 8: *COBOL Mapping Changes for IDL Data Types*

IDL Data Type	Orbix 6.2 IDL Compiler Output	gencbi Output
long	S9(10) BINARY	S9(09) BINARY
unsigned long	9(10) BINARY	9(09) BINARY
short	S9(5) BINARY	S9(4) BINARY
unsigned short	9(5) BINARY	9(4) BINARY

Reason for Mapping Changes

The mappings have been changed so that the COBOL runtime can marshal the complete range of values for `CORBA::Long`, `CORBA::ULong`, `CORBA::Short`, and `CORBA::UShort` respectively.

Sample IDL

The following IDL sample illustrates the changes for group moves using the specified data types:

```
//example idl member
interface example
{
    typedef long long_array[10];
    attribute long_array myarray;
};
```

Orbix 2.3.x IDL to COBOL Mapping

The following code sample represents the Orbix 2.3.x mapping type:

```
// genctl generated code sample
WORKING-STORAGE SECTION.
03 MY-LONG-ARRAY10          OCCURS 10.
   05 MY-LONGARRAY-ELEMENT PIC S9(9) BINARY.

03 WS-SUB                  PIC S9(09) BINARY VALUE 0.
```

Orbix 6.2 IDL to COBOL Mapping

The following code sample represents the Orbix 6.2 mapping type

```
// Orbix 6.0 IDL Compiler generated code sample
01 EXAMPLE-MYARRAY-ARGS.
   03 RESULT-1              OCCURS 10 TIMES.
   05 RESULT                PICTURE S9(10) BINARY.

*Loop incrementing WS-SUB

MOVE MY-LONG-ARRAY10(WS-SUB) TO
RESULT-1 OF EXAMPLE-MYARRAY-ARGS(WS-SUB).
```

Migration Impact

Any group move with Working Storage definitions from the `genctl` mapping type is subject to unpredictable results at runtime. All such cases should be changed to reflect the new mapping.

Mapping for IDL type Any

Overview

The type `any` mapping for COBOL has changed to comply with the OMG COBOL specification.

This section discusses the following topics:

- [Sample IDL](#)
- [The `genobl` Utility Mapping](#)
- [Orbix 6.2 Mapping](#)
- [Migration Impact](#)

Sample IDL

The following sample IDL illustrates this change:

```
interface example
{
    typedef any a_any;
    readonly attribute a_any aany;
};
```

The `genobl` Utility Mapping

The `genobl` utility outputs the following code for the preceding IDL sample:

```
*****
//Orbix COBOL 2.3 mapping
01 EXAMPLE-AANY-ARGS.
   03 RESULT.
      05 RESULT-TYPE                POINTER.
      05 RESULT-VALUE              POINTER.
      05 RESULT-RELEASE            PICTURE 9(01).
```

Orbix 6.2 Mapping

Orbix 6.2 outputs the following code for the preceding IDL sample:

```
01 EXAMPLE-AANY-ARGS.
   03 RESULT                POINTER VALUE NULL.
```

Migration Impact

There is a migration impact only for applications which reference any of the individual components of the original mapping, that is `XXX-TYPE`, `XXX-VALUE`, and the `XXX-RELEASE` data items (this is not expected).

CORBA Copybook Additions

Overview

There have been several additions to the supplied CORBA copybook.

This section discusses the following topics:

- [Migration Impact](#)
- [Workaround](#)
- [CORBA Copybook Definition Example](#)

Migration Impact

There is a possibility that some of the names might conflict with those defined in you application. For a complete list of identifier names please refer to the copybook located in `orbixhlq.INCLUDE.COPYLIB`.

Workaround

If any compile errors occur make the necessary changes to the application to resolve them.

CORBA Copybook Definition Example

The following definition is defined in the CORBA copybook:

```
01 ORBIX-EXCEPTION-TEXT.  
03 ERROR-TEXT                PICTURE X(196).  
03 ERROR-TEXT-LEN           PICTURE 9(009) BINARY  
                             VALUE 196.
```

Parameter Passing of Object References in IDL Operations

Overview

The Orbix 6.2 COBOL runtime adheres to the memory management rules more strictly than the Orbix 2.3.x COBOL product.

Migration Impact

When migrating Orbix 2.3.x based applications using object references as operation parameters you are advised to refer to the *COBOL Programmer's Guide and Reference* for further details about memory management, paying particularly attention to when and where the `OBJDUP` and `OBJREL` APIs are called.

CORBA Object Location and Binding

Overview

This section summarizes the differences between Orbix 2.3.x object location mechanisms and Orbix 6.2 object location mechanisms.

In This Section

This section discusses the following topics:

Migration Overview and Example	page 140
The Naming Service	page 142
Object-String Conversion	page 144

Migration Overview and Example

In This Section

This subsection provides a migration overview for using `OBJSET` and an example of the differences.

This subsection discusses the following topics:

- [Migration Impact](#)
- [Migration Impact](#)
- [Orbix 6.2 and OBJSET](#)
- [Orbix 2.3.x Object Location Mechanism Example](#)

Migration Impact

Calls to the `OBJSET` API which rely on a fabricated object reference are illegal in Orbix 6.2. This API has been deprecated. The recommended replacement API is `STRTOOBJ` (as specified in the COBOL OMG specification).

Orbix 2.3.x and OBJSET

One way to locate an object in an Orbix 2.3.x application is to use the `OBJSET` API (equivalent to `_bind()` in C++), with a fabricated object reference constructed from the host name and server name in an Orbix object key, and the port information in the daemon. The daemon uses this information to locate (and activate if requested) the correct server. The server can then use the marker to locate the correct object.

Orbix 6.2 and OBJSET

If the application is calling `OBJSET` with a fabricated object reference (the application can still use it with an IOR or `corbaloc`) it must be replaced with one of the following object location mechanisms:

- Naming service (batch only), see [“The Naming Service” on page 142](#).
- Object-string conversion, see [“Object-String Conversion” on page 144](#).
- Calls to `OBJRIR` (batch only), see the *COBOL Programmer’s Guide and Reference*.

All these alternatives are based on the use of CORBA standard interoperable object references (IORs), the difference being in where the IORs are stored and how they are retrieved by the client application.

Orbix 2.3.x Object Location Mechanism Example

Example of the Orbix 2.3.x Object Location Mechanism:

```
MOVE SPACES TO WS-STRING-OBJ-REF
STRING ":\ "
    OR-HOST DELIMITED BY SPACE
    " ; "
    OR-SERVER DELIMITED BY SPACE
    " ; "
    OR-MARKER DELIMITED BY SPACE
    " ; "
    OR-IR DELIMITED BY SPACE
    " ; "
    OR-IRSRVR DELIMITED BY SPACE
    " ; "
    OR-INTF DELIMITED BY SPACE
INTO WS-STRING-OBJ-REF
END-STRING

DISPLAY "OBJECT REFERENCE = ' " WS-STRING-OBJ-REF " '"
CALL "OBJSET" USING WS-STRING-OBJ-REF
                    SERVER-OBJ
```

The Naming Service

Overview

The Naming Service is easy to understand and use if the application's naming graph is not too complex. The triplet of *markerName*, *serverName*, *hostName* used by the OBJSET API to locate an object, is replaced by a simple *name* in the Naming Service.

This subsection discusses the following topics:

- [Access to the Naming Service](#)
 - [Resolving Object Names](#)
 - [URL Syntax and IOR Configuration](#)
-

Access to the Naming Service

All applications should use the interoperable Naming Service, which provides access to future Naming Service implementations.

Access to the Naming Service can easily be wrapped. The only potential drawback in using the Naming Service is that it might become a single point of failure or performance bottleneck. If you use the Naming Service only to retrieve initial object references, these problems are unlikely to arise.

Resolving Object Names

An object's name is an abstraction of the object location—the location details are stored in the Naming Service. Use the following steps to resolve the Object names:

Step	Action
1	Call OBJRIR with <code>NameService</code> as its argument. This obtains an initial reference to the Naming Service.
2	The client uses the Naming Service to resolve the names of CORBA objects and receives object references in return.

URL Syntax and IOR Configuration

The URL syntax that the Naming Service provides makes it easier to configure IORs—and is similar to `_bind()` by letting you specify host, port, and well known object key in readable format. An example of the syntax for both types is outlined as follows:

- Stringified IOR syntax example:

“IOR:004301EF100...”

- URL type IOR syntax example:

“corbaloc::1.2@myhost:3075/NamingService”

With the URL syntax, `corbaloc` is the protocol name, the IIOP version number is `1.2`, the host name is `myhost`, and the port number is `3075`.

Note: Orbix 6.2 requires you to register a stringified IOR against a well known key with the Orbix 6.2 locator, which centralizes the use of stringified IORs in a single place, and lets you widely distribute readable URLs for clients.

Object-String Conversion

Overview

This subsection describes the migration impact of passing a fabricated object string as its first parameter to `OBJSET`.

This subsection discusses the following topics:

- [Migration impact using OBJSET](#)
 - [CORBA-compliant string-object conversion functions](#)
-

Migration impact using OBJSET

If the application is passing a fabricated object string (equivalent to `_bind()` in C++) as its first parameter to `OBJSET`, this string must now be of one of the following formats:

- a stringified interoperable object reference (IOR).
- a `corbaloc` formatted URL string.
- an `itmfaloc` formatted URL string.

Refer to the `STRTOOBJ` API in the *COBOL Programmers Guide Reference* for more details.

CORBA-compliant string-object conversion functions

The COBOL runtime offers two CORBA-compliant conversion APIs:

- `STRTOOBJ`
- `OBJTOSTR`

API Migration Issues

In this Section

This section discusses the following topics:

Deprecated APIs	page 146
ORBEXEC and USER Exception parameters	page 147
ORBSTAT	page 148
ORBALLOC	page 149

Deprecated APIs

Deprecated and Replacement APIs

Table 9 lists the COBOL APIs that are deprecated in Orbix Mainframe 6.2. It also lists their replacements where appropriate:

Table 9: *Deprecated COBOL APIs and Their Replacements*

Deprecated APIs	Replacement APIs
OBJGET	<i>Not replaced</i>
ORBALLOC	MEMALLOC
ORBREGO	ORBREG + OBJNEW
ORBFREE	MEMFREE
STRSETSP	STRSETP
OBJGETM	OBJGETID
OBJSETM	OBJNEW
OBJGETI	OBJTOSTR
OBJSET	STRTOOBJ
ORBGET	COAGET
ORBINIT	COARUN
ORBPUT	COAPUT
ORBREQ	COAREQ

Refer to the *COBOL Programmer's Guide and Reference* for full details of all the COBOL APIs supported.

ORBEXEC and USER Exception parameters

Overview

The `ORBEXEC` API function takes an extra parameter in Orbix 6.2.

This subsection discusses the following topics:

- [ORBEXEC in Orbix 2.3.x](#)
 - [ORBEXEC in Orbix 6.2](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

ORBEXEC in Orbix 2.3.x

The `ORBEXEC` API function in Orbix 2.3.x takes three parameters.

ORBEXEC in Orbix 6.2

The `ORBEXEC` API function in Orbix 6.2 takes four parameters instead of three. The fourth parameter is the user exception identifier.

Migration Impact

Any existing application code that calls `ORBEXEC` must be modified to include this extra parameter (the COBOL compiler does not check the number of parameters that are passed to `ORBEXEC`).

For any IDL that contains no user exception definitions, a dummy exception block is generated by the IDL compiler. The user exception block defined as a level 01 generated by the IDL compiler is then passed as the fourth parameter to `ORBEXEC`. This change has been introduced to support user exceptions in the COBOL runtime.

Refer to the *COBOL Programmer's Guide and Reference* for further details about the parameters of `ORBEXEC`.

In Summary

Affects COBOL clients only. Requires minor code change.

ORBSTAT

Overview

The ORBSTAT API is not optional in Orbix 6.2.

This subsection discusses the following topics:

- [ORBSTAT Functionality](#)
 - [Orbix 2.3.x and ORBSTAT](#)
 - [Orbix 6.2 and ORBSTAT](#)
 - [Migration Impact](#)
 - [Workaround](#)
-

ORBSTAT Functionality

The ORBSTAT API is used to register the ORBIX-STATUS-INFORMATION block with the COBOL runtime. This level 01 structure (ORBIX-STATUS-INFORMATION) is defined in the CORBA supplied copybook and allows the runtime to report exceptions.

Orbix 2.3.x and ORBSTAT

In Orbix 2.3.x, if ORBSTAT is not called and when the COBOL runtime encountered a system exception the program just ignores the exception

Orbix 6.2 and ORBSTAT

When the Orbix 6.2 COBOL runtime encounters a system exception and the ORBIX-STATUS-INFORMATION block is not registered with the runtime, the program terminates with the error below.

Migration Impact

This change only affects applications that don't already call the ORBSTAT API, and that encounter a runtime exception. When this happens the COBOL runtime outputs the following message and exits completely:

```
An exception has occurred but ORBSTAT has not been called. Place
the ORBSTAT API call in your application, compile and rerun.
Exiting now.
```

Workaround

To workaround this problem perform the following steps:

1. Place the ORBSTAT API call in your application.
2. Compile and run the application.

ORBALLOC

Overview

The Orbix 6.2 IDL Compiler has changed the mapping for IDL data types, `long`, `unsigned long`, `short` and `unsigned short`. These changes might effect the use of the deprecated `ORBALLOC` API.

This subsection discusses the following topics:

- [Mapping Changes](#)
- [Reason for Mapping Changes](#)
- [Migration Impact](#)
- [Workaround](#)

Mapping Changes

The following table represents the changes to the Working Storage data item definitions for the appropriate IDL data types:

Table 10: *ORBALLOC and Mapping Changes for IDL Data Types*

IDL Data Type	Orbix 6.2 IDL Compiler Output	gencbl Output
<code>long</code>	<code>S9(10) BINARY</code>	<code>S9(09) BINARY</code>
<code>unsigned long</code>	<code>9(10) BINARY</code>	<code>9(09) BINARY</code>
<code>short</code>	<code>S9(5) BINARY</code>	<code>S9(4) BINARY</code>
<code>unsigned short</code>	<code>9(5) BINARY</code>	<code>9(4) BINARY</code>

Reason for Mapping Changes

The mappings have been changed so that the COBOL runtime can marshal the complete range of values for `CORBA::Long`, `CORBA::ULong`, `CORBA::Short`, and `CORBA::UShort` respectively.

Migration Impact

The migration impact affects applications that call the deprecated `ORBALLOC` API, which allocates the specified number of bytes at runtime, if the type(s) `ORBALLOC` is allocating memory for contains one of more of the following: `9(10)BINARY`, `9(5)BINARY`, `S9(10)BINARY` or `S9(05)BINARY` and the exact memory requirements are specified.

Workaround

There are two scenarios for dealing with this, these are:

- If the application is using sequences, determine if the deprecated `ORBALLOC` API is being called, if so, use the `SEQALLOC` API in place of it.
- Determine if the deprecated `ORBALLOC` API is being called, and if so, increase the memory to be allocated to the Working Storage data items by the appropriate amount.

COBOL IMS Server Migration Issues

Overview

This section describes the source code changes required when migrating COBOL IMS Orbix 2.3.x servers to COBOL IMS Orbix 6.2 servers.

Note: This section must be read in conjunction with the other COBOL migration issues outlined in this document.

In This Section

This section discusses the following topics:

Server Mainline Program Requirement for IMS Servers	page 152
The Linkage Section for IMS Servers	page 156
Access to the Program Communication Block for IMS Servers	page 162
Error Checking Generation at Runtime for IMS Servers	page 164

Server Mainline Program Requirement for IMS Servers

Overview

A server mainline program is required for all IMS COBOL server programs running in an Orbix Mainframe 6.2 application.

This subsection discusses the following topics:

- [Migration Impact](#)
- [Migration Sample IDL](#)
- [Server Mainline for the Simple IDL](#)

Migration Impact

The migration impact is that every Orbix 2.3.x IMS COBOL server now requires a server mainline to run inside IMS. The server mainline can be generated by running the Orbix 6.2 IDL COBOL compiler and specifying the `:-S:-TIMS` compiler arguments.

Refer to the *COBOL Programmer's Guide and Reference* for more details of compiler arguments.

Migration Sample IDL

Consider the following IDL, called `simple`,

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

Server Mainline for the Simple IDL

The compiler output for the Orbix 6.2 IDL compiler produces two files for the `simple` IDL: a server implementation called `SIMPLES` and a server mainline called `SIMPLESV`. The following is the server mainline source code for IMS, `SIMPLESV`, produced by the Orbix 6.2 IDL compiler when the compiler arguments `:-S:-TIMS` are specified.

Note: The server implementation is generated in IMS only if the `:-Z:-TIMS` arguments are used with the Orbix 6.2 IDL compiler.

Example 4: *Server Mainline for the simple IDL with the Orbix 6.2 IDL Compiler (Sheet 1 of 3)*

```

*****
*   Description:
*       This program is a IMS server mainline for interfaces
*       described in SIMPLE
*****
IDENTIFICATION DIVISION.
PROGRAM-ID.          SIMPLESV.
ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.

COPY SIMPLE.
COPY CORBA.
COPY WSIMSPCB.

01 ARG-LIST                                PICTURE X(01)
                                           VALUE SPACES.
01 ARG-LIST-LEN                            PICTURE 9(09) BINARY
                                           VALUE 0.
01 ORB-NAME                                PICTURE X(10)
                                           VALUE
                                           "simple_orb".
01 ORB-NAME-LEN                            PICTURE 9(09) BINARY
                                           VALUE 10.
01 SERVER-NAME                             PICTURE X(07)
                                           VALUE
                                           "simple ".
01 SERVER-NAME-LEN                         PICTURE 9(09) BINARY
                                           VALUE 6.
01 INTERFACE-LIST.
   03 FILLER                               PICTURE X(28)
                                           VALUE
                                           "IDL:Simple/SimpleObject:1.0 ".
01 INTERFACE-NAMES-ARRAY REDEFINES INTERFACE-LIST.
   03 INTERFACE-NAME OCCURS 1 TIMES PICTURE X(28).
01 OBJECT-ID-LIST.
   03 FILLER                               PICTURE X(27)
                                           VALUE
                                           "Simple/SimpleObject_object ".
01 OBJECT-ID-ARRAY REDEFINES OBJECT-ID-LIST.
   03 OBJECT-IDENTIFIER OCCURS 1 TIMES PICTURE X(27).

```

Example 4: *Server Mainline for the simple IDL with the Orbix 6.2 IDL Compiler (Sheet 2 of 3)*

```

*****
* Object values for the Interface(s)
*****
01 SIMPLE-SIMPLEOBJECT-OBJ          POINTER
                                     VALUE NULL.

COPY LSIMSPCB.

PROCEDURE DIVISION USING LS-IO-PCB, LS-ALT-PCB.

INIT.
    PERFORM UPDATE-WS-PCBS.

    CALL "ORBSTAT" USING ORBIX-STATUS-INFORMATION.
    SET WS-ORBSTAT TO TRUE.
    PERFORM CHECK-STATUS.

    CALL "ORBARGS" USING ARG-LIST
        ARG-LIST-LEN
        ORB-NAME
        ORB-NAME-LEN.
    SET WS-ORBARGS TO TRUE.
    PERFORM CHECK-STATUS.

    CALL "ORBSRVR" USING SERVER-NAME
        SERVER-NAME-LEN.
    SET WS-ORBSRVR TO TRUE.
    PERFORM CHECK-STATUS.

```

Example 4: *Server Mainline for the simple IDL with the Orbix 6.2 IDL Compiler (Sheet 3 of 3)*

```

*****
* Interface Section Block
*****

* Generating Object Reference for interface Simple/SimpleObject
  CALL "ORBREG" USING SIMPLE-SIMPLEOBJECT-INTERFACE.
  SET WS-ORBREG TO TRUE.
  PERFORM CHECK-STATUS.

  CALL "OBJNEW" USING SERVER-NAME
    INTERFACE-NAME OF INTERFACE-NAMES-ARRAY(1)
    OBJECT-IDENTIFIER OF OBJECT-ID-ARRAY(1)
    SIMPLE-SIMPLEOBJECT-OBJ.
  SET WS-OBJNEW TO TRUE.
  PERFORM CHECK-STATUS.

  CALL "COARUN".
  SET WS-COARUN TO TRUE.
  PERFORM CHECK-STATUS.
  CALL "OBJREL" USING SIMPLE-SIMPLEOBJECT-OBJ.
  SET WS-OBJREL TO TRUE.
  PERFORM CHECK-STATUS.
  EXIT-PRG.
  GOBACK.

*****
* Populate the working storage PCB definitions
*****
  COPY UPDTPCBS.

*****
* Check Errors Copybook
*****
  COPY CERRSMFA.

```

The Linkage Section for IMS Servers

Overview

This subsection describes the differences between an Orbix 2.3.x IMS COBOL server and an Orbix 6.2 IMS COBOL server with regard to how the program communication block is exposed to Orbix applications.

This subsection discusses the following topics:

- [Migration Impact](#)
 - [Orbix 2.3.x Server Implementation for Simple IDL](#)
 - [Orbix 6.2 Server Implementation for Simple IDL](#)
 - [Linkage Section Migration](#)
-

Migration Impact

The linkage section of an Orbix 2.3.x server implementation must be removed.

Orbix 2.3.x Server Implementation for Simple IDL

The server implementation for the Orbix 2.3.x Compiler output for the `simple` IDL is as follows:

Example 5: *Orbix 2.3.x Compiler Output for the Simple IDL (Sheet 1 of 3)*

```
*****
* Identification Division
*****
IDENTIFICATION DIVISION.
PROGRAM-ID.          SIMPLES.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY SIMPLE.
COPY CORBA.

01 WS-INTERFACE-NAME                PICTURE X(30).
01 WS-INTERFACE-NAME-LENGTH         PICTURE 9(09) BINARY
                                     VALUE 30.
01 WS-ERROR-FUNC                    PICTURE X(09)
                                     VALUE SPACES.
```

Example 5: *Orbix 2.3.x Compiler Output for the Simple IDL (Sheet 2 of 3)*

```

LINKAGE SECTION.
*
** IMS linkage section data items
*
01 IOPCB.
    02 LTERM-NAME    PIC X(8).
    02 FILLER        PIC X(2).
    02 IOSTATUS      PIC XX.
    02 FILLER        PIC X(20).
01 DBPCB.
    02 DBNAME        PIC X(8).
    02 SEG-LEVEL-NO PIC X(2).
    02 DBSTATUS      PIC XX.
    02 FILLER        PIC X(20).
01 ALTPCB.
    02 DEST-TRAN     PIC X(8).
    02 FILLER        PIC X(2).
    02 ALTSTATUS     PIC XX.
    02 FILLER        PIC X(20).

*****
* Procedure Division
*****
PROCEDURE DIVISION USING IOPCB ALTPCB DBPCB.

    ENTRY "DISPATCH".
    CALL "ORBSTAT" USING ORBIX-STATUS-INFORMATION.
    MOVE "ORBSTAT" TO WS-ERROR-FUNC.
    PERFORM CHECK-STATUS.
    CALL "ORBREQ"   USING REQUEST-INFO.
    MOVE "ORBREQ" TO WS-ERROR-FUNC.
    PERFORM CHECK-STATUS.

* Resolve the pointer reference to the interface name which is
* the fully scoped interface name

    CALL "STRGET"   USING INTERFACE-NAME
                                WS-INTERFACE-NAME-LENGTH
                                WS-INTERFACE-NAME.

    SET WS-STRGET TO TRUE.
    PERFORM CHECK-STATUS.

```

Example 5: *Orbix 2.3.x Compiler Output for the Simple IDL (Sheet 3 of 3)*

```

*****
* Interface(s) evaluation:
*****
      MOVE SPACES TO SIMPLE-SIMPLEOBJECT-OPERATION.

      EVALUATE WS-INTERFACE-NAME
      WHEN 'Simple/SimpleObject'

* Resolve the pointer reference to the operation information
      CALL "STRGET" USING OPERATION-NAME
                          SIMPLE-S-3497-OPERATION-LENGTH
                          SIMPLE-SIMPLEOBJECT-OPERATION
      MOVE "STRGET" TO WS-ERROR-FUNC
      PERFORM CHECK-STATUS
      DISPLAY  "Simple::" SIMPLE-SIMPLEOBJECT-OPERATION
              "invoked"
      END-EVALUATE.

COPY SIMPLED.

      GOBACK.

DO-SIMPLE-SIMPLEOBJECT-CALL-ME.
      CALL "ORBGET"      USING SIMPLE-SIMPLEOBJECT-70FE-ARGS.
      MOVE "ORBGET" TO WS-ERROR-FUNC.
      PERFORM CHECK-STATUS.

      CALL "ORBPUT"      USING SIMPLE-SIMPLEOBJECT-70FE-ARGS.
      MOVE "ORBPUT" TO WS-ERROR-FUNC.
      PERFORM CHECK-STATUS.

*****
* Check Errors Section
*****
CHECK-STATUS SECTION.

      IF EXCEPTION-NUMBER NOT EQUAL 0 THEN
      DISPLAY "Server Impl: Call Failed in " WS-ERROR-FUNC
      DISPLAY "Server Impl: Exception Value is "
      EXCEPTION-NUMBER
      GOBACK
      END-IF.

```

Orbix 6.2 Server Implementation for Simple IDL

The following is the server implementation compiler output, `SIMPLES`, for the Orbix 6.2 IDL compiler:

Example 6: *Orbix 6.2 Server Implementation Code for Simple IDL (Sheet 1 of 2)*

```
*****
* Identification Division
*****
IDENTIFICATION DIVISION.
PROGRAM-ID.          SIMPLES.

ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY SIMPLE.
COPY CORBA.
COPY WSIMSPCB.

01 WS-INTERFACE-NAME          PICTURE X(30).
01 WS-INTERFACE-NAME-LENGTH  PICTURE 9(09) BINARY
                              VALUE 30.

*****
* Procedure Division
*****
PROCEDURE DIVISION.

    ENTRY "DISPATCH".

    CALL "COAREQ"    USING REQUEST-INFO.
    SET WS-COAREQ TO TRUE.
    PERFORM CHECK-STATUS.

* Resolve the pointer reference to the interface name which is
* the fully scoped interface name

    CALL "STRGET"    USING INTERFACE-NAME
                              WS-INTERFACE-NAME-LENGTH
                              WS-INTERFACE-NAME.

    SET WS-STRGET TO TRUE.
    PERFORM CHECK-STATUS.
```

Example 6: *Orbix 6.2 Server Implementation Code for Simple IDL (Sheet 2 of 2)*

```

*****
* Interface(s) evaluation:
*****
      MOVE SPACES TO SIMPLE-SIMPLEOBJECT-OPERATION.

      EVALUATE WS-INTERFACE-NAME
      WHEN 'IDL:Simple/SimpleObject:1.0'

* Resolve the pointer reference to the operation information
      CALL "STRGET" USING OPERATION-NAME
                          SIMPLE-S-3497-OPERATION-LENGTH
                          SIMPLE-SIMPLEOBJECT-OPERATION

      SET WS-STRGET TO TRUE
      PERFORM CHECK-STATUS
      DISPLAY  "Simple::" SIMPLE-SIMPLEOBJECT-OPERATION
              "invoked"
      END-EVALUATE.

COPY SIMPLED.

      GOBACK.

DO-SIMPLE-SIMPLEOBJECT-CALL-ME.
      CALL "COAGET"   USING SIMPLE-SIMPLEOBJECT-70FE-ARGS.
      SET WS-COAGET TO TRUE.
      PERFORM CHECK-STATUS.

      CALL "COAPUT"   USING SIMPLE-SIMPLEOBJECT-70FE-ARGS.
      SET WS-COAPUT TO TRUE.
      PERFORM CHECK-STATUS.

*****
* Check Errors Copybook
*****
      COPY CERRSMFA.

```

Linkage Section Migration

The linkage section in the Orbix 2.3.x compiler output which is highlighted in the [“Orbix 2.3.x Server Implementation for Simple IDL” on page 156](#) must be omitted altogether. The Orbix 6.2 IDL compiler produces a linkage section in the server mainline which appears as, `COPY LSIMSPCB`. The copybook `LSIMSPCB` is of the format:

LINKAGE SECTION.

01 LS-IO-PCB.

03 LS-IOPCB-LTERM-NAME	PICTURE X(8).
03 LS-IOPCB-DLI-RESERVE	PICTURE X(2).
03 LS-IOPCB-STATUS-CODE	PICTURE X(2).
03 LS-IOPCB-IN-PREFIX.	
05 LS-IOPCB-JULIAN-DATE	PICTURE S9(7) COMP-3.
05 LS-IOPCB-PCB-TIME-OF-DAY	PICTURE S9(7) COMP-3.
05 LS-IOPCB-MSG-SEQ	PICTURE S9(7) COMP.
03 LS-IOPCB-MOD-NAME	PICTURE X(8).
03 LS-IOPCB-RACF-ID	PICTURE X(8).

01 LS-ALT-PCB.

03 LS-ALTPCB-DEST-NAME	PICTURE X(8).
03 LS-ALTPCB-RESERVED	PICTURE X(2).
03 LS-ALTPCB-STATUS-CODE	PICTURE X(2).

Access to the Program Communication Block for IMS Servers

Overview

Orbix 2.3.x compiler generated code exposes the program communication block in the server implementation. Orbix 6.2 IDL compiler generated code exposes the program communication block in the server mainline. This data is accessible from the Orbix 6.2 server implementation by using the supplied `WSIMSPCB` and `UPDTPCBS` copybooks.

This subsection discusses the following topics:

- [Orbix 6.2 Server Mainline Code](#)
- [The copybook `WSIMSPCB` Format](#)
- [The copybook `UPDTPCBS` Format](#)

Orbix 6.2 Server Mainline Code

The server mainline generated by the Orbix 6.2 IDL compiler allows access to the program communication block data by populating the corresponding working storage data from the linkage section definitions using the paragraph `UPDATE-WS-PCBS`. The Working Storage data is defined in the `WSIMSPCB` copybook, the Linkage Section definitions are defined in the `LSIMSPCB` copybook and the `UPDATE-WS-PCBS` paragraph is defined in the `UPDTPCBS` copybook. These three copybooks are shipped with the product in `orbixhlq.INCLUDE.COPYLIB`.

For example, consider “[Server Mainline for the Simple IDL](#)” on page 152, the working storage section contains `COPY WSIMSPCB` which is populated from `LSIMSPCB` using the `UPDATE-WS-PCBS` paragraph defined in `UPDTPCBS`.

Note: If the server implementation requires access to the program communication block data it must have a copy statement for the copybook `WSIMSPCB` in its working storage section.

The copybook WSIMSPCB Format

The copybook WSIMSPCB has the format:

```
* Name: WSIMSPCB*
*****
** Program communication data area for use in COBOL IMS server.*

01 WS-IO-PCB IS EXTERNAL.
   03 WS-IOPCB-LTERM-NAME          PICTURE X(8).
   03 WS-IOPCB-DLI-RESERVE        PICTURE X(2).
   03 WS-IOPCB-STATUS-CODE        PICTURE X(2).
   03 WS-IOPCB-IN-PREFIX.
     05 WS-IOPCB-JULIAN-DATE      PICTURE S9(7) COMP-3.
     05 WS-IOPCB-PCB-TIME-OF-DAY PICTURE S9(7) COMP-3.
     05 WS-IOPCB-MSG-SEQ          PICTURE S9(7) COMP.
   03 WS-IOPCB-MOD-NAME           PICTURE X(8).
   03 WS-IOPCB-RACF-ID            PICTURE X(8).
01 WS-ALT-PCB IS EXTERNAL.
   03 WS-ALTPCB-DEST-NAME         PICTURE X(8).
   03 WS-ALTPCB-RESERVED          PICTURE X(2).
   03 WS-ALTPCB-STATUS-CODE      PICTURE X(2).
```

The copybook UPDTPCBS Format

The copybook UPDTPCBS is of the format:

```
* Name: UPDTPCBS*
*****
*
* The following is used to move the PCB linkage-section defined
* data to the corresponding working-storage definitions for use
* in the server implementaion.
*
UPDATE-WS-PCBS.
  MOVE LS-IOPCB-LTERM-NAME      TO WS-IOPCB-LTERM-NAME.
  MOVE LS-IOPCB-DLI-RESERVE     TO WS-IOPCB-DLI-RESERVE.
  MOVE LS-IOPCB-STATUS-CODE     TO WS-IOPCB-STATUS-CODE.
  MOVE LS-IOPCB-JULIAN-DATE     TO WS-IOPCB-JULIAN-DATE.
  MOVE LS-IOPCB-PCB-TIME-OF-DAY TO
                                WS-IOPCB-PCB-TIME-OF-DAY.
  MOVE LS-IOPCB-MSG-SEQ         TO WS-IOPCB-MSG-SEQ.
  MOVE LS-IOPCB-MOD-NAME        TO WS-IOPCB-MOD-NAME.
  MOVE LS-IOPCB-RACF-ID         TO WS-IOPCB-RACF-ID.
  MOVE LS-ALTPCB-DEST-NAME      TO WS-ALTPCB-DEST-NAME.
  MOVE LS-ALTPCB-RESERVED       TO WS-ALTPCB-RESERVED.
  MOVE LS-ALTPCB-STATUS-CODE    TO WS-ALTPCB-STATUS-CODE.
```

Error Checking Generation at Runtime for IMS Servers

Overview

This subsection summarizes the differences between `gencb1` and the Orbix 6.2 IDL Compiler in relation to the `CHECK-STATUS` paragraph used for error checking.

This subsection discusses the following topics:

- [The `gencb1` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

The `gencb1` Utility

The `CHECK-STATUS` paragraph is generated by `gencb1` for each server when it is run with the `-E` option.

The Orbix 6.2 IDL Compiler

The `CHECK-STATUS` paragraph is shipped as a static copybook called `CERRSMFA`, in the `orbixhlq.INCLUDE.COPYLIB` in Orbix 6.2. The reason that the Orbix 6.2 IDL Compiler doesn't generate this procedure is that, regardless of the IDL, the procedure code is unchanged.

Note: The `CHECK-STATUS` paragraph for IMS servers is different from batch in the following way: the `CHECK-STATUS` paragraph does not set the `RETURN-CODE` register, and calls `GOBACK` instead of `STOP RUN` if a system exception occurs.

Migration Impact

There is no migration impact, however IONA recommend you use the `CERRSMFA` copybook which shows the system exception encountered in a more user-friendly format.

COBOL IMS Client Migration Issues

Overview

This section describes the source code changes required when migrating COBOL IMS Orbix 2.3.x clients to COBOL IMS Orbix 6.2 clients.

Note: This section must be read in conjunction with the other COBOL migration issues outlined in this document.

In This Section

This section discusses the following topics:

The Linkage Section for IMS Clients	page 166
Error Checking Generation at Runtime for IMS Clients	page 168
Extra Copybooks in Orbix 6.2 for IMS Clients	page 169

The Linkage Section for IMS Clients

Overview

The linkage section in an Orbix 2.3.x IMS client implementation and the linkage section in an Orbix 6.2 IMS client implementation have different definitions.

This subsection discusses the following topics:

- [Migration impact](#)
 - [Orbix 2.3.x client implementation sample](#)
 - [Orbix 6.2 client implementation](#)
-

Migration impact

The linkage section of an Orbix 2.3.x client implementation must be replaced with `COPY LSIMSPCB`, and replace `PROCEDURE DIVISION USING IOPCB` with `PROCEDURE DIVISION USING LS-IO-PCB, LS-ALT-PCB`.

Orbix 2.3.x client implementation sample

The client implementation for the Orbix 2.3.x for the linkage section is as follows:

```
LINKAGE SECTION.  
  
01  IOPCB.  
    02  LTERM-NAME  PICTURE X(8).  
    02  FILLER      PICTURE X(2).  
    02  TPSTATUS    PICTURE XX.  
    02  FILLER      PICTURE X(20).  
  
PROCEDURE DIVISION USING IOPCB.
```

Orbix 6.2 client implementation

The client implementation for the Orbix 6.2 for the linkage section is as follows:

```
COPY LSIMSPCB.  
PROCEDURE DIVISION USING LS-IO-PCB, LS-ALT-PCB.
```

where the contents of `COPY LSIMSPCB` is:

```
LINKAGE SECTION.  
01 LS-IO-PCB.  
    03 LS-IOPCB-LTERM-NAME          PICTURE X(8).  
    03 LS-IOPCB-DLI-RESERVE         PICTURE X(2).  
    03 LS-IOPCB-STATUS-CODE        PICTURE X(2).  
    03 LS-IOPCB-IN-PREFIX.  
        05 LS-IOPCB-JULIAN-DATE     PICTURE S9(7) COMP-3.  
        05 LS-IOPCB-PCB-TIME-OF-DAY PICTURE S9(7) COMP-3.  
        05 LS-IOPCB-MSG-SEQ         PICTURE S9(7) COMP.  
    03 LS-IOPCB-MOD-NAME           PICTURE X(8).  
    03 LS-IOPCB-RACF-ID            PICTURE X(8).  
01 LS-ALT-PCB.  
    03 LS-ALTPCB-DEST-NAME         PICTURE X(8).  
    03 LS-ALTPCB-RESERVED          PICTURE X(2).  
    03 LS-ALTPCB-STATUS-CODE      PICTURE X(2).
```

Error Checking Generation at Runtime for IMS Clients

Overview

This subsection summarizes the differences between an Orbix 2.3.x client and an Orbix 6.2 client in relation to the `CHECK-STATUS` paragraph used for error checking.

This subsection discusses the following topics:

- [IMS clients in Orbix 2.3.x](#)
- [IMS clients in Orbix 6.2](#)
- [Migration impact](#)

IMS clients in Orbix 2.3.x

There is no copybook shipped for error-checking for IMS client code in Orbix 2.3.x. Customers are required to implement their own error checking procedure.

IMS clients in Orbix 6.2

For IMS clients a `CHKCLIMS` copybook is shipped in the `orbixhlg.INCLUDE.COPYLIB` in Orbix 6.2.

Note: The `CHECK-STATUS` paragraph for IMS clients is different from batch in the following way: the `CHECK-STATUS` paragraph does not set the `RETURN-CODE` register, and calls `GOBACK` instead of `STOP RUN` if a system exception occurs. It also writes a message to the IMS output message queue to show which API has failed.

Migration impact

There is no migration impact, however IONA recommend you use the `CHKCLIMS` copybook which shows the system exception encountered in a more user-friendly format.

Extra Copybooks in Orbix 6.2 for IMS Clients

Overview

This subsection describes differences in the code format between Orbix 2.3.x and Orbix 6.2 in regard to IMS clients.

This subsection discusses the following topics:

- [Migration impact](#)
 - [Orbix 6.2 IMS client code](#)
 - [Orbix 2.3.x IMS client code](#)
-

Migration impact

There is no migration impact. This subsection merely offers an explanation for why extra copybooks are shipped with Orbix 6.2 that are not shipped with Orbix 2.3.x.

The reason this code is shipped in copybooks in Orbix 6.2 is for ease of use and non-replication of code because it is common code for any IMS client.

Orbix 6.2 IMS client code

In Orbix 6.2 client code the following copy books are shipped:

Table 11: *Extra Copybooks that ship with Orbix 6.2*

Copybook	Description
WSIMSCL	This is relevant to IMS clients only. It contains a COBOL data definition that defines the format of the message that can be written by the paragraph contained in <code>orbixhlg.INCLUDE.COPYLIB(IMSWRITE)</code> . It also contains COBOL data definitions for calling the <code>GU</code> (get unique) and <code>ISRT</code> (insert) commands.
GETUNIQUE	This is relevant to IMS clients only. It contains a COBOL paragraph that can be called by the client, to retrieve specific IMS segments. It does this by using the supplied IBM routine (interface) <code>CBLTDLI</code> to make an IMS DC (data communications) call that specifies the <code>GU</code> (get unique) function command.

Table 11: *Extra Copybooks that ship with Orbix 6.2*

Copybook	Description
IMSWRITE	This is relevant to IMS clients only. It contains a COBOL paragraph called <code>WRITE-DC-TEXT</code> , to write a segment to the IMS output message queue. It does this by using the supplied IBM routine (interface) <code>CBLTDLI</code> to make an IMS DC (data communications) call that specifies the <code>ISRT</code> (insert) function command.

In Orbix 6.2 these copybooks are located in `orbixhlq.INCLUDE.COPYLIB`. This code is also included in the demonstrations.

Orbix 2.3.x IMS client code

For Orbix 2.3.x this code is part of the demonstration code for the Orbix 2.3.x demonstrations.

COBOL CICS Server Migration Issues

Overview

This section describes the source code changes required when migrating COBOL CICS Orbix 2.3.x servers to COBOL CICS Orbix 6.2 servers.

Note: This section must be read in conjunction with the other COBOL migration issues outlined in this document.

In This Section

This section discusses the following topics:

Server Mainline Program Requirement for CICS Servers	page 172
Access to the EXEC Interface Block Data Structure	page 176
Error Checking Generation at Runtime for CICS Servers	page 177

Server Mainline Program Requirement for CICS Servers

Overview

A server mainline program is required for all CICS COBOL programs running in an Orbix Mainframe 6.2 application.

This subsection discusses the following topics:

- [Migration Impact](#)
- [Migration Sample IDL](#)
- [Server Mainline for the Simple IDL](#)

Migration Impact

The migration impact is that every Orbix 2.3.x CICS COBOL server now requires a server mainline to run inside CICS. The server mainline can be generated by running the Orbix 6.2 IDL COBOL compiler and specifying the `:-S:-TCICS` compiler arguments.

Refer to the *COBOL Programmer's Guide and Reference* for more details of compiler arguments.

Migration Sample IDL

Consider the following IDL, called `simple`,

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

Server Mainline for the Simple IDL

The compiler output for the Orbix 6.2 IDL compiler produces two files for the `simple` IDL: a server implementation called `SIMPLES` and a server mainline called `SIMPLESV`. The following is the server mainline source code for CICS, `SIMPLESV`, produced by the Orbix 6.2 IDL compiler when the compiler arguments `:-S:-TCICS` are specified.

Note: The server implementation is generated in CICS only if the `:-Z:-TCICS` arguments are used with the Orbix 6.2 IDL compiler.

Example 7: Server Mainline for the simple IDL with the Orbix 6.2 IDL Compiler (Sheet 1 of 3)

```

*****
*   Description:
*       This program is a CICS server mainline for interfaces
*       described in SIMPLE
*****
IDENTIFICATION DIVISION.
PROGRAM-ID.          SIMPLSV.
ENVIRONMENT DIVISION.
DATA DIVISION.

WORKING-STORAGE SECTION.

COPY SIMPLE.
COPY CORBA.

01 ARG-LIST                      PICTURE X(01)
                                VALUE SPACES.
01 ARG-LIST-LEN                  PICTURE 9(09) BINARY
                                VALUE 0.
01 ORB-NAME                      PICTURE X(10)
                                VALUE
                                "simple_orb".
01 ORB-NAME-LEN                  PICTURE 9(09) BINARY
                                VALUE 10.
01 SERVER-NAME                   PICTURE X(07)
                                VALUE
                                "simple ".
01 SERVER-NAME-LEN               PICTURE 9(09) BINARY
                                VALUE 6.
01 INTERFACE-LIST.
    03 FILLER                     PICTURE X(28)
                                VALUE
                                "IDL:Simple/SimpleObject:1.0 ".
01 INTERFACE-NAMES-ARRAY REDEFINES INTERFACE-LIST.
    03 INTERFACE-NAME OCCURS 1 TIMES PICTURE X(28).
01 OBJECT-ID-LIST.
    03 FILLER                      PICTURE X(27)
                                VALUE
                                "Simple/SimpleObject_object ".
01 OBJECT-ID-ARRAY REDEFINES OBJECT-ID-LIST.
    03 OBJECT-IDENTIFIER OCCURS 1 TIMES PICTURE X(27).

```

Example 7: *Server Mainline for the simple IDL with the Orbix 6.2 IDL Compiler (Sheet 2 of 3)*

```

*****
* Object values for the Interface(s)
*****
01 SIMPLE-SIMPLEOBJECT-OBJ          POINTER
                                     VALUE NULL.

PROCEDURE DIVISION

    INIT.

        CALL "ORBSTAT" USING ORBIX-STATUS-INFORMATION.
        SET WS-ORBSTAT TO TRUE.
        PERFORM CHECK-STATUS.

        CALL "ORBARGS" USING ARG-LIST
            ARG-LIST-LEN
            ORB-NAME
            ORB-NAME-LEN.
        SET WS-ORBARGS TO TRUE.
        PERFORM CHECK-STATUS.

        CALL "ORBSRVR" USING SERVER-NAME
            SERVER-NAME-LEN.
        SET WS-ORBSRVR TO TRUE.
        PERFORM CHECK-STATUS.

```

Example 7: *Server Mainline for the simple IDL with the Orbix 6.2 IDL Compiler (Sheet 3 of 3)*

```

*****
* Interface Section Block
*****

* Generating Object Reference for interface Simple/SimpleObject
  CALL "ORBREG" USING SIMPLE-SIMPLEOBJECT-INTERFACE.
  SET WS-ORBREG TO TRUE.
  PERFORM CHECK-STATUS.

  CALL "OBJNEW" USING SERVER-NAME
    INTERFACE-NAME OF INTERFACE-NAMES-ARRAY(1)
    OBJECT-IDENTIFIER OF OBJECT-ID-ARRAY(1)
    SIMPLE-SIMPLEOBJECT-OBJ.
  SET WS-OBJNEW TO TRUE.
  PERFORM CHECK-STATUS.

  CALL "COARUN".
  SET WS-COARUN TO TRUE.
  PERFORM CHECK-STATUS.
  CALL "OBJREL" USING SIMPLE-SIMPLEOBJECT-OBJ.
  SET WS-OBJREL TO TRUE.
  PERFORM CHECK-STATUS.
  EXIT-PRG.
  GOBACK.

*****
* Check Errors Copybook
*****
COPY CERRSMFA.

```

Note: The batch implementation program is the same as the CICS implementation program except the CICS implementation program has a COPY CERRSMFA instead of a COPY CHKERRS

Access to the EXEC Interface Block Data Structure

Overview

This subsection describes the migration impact for CICS COBOL servers whose implementation requires access to the EXEC interface block (EIB) data structure. It discusses the following topics:

- “Migration Impact”
- “Required Code”

Migration Impact

Because Orbix 6.2 requires that all CICS COBOL servers have a server mainline, the implementation program is now a sub-program that is entered via a `DISPATCH` entry point. By default, the CICS program does not pass along the address of the EIB structure. As a result, you must add some additional code to your COBOL server implementation programs.

Required Code

In Working Storage, include the following `COPY` statement:

```
...  
COPY WSCICSSV  
...
```

Note: The WSCICSV contains the following line:

```
01 WS-EIB-POINTER          USAGE IS POINTER VALUE NULL.
```

At the start of your Procedure Division, after the `DISPATCH` entry point, add the following code:

```
EXEC CICS ADDRESS  
      EIB (WS-EIB-POINTER)  
      NOHANDLE  
END-EXEC.  
SET ADDRESS OF DFHEIBLK  
      TO WS-EIB-POINTER.
```

Error Checking Generation at Runtime for CICS Servers

Overview

This subsection summarizes the differences between `gencl` and the Orbix 6.2 IDL Compiler in relation to the `CHECK-STATUS` paragraph used for error checking.

This subsection discusses the following topics:

- [The `gencl` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)

The `gencl` Utility

The `CHECK-STATUS` paragraph is generated by `gencl` for each server when it is run with the `-E` option.

The Orbix 6.2 IDL Compiler

The `CHECK-STATUS` paragraph is shipped as a static copybook called `CERRSMFA`, in the `orbixhlq.INCLUDE.COPYLIB` in Orbix 6.2. The reason that the Orbix 6.2 IDL Compiler doesn't generate this procedure is that, regardless of the IDL, the procedure code is unchanged.

Note: The `CHECK-STATUS` paragraph for CICS servers is different from batch in the following way: the `CHECK-STATUS` paragraph does not set the `RETURN-CODE` register, and calls `GOBACK` instead of `STOP RUN` if a system exception occurs.

Migration Impact

There is no migration impact, however IONA recommend you use the `CERRSMFA` copybook which shows the system exception encountered in a more user-friendly format.

COBOL CICS Client Migration Issues

Overview

This section describes the source code changes required when migrating COBOL CICS Orbix 2.3.x clients to COBOL CICS Orbix 6.2 clients.

Note: This section must be read in conjunction with the other COBOL migration issues outlined in this document.

In This Section

This section discusses the following topics:

Error Checking Generation at Runtime for CICS Clients	page 179
Extra Copybooks in Orbix Mainframe 6.2	page 180

Error Checking Generation at Runtime for CICS Clients

Overview

This subsection summarizes the differences between an Orbix 2.3.x client and an Orbix 6.2 client in relation to the `CHECK-STATUS` paragraph used for error checking.

This subsection discusses the following topics:

- [CICS clients in Orbix 2.3.x](#)
- [CICS clients in Orbix 6.2](#)
- [Migration impact](#)

CICS clients in Orbix 2.3.x

There is no copybook shipped for error-checking for CICS client code in Orbix 2.3.x. Customers are required to implement their own error checking procedure.

CICS clients in Orbix 6.2

For CICS clients a `CHKCLCIC` copybook is shipped in the `orbixhlq.INCLUDE.COPYLIB` in Orbix 6.2.

Note: The `CHECK-STATUS` paragraph for CICS clients is different from batch in the following way: the `CHECK-STATUS` paragraph does not set the `RETURN-CODE` register, and calls `GOBACK` instead of `STOP RUN` if a system exception occurs. It also writes a message to the CICS terminal to show which API has failed.

Migration impact

There is no migration impact, however IONA recommend you use the `CHKCLCIC` copybook which shows the system exception encountered in a more user-friendly format.

Note: `CHKCLCIC` is relevant to CICS clients only. It contains a COBOL paragraph that has been translated by the CICS TS 1.3 translator. This paragraph can be called by the client, to check if a system exception has occurred and report it.

Extra Copybooks in Orbix Mainframe 6.2

Overview

This subsection describes differences in the code format between Orbix 2.3.x and Orbix 6.2.

This subsection discusses the following topics:

- [Migration impact](#)
- [Orbix 6.2 CICS client code](#)
- [Orbix 2.3.x CICS client code](#)

Migration impact

There is no migration impact. This subsection merely offers an explanation for why extra copybooks are shipped with Orbix 6.2 that are not shipped with Orbix 2.3.x.

The reason this code is shipped in copybooks in Orbix 6.2 is for ease of use and non-replication of code because it is common code for any CICS client.

Orbix 6.2 CICS client code

In Orbix 6.2 client code the following copy books are shipped:

Table 12: *Extra Copybooks that ship with Orbix 6.2*

Copybook	Description
WSCICSCL	This is relevant to CICS clients only. It contains a COBOL data definition that defines the format of the message that can be written by the paragraph contained in <code>orbixhlq.INCLUDE.COPYLIB(CICWRITE)</code> .
CICWRITE	This is relevant to CICS clients only. It contains a COBOL paragraph that has been translated by the CICS TS 1.3 translator. This paragraph can be called by the client, to write any messages raised by the supplied demonstrations to the CICS terminal.

In Orbix 6.2 these copybooks are located in `orbixhlq.INCLUDE.COPYLIB`. This code is also included in the demonstrations.

Orbix 2.3.x CICS client code

For Orbix 2.3.x this code is part of the demonstration code for the Orbix 2.3.x demonstrations.

Miscellaneous

In This Section

This section discusses miscellaneous migration issues.

This section discusses the following topics:

- [Interface Repository Server](#)
- [Command Line arguments](#)
- [DISPATCH Reference](#)

Interface Repository Server

In Orbix 2.3.x, `gencb1` requires the Interface Repository (IFR) server to be running to access the IDL source which is registered with the IFR server using `putidl`.

In Orbix 6.2, the IDL COBOL compiler accesses the IDL source directly, from the input IDL member (data set), and therefore does not need to access the IFR. Hence IDL members can be accessed independently (and IDL to COBOL development can proceed) without the need for any Orbix 6.2 services to be running.

Command Line arguments

The command-line arguments for the Orbix 6.2 IDL Compiler are different in some cases to the `gencb1` arguments. However, functionality common to both compilers can be achieved.

DISPATCH Reference

There is a minor code change in Orbix 6.2 for the `DISPATCH` reference used in Orbix 2.3.x. In Orbix 2.3.x, clients required the `DISPATCH` reference to compile and link a COBOL client with a COA. This reference is located in either of the following sections of code:

```
IDENTIFICATION DIVISION.
```

```
PROGRAM-ID. "DISPATCH" .
```

```
PROCEDURE DIVISION.
```

```
ENTRY "DISPATCH" .
```

In Orbix 6.2 this reference is not required. There is no migration impact in removing this reference.

PL/I Migration Issues

This chapter describes the issues involved in migrating PL/I applications from an Orbix 2.3-based IONA mainframe solution to Orbix Mainframe 6.2.

In this Chapter

This chapter discusses the following topics:

Fully Qualified Level 1 Data Names	page 185
Maximum Length of PL/I Data Names	page 188
IDL Constant Definitions Mapped to Fully Qualified Names	page 192
Typecode Name and Length Identifiers	page 195
Include Member names Based on the IDL Member name	page 196
Reserved PL/I Keywords for Module or Interface Names	page 203
Orbix PL/I Error Checking	page 204
CORBA Object Location and Binding	page 205
CORBA Include Member Additions	page 211
API Migration Issues	page 212
Server Accessor (Z Member)	page 216

PL/I IMS Server Migration Issues	page 222
PL/I IMS Client Migration issues	page 230
PL/I CICS Server Migration Issues	page 236
PL/I CICS Client Migration Issues	page 243
Miscellaneous	page 244

Fully Qualified Level 1 Data Names

Overview

This section summarizes the differences in the way that `genpli` and the Orbix 6.2 IDL Compiler generate level 01 data names.

This section discusses the following topics:

- [The `genpli` Utility and Data Names](#)
- [Orbix 6.2 IDL Compiler and Data Names](#)
- [Migration Impact](#)
- [Sample IDL](#)
- [The `genpli` Utility Output](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Workaround](#)
- [Using the `-M` Argument](#)
- [In Summary](#)

The `genpli` Utility and Data Names

The Orbix 2.3.x `genpli` utility by default uses only the local name as the generated data name. The `-L` and `-J` arguments are supplied with `genpli` to allow you to generate module-prefixed or interface-prefixed data names. In practice these arguments are seldom used by customers. Also, `genpli` can only support interfaces that are defined within a single module.

Orbix 6.2 IDL Compiler and Data Names

The Orbix 6.2 IDL Compiler replaces the `genpli` utility. The Orbix 6.2 IDL Compiler generates fully qualified names for PL/I level 01 data items. This means that it includes both module and interface names as prefixes in PL/I data names. It can therefore support any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces).

The ability of the Orbix 6.2 IDL Compiler to generate fully qualified names ensures the uniqueness of each generated name when, for example, the same operation name or attribute is used at a different scope within an IDL member.

Migration Impact

Orbix 6.2 IDL Compiler generates data names that are different from those generated by `genpli`, for example, if the `-J` and `-L` arguments are not supplied to generate PL/I code from a given interface, or if the generated name has to be truncated due to the PL/I restriction on the length of data names.

By default, the Orbix 6.2 IDL Compiler provides the same functionality as the `-L` and `-J` arguments provided with `genpli`. The `-M` argument provided with the Orbix 6.2 IDL Compiler can be used to generate code similar to that generated by `genpli` without the `-L` and `-J` arguments.

Sample IDL

Consider the following IDL for example:

```
//IDL
interface grid {
    void set(in short n, in short m, in long value);
};
```

The genpli Utility Output

The `genpli` utility generates the following PL/I code, based on the preceding IDL:

```
dcl 1 idl_set_type based,
    3 n                fixed bin(15)  init(0),
    3 m                fixed bin(15)  init(0),
    3 idl_value        fixed dec(8,2)  init(0);
```

Orbix 6.2 IDL Compiler Output

By contrast, the Orbix 6.2 IDL Compiler generates the following PL/I code, based on the preceding IDL:

```
dcl 1 grid_idl_set_type based,
    3 n                fixed bin(15)  init(0),
    3 m                fixed bin(15)  init(0),
    3 idl_value        fixed dec(8,2)  init(0);
```

Workaround

Use the `-M` argument that is provided with the Orbix 6.2 IDL Compiler to avoid having to make changes to your application source code. The `-M` argument allows you to generate a mapping member that you can then use

to map alternative names to your fully qualified data names. You can set these alternative names in the mapping member to be the same as the PL/I data names that are generated by `genpli`.

Using the -M Argument

You must run the Orbix 6.2 IDL Compiler twice with the `-M` argument. The first run generates the mapping member, complete with the fully qualified names and the alternative name mappings. Initially, the alternative name mappings are the same as the fully qualified names, so you must manually edit the mapping member to change the alternative names to the names that you want to use. Then run the `-M` argument again, this time to generate your PL/I include member complete with the alternative data names that you have set up in the specified mapping member.

Refer to the *PL/I Programmer's Guide and Reference* for an example of how to use the `-M` argument.

In Summary

Affects both clients and servers. Requires use of the described workaround or code changes.

Maximum Length of PL/I Data Names

Overview

This section summarizes the differences in the way that `genpli` and the Orbix 6.2 IDL Compiler process IDL identifier names that exceed 30 characters.

This section discusses the following topics:

- [The `genpli` Utility and long Data Names](#)
- [Problems with the `genpli` Method](#)
- [Orbix 6.2 IDL Compiler Solution](#)
- [Migration Impact](#)
- [Sample IDL](#)
- [The `genpli` utility Generated Data Names](#)
- [Orbix 6.2 IDL Compiler Generated Data Names](#)
- [In Summary](#)

The `genpli` Utility and long Data Names

Because `genpli` only supports the PL/I for MVS & VM compiler, a 31-character restriction is placed on the length of data names. The method used by `genpli` to generate data names for identifiers exceeding 31 characters is to truncate the identifier name to the first 27 characters and attaches a four-character numeric suffix, starting at 0000.

Problems with the `genpli` Method

This method is prone to problems if the original IDL for a completed application has to be subsequently modified, and the modifications involve IDL identifiers exceeding 31 characters being added mapped to member names. In such a case, the regenerated suffixes for the various data names do not match the original suffixes generated. This results in customers having to make undesirable source code changes.

Orbix 6.2 IDL Compiler Solution

To avoid this problem, the Orbix 6.2 IDL Compiler implements a new method. This new method ensures that the same suffix is always regenerated for a particular data name.

Migration Impact

The Orbix 6.2 IDL Compiler method generates completely different suffixes than the `genpli` suffixes for customer applications where such a scenario applies.

The following example illustrates these changes.

Sample IDL

Consider the following IDL:

```
// IDL
interface longname{
struct complex {
    long
        thisIsAreallyLongFeatureNamewithAnotherReallyLongFeatureExtensionAtTheEnd;
    long
        yetAnotherReallyLongFeatureNamewithAnotherReallyLongFeatureExtension;
    long
        ThirdLastYetAnotherReallyLongFeatureNamewithAnotherReallyLongFeatureExtension;
};
    void initialise();
    void op1(in complex ii);
    complex op2(in complex ii, inout complex io, out complex oo);
};
```

The genpli utility Generated Data Names

The `genpli` utility generates data names as follows based on the preceding IDL:

```
dcl 1 op1_type based,
  3 ii,
    5 thisIsAReallyLongFeatureNam0003    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeature0004    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyLo0005    fixed bin(31)  init(0);

dcl 1 op2_type based,
  3 ii,
    5 thisIsAReallyLongFeatureNam0006    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeature0007    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyLo0008    fixed bin(31)  init(0);

  3 io,
    5 thisIsAReallyLongFeatureNam0009    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeature0010    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyLo0011    fixed bin(31)  init(0);

  3 oo,
    5 thisIsAReallyLongFeatureNam0012    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeature0013    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyLo0014    fixed bin(31)  init(0);

  3 result,
    5 thisIsAReallyLongFeatureNam0015    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeature0016    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyLo0017    fixed bin(31)  init(0);
```

Orbix 6.2 IDL Compiler Generated Data Names

The Orbix 6.2 IDL Compiler generates data names as follows based on the preceding IDL:

```
dcl 1 longname_op1_type based,
  3 ii,
    5 thisIsAReallyLongFeatureNa_e658    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeatur_7628    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyL_e278    fixed bin(31)  init(0);

dcl 1 longname_op2_type based,
  3 ii,
    5 thisIsAReallyLongFeatureNa_e658    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeatur_7628    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyL_e278    fixed bin(31)  init(0);

  3 io,
    5 thisIsAReallyLongFeatureNa_e658    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeatur_7628    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyL_e278    fixed bin(31)  init(0);

  3 oo,
    5 thisIsAReallyLongFeatureNa_e658    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeatur_7628    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyL_e278    fixed bin(31)  init(0);

  3 result,
    5 thisIsAReallyLongFeatureNa_e658    fixed bin(31)  init(0),
    5 yetAnotherReallyLongFeatur_7628    fixed bin(31)  init(0),
    5 ThirdLastYetAnotherReallyL_e278    fixed bin(31)  init(0);
```

In Summary

Affects clients and servers where IDL identifiers exceed 31 characters.
Requires code changes.

IDL Constant Definitions Mapped to Fully Qualified Names

Overview

IDL constant definitions are mapped, in Orbix 6.2, to fully qualified data names, because the Orbix 6.2 IDL Compiler can process any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces). Therefore, the same constant names can be used at different scopes, and uniqueness of data names is imperative.

This section discusses the following topics:

- [IDL Output Comparison](#)
- [Migration Impact](#)
- [Sample IDL](#)
- [The Orbix 6.2 IDL Compiler Mapping for Constants](#)
- [Legacy Support](#)
- [In Summary](#)

IDL Output Comparison

[Table 13](#) lists the differences between the Orbix 6.2 IDL Compiler and the `genpli` mapping for IDL constant definitions:

Table 13: *PL/I Compiler Output for IDL Constant Definitions*

	Orbix 6.2 IDL Compiler	The <code>genpli</code> Utility
Global constant at IDL member level	<code>dcl 1 global_FQN_consts, 3 localname...</code>	<code>dcl 1 global_TEST_consts, 3 localname...</code>
Global constant at module level	<code>dcl 1 FQN_consts, 3 localname...</code>	<code>dcl 1 modulename_module_consts, 3 localname...</code>
Constant at interface level	<code>dcl 1 FQN_consts, 3 localname...</code>	<code>dcl 1 interfacename_consts, 3 localname...</code>

In the preceding example, *FQN* represents the fully qualified name for the module or interface where the constant is defined.

Migration Impact

The `module` keyword that is generated by `genpli` is not used in Orbix 6.2, because there is support for more than one level of module. With `genpli`, only one level of module is supported. .

Note: The `global` keyword is still used, but in the case of `genpli`, refers to all constant definitions defined in the Interface Repository. In the case of Orbix 6.2 it refers to all constants defined at global scope in the IDL member being processed.

Note: The Interface Repository server is not required by the Orbix 6.2 IDL Compiler when generating PL/I definitions from IDL. For further details refer to [“Interface Repository Server” on page 244](#).

Sample IDL

Consider the following IDL member, called `TEST`, which defines four constants with the same name—`myconstant`—at different levels:

```
//test.idl
const long myconstant = 1;
module m1
{
    const long myconstant = 1;
    interface fred
    {
        const long myconstant = 1;
        void myop();
    };
    module m2
    {
        interface fred
        {
            const long myconstant = 1;
            void myop();
        };
    };
};
```

The Orbix 6.2 IDL Compiler Mapping for Constants

The Orbix 6.2 IDL Compiler mapping for the constants results in the following data names:

```

/*-----*/
/* Constants in root scope: */
/*-----*/
dcl 1 global_TEST_consts ,
    3 myconstant                fixed bin(31)  init(1);

/*-----*/
/* Constants in m1: */
/*-----*/
dcl 1 m1_consts ,
    3 myconstant                fixed bin(31)  init(1);

/*-----*/
/* Constants in m1/fred: */
/*-----*/
dcl 1 m1_fred_consts ,
    3 myconstant                fixed bin(31)  init(1);

/*-----*/
/* Constants in m1/m2/fred: */
/*-----*/
dcl 1 m1_m2_fred_consts ,
    3 myconstant                fixed bin(31)  init(1);

```

Legacy Support

It is not feasible to provide full legacy support in this case. However, you can use the `-M` argument with the Orbix 6.2 IDL Compiler to control the *FQN* (Fully Qualified Name) shown in the preceding example. You can also use the `-O` argument with the Orbix 6.2 IDL Compiler to determine the name of the generated include member, which defaults to the IDL member name when it is first generated.

Refer to the *PL/I Programmer's Guide and Reference* for an example of how to use the `-M` and `-O` arguments.

In Summary

Affects clients and servers. Requires code changes where constants are used.

Typecode Name and Length Identifiers

Overview

This sections summarizes the different output for `genpli` and the Orbix 6.2 IDL Compiler for typecode and typecode length data names.

This section discusses the following topics:

- [The genpli Utility Output](#)
- [Orbix 6.2 IDL Compiler Output](#)
- [Migration Impact](#)

The genpli Utility Output

The typecodes and typecode length names generated by `genpli` used the names `interfacename_type` and `interfacename_type_length`. This is not suitable for a situation where an IDL member contains multiple nested levels of modules and interfaces, because unique data names can not be generated in this case.

Orbix 6.2 IDL Compiler Output

Because the Orbix 6.2 IDL Compiler can process any level of scoping in an IDL member (that is, multiple levels of nested modules and interfaces), the generated data names are of the form `idlmembername_type` and `idlmembername_type_length`. This ensures the uniqueness of the data names.

Migration Impact

However, this has a migration impact if either of the following apply:

- IDL member names are different from the interface names they contain.
- More than one interface is defined in an IDL member.

Refer to [“IDL Member names Different from Interface Names” on page 199](#) for details of the migration impact.

Refer to [“More than One Interface in an IDL Member” on page 201](#) for details of the migration impact.

Include Member names Based on the IDL Member name

Overview

Include member names in Orbix 6.2 are generated based on the IDL member name instead of being based on the interface name, as is the case with `genpli`. The reason for this change is because the Orbix 6.2 IDL Compiler can process any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces).

This section discusses the following topics:

- [The genpli Utility](#)
- [Orbix 6.2 IDL Compiler](#)
- [Sample IDL](#)
- [Problem with The genpli Utility](#)
- [Orbix 6.2 IDL Compiler Solution](#)
- [Migration Impact](#)

The genpli Utility

Include member names are generated based on the interface name with `genpli`.

Orbix 6.2 IDL Compiler

Include member names are generated based on the IDL member name. This is because the Orbix 6.2 IDL Compiler can process any level of scoping in IDL members (that is, multiple levels of nested modules and interfaces). Therefore, because the same interface name might be defined at different levels within the same IDL member, this renders it impossible to base include member names on interface names.

Sample IDL

For example, consider the following IDL member called `myidl`:

```
//myidl
module m1
{
    interface fred
    {
        void myop();
    };
    module m2
    {
        interface fred
        {
            void myop();
        };
    };
};
```

Problem with The `genpli` Utility

The `genpli` utility can not process correctly the preceding IDL, because it contains more than one level of module.

If the interface name is used to generate the include member name, it generates a set of PL/I include members for each interface defined. But because both interfaces share the same name, which is `fred` in the preceding example, the generation of one set of include members overwrites the other.

Orbix 6.2 IDL Compiler Solution

The Orbix 6.2 IDL Compiler generates PL/I include member names based on the IDL member name, which is `myidl` in the preceding example. Therefore, the definitions for all the interfaces contained within this IDL member are produced in the `myidl` include members. (This is also how the IDL compiler generates C++ and Java files.)

Migration Impact

This has a migration impact if either of the following apply:

- IDL member names are different from the interface names they contain.
- More than one interface is defined in an IDL member.

The migration impact for each of these situations is described in the following subsections;

Note: The Typecode and typecode length data name migration issue is very similar to the include member names based on interface and module name issue, hence these scenarios are dealt with in only one section.

IDL Member names Different from Interface Names

In This Section

This section discusses the following topics:

- [Sample IDL](#)
 - [Generated Include Member Name Comparison Table](#)
 - [Genpli Utility-Generated Include Member Names](#)
 - [Orbix 6.2 IDL Compiler-Generated Include Member Names](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

Sample IDL

Consider the following IDL member called `GRID`, which defines an interface called `fred`:

```
//grid.idl
interface fred
{
    void myop(in long mylong);
};
```

Generated Include Member Name Comparison Table

The preceding IDL member results in the following include members being generated:

Table 14: *PL/I Compiler Output Comparison GRID Include Member Names*

The genpli Utility	Orbix 6.2
FREDD	GRIDD
FREDM	GRIDM
FREDR	GRIDL
FREDT	GRIDT
FREDX	GRIDX

Genpli Utility-Generated Include Member Names

In the case of the `genpli` utility, the generated include Member names are based on the interface name, which is `fred` in the preceding example.

Orbix 6.2 IDL Compiler-Generated Include Member Names

In the case of the Orbix 6.2 IDL Compiler, the generated include member names are based on the IDL member name, which is `grid` in the preceding example.

Migration Impact

If your IDL member name is not the same as the interface name it contains you can use the `-o` argument with the Orbix 6.2 IDL Compiler to map the name of the generated PL/I include members (which, in Orbix 6.2, is based by default on the IDL member name) to an alternative name if your IDL member name is not the same as the interface names it contains. This means you can avoid having to change the `%include` statements (for example, from `%include FRED` to `%include GRID`) in your application source code.

Refer to the *PL/I Programmer's Guide and Reference* for an example of how to use the `-o` argument.

In Summary

Affects clients and servers. Requires minor code change or use of the described workaround.

More than One Interface in an IDL Member

In This Section

This section discusses the following topics:

- [The genpli Utility](#)
 - [Orbix 6.2 IDL Compiler](#)
 - [Sample IDL](#)
 - [IDL Output Comparison](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

The genpli Utility

The `genpli` utility generates a set of include members for each interface definition, and bases the name for each set of include members on the associated interface name.

Orbix 6.2 IDL Compiler

The Orbix 6.2 IDL Compiler generates only one set of include members for an IDL member, and it bases the name for that set of include members on the IDL member name. If an IDL member contains N interfaces (where N is greater than one), your existing application code now contains $N-1$ redundant `%include` statements.

Sample IDL

For example, consider the following IDL member, called `GRID`, which contains the two interfaces called `grid` and `block`:

```
// grid.idl
interface grid
{
    void sizeofgrid(in long mysize1, in long
                   mysize2);
};

interface block
{
    void area(in long myarea);
};
```

IDL Output Comparison

The differences in the way `genpli` and the Orbix 6.2 IDL Compiler process the preceding IDL can be outlined as follows:

Table 15: *PL/I Compiler Deprecated IDL Generated Members and Their Replacements*

The Orbix 6.2 IDL Compiler	The <code>genpli</code> utility
<p>Generates only one set of include members that contain all the definitions for all interfaces contained within the IDL member. The include member names are based on the IDL member name. For example:</p> <p>GRIDD GRIDL GRIDM GRIDT GRIDX</p>	<p>Generates a set of include members for each interface, based on each interface name. For example:</p> <p>GRIDD, BLOCKD GRIDR, BLOCKR GRIDM, BLOCKM GRIDT, BLOCKT GRIDX, BLOCKX</p>

Migration Impact

Based on the preceding example, the `BLOCK` include members are redundant with the Orbix 6.2 IDL Compiler. Therefore, the `%include` statements pertaining to these must be removed from the application code.

In Summary

Affects clients and servers. Requires minor code change.

Reserved PL/I Keywords for Module or Interface Names

Overview

This section illustrates the different ways that `genpli` and the Orbix 6.2 IDL Compiler treat PL/I keywords used as module or interface names.

Note: The Orbix 6.2 IDL compiler supports the PL/I-reserved words pertaining to the IBM PL/I for MVS & VM version 1.1.1 and Enterprise PL/I compilers.

This section discusses the following topics:

- [The genpli Utility](#)
- [Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)
- [In Summary](#)

The genpli Utility

If a reserved PL/I keyword is used as an IDL interface or module name, it is not treated as a reserved word by `genpli`.

Orbix 6.2 IDL Compiler

If a reserved PL/I keyword is used as an IDL interface or module name, it is treated as a reserved word by the Orbix 6.2 IDL Compiler.

Migration Impact

This has a migration impact for any customers that use reserved PL/I keywords as IDL interface or module names. If any customers are using reserved PL/I keywords, source code changes are required to their applications to cater for `IDL-` prefixed names that are generated for identifiers in Orbix 6.2.

In Summary

Affects clients and servers where module or interface names are reserved PL/I keywords. Requires code change or use of the workaround described in [“Fully Qualified Level 1 Data Names” on page 185](#) to resolve this issue down to the operation names level.

Orbix PL/I Error Checking

Overview

This section summarizes the difference between `genpli` and the Orbix 6.2 IDL Compiler in regard to the `CHECK_ERRORS` function.

This section discusses the following topics:

- [The `genpli` Utility](#)
- [The Orbix 6.2 IDL Compiler](#)
- [Migration Impact](#)
- [In Summary](#)

The `genpli` Utility

The PL/I `CHECK_ERRORS` function is generated by `genpli` for each server.

The Orbix 6.2 IDL Compiler

In Orbix 6.2, the member that contains the `CHECK_ERRORS` function is placed into a static member called `CHKERRS`.

Migration Impact

It is no longer necessary to generate an IDL-dependent member for error checking. If your implementation code contains a `%include interfacenameR;` statement, you must update it to read as `%include CHKERRS;` instead.

In Summary

Affects clients and servers. Requires minor code change.

CORBA Object Location and Binding

Overview

This section summarizes the differences between Orbix 2.3.x object location mechanisms and Orbix 6.2 object location mechanisms.

In This Section

This section discusses the following topics:

Migration Overview and Example	page 206
Naming Service	page 208
Object-String Conversion	page 210

Migration Overview and Example

In This Section

This section discusses the following topics:

- [Migration Impact](#)
 - [Orbix 2.3.x Object Location Mechanisms](#)
 - [Orbix 6.2 Object Location Mechanisms](#)
 - [Orbix 2.3.x Object Location Mechanism Example](#)
-

Migration Impact

Calls to the `OBJSET` API which rely on a fabricated object reference are illegal in Orbix 6.2. This API has been deprecated. The recommended replacement API is `STR2OBJ` (as specified in the PL/I OMG specification).

Orbix 2.3.x Object Location Mechanisms

One way to locate an object in an Orbix 2.3.x application is to use the API `OBJSET` (equivalent to `_bind()` in C++), with a fabricated object reference constructed from the host name and server name in an Orbix object key, and the port information in the daemon. The daemon uses this information to locate (and activate if requested) the correct server. The server can then use the marker to locate the correct object.

Note: The `OBJSET` API is deprecated and the recommended replacement API is `STR2OBJ` as specified by the OMG PL/I specification.

Orbix 6.2 Object Location Mechanisms

If the application is calling `OBJSET` with the fabricated object reference (the application can still use it with an IOR or `corbaloc`) it must be replaced it with one of the following object location mechanisms:

- Naming service (batch only), see [“Naming Service” on page 208](#).
- Object-string conversion, see [“Object-String Conversion” on page 210](#).
- Calls to `OBJRIR` (batch only), see the *PL/I Programmer’s Guide and Reference*.

All these alternatives are based on the use of CORBA standard interoperable object references (IORs), the difference being in where the IORs are stored and how they are retrieved by the client application.

Orbix 2.3.x Object Location Mechanism Example

Example of the Orbix 2.3.x Object Location Mechanism:

```
object_name=':\pluto:grid::IR:grid ' ;  
call objset(object_name,obj_ref);
```

Naming Service

Overview

The Naming Service is easy to understand and use if the application's naming graph is not too complex. The triplet of *markerName*, *serverName*, *hostName* used by the `OBJSET` API to locate an object, is replaced by a simple *name* in the Naming Service.

This section discusses the following topics:

- [Access to the Naming Service](#)
- [Resolving Object Names](#)
- [URL Syntax and IOR Configuration](#)

Access to the Naming Service

All applications should use the interoperable Naming Service, which provides access to future Naming Service implementations.

Access to the Naming Service can easily be wrapped. The only potential drawback in using the Naming Service is that it might become a single point of failure or performance bottleneck. If you use the Naming Service only to retrieve initial object references, these problems are unlikely to arise.

Resolving Object Names

An object's name is an abstraction of the object location — the location details are stored in the Naming Service. Use the following steps to resolve Object names:

Step	Action
1	Call <code>OBJRIR</code> with <code>NameService</code> as its argument. An initial reference to the Naming Service is obtained.
2	The client uses the Naming Service to resolve the names of CORBA objects and receives object references in return.

URL Syntax and IOR Configuration

The URL syntax that the interoperable Naming Service provides makes it easier to configure IORs—and is similar to `_bind()` by letting you specify host, port, and well known object key in readable format. An example of the syntax for both types is outlined as follows.

- Stringified IOR syntax example:

“IOR:004301EF100...”

- URL type IOR syntax example:

“corbaloc::1.2@myhost:3075/NamingService”

With the URL syntax, `corbaloc` is the protocol name, the IIOP version number is `1.2`, the host name is `myhost`, and the port number is `3075`.

Note: Orbix 6.2 requires you to register a stringified IOR against a well known key with the Orbix 6.2 locator, which centralizes the use of stringified IORs in a single place, and lets you widely distribute readable URLs for clients.

Object-String Conversion

In This Section

This section discusses the following topics:

- [Migration impact using OBJSET](#)
 - [CORBA-compliant String-object Conversion Functions](#)
-

Migration impact using OBJSET

If the application is passing a fabricated object string (equivalent to `_bind()` in C++) as its first parameter to `OBJSET`, this string must now be of one of the following formats:

- a stringified interoperable object reference (IOR).
- a `corbaloc` formatted URL string.
- an `itmfaloc` formatted URL string.

Refer to the `STRT2OBJ` API in the *PL/I Programmers Guide Reference* for more details.

CORBA-compliant String-object Conversion Functions

The PL/I runtime offers two CORBA-compliant string-object conversion APIs:

`STR2OBJ`

`OBJ2STR`

CORBA Include Member Additions

Overview

There have been several additions to the supplied CORBA include member. This section discusses the following topics:

- [Migration Impact](#)
- [Workaround](#)

Migration Impact

There is a possibility that some of the new identifiers might conflict with those defined in your application. For a complete list of identifiers, please refer to the supplied include members located in

`orbixhlq.INCLUDE.PLINCL(CORBA)`.

Workaround

It might be necessary to change some of your PL/I data names if they conflict with any of the new data names added to the PL/I CORBA include member.

API Migration Issues

In This Section

This section contains the following subsections:

Deprecated APIs	page 213
PODSTAT in Orbix 6.2	page 214
PODEXEC and User Exception parameters	page 215

Deprecated APIs

Deprecated and Replacement APIs

Table 16 provides a list of the PL/I APIs that are deprecated in Orbix Mainframe 6.2. In some cases, an API has been replaced with another. This is outlined, where applicable.

Table 16: *Deprecated PL/I APIs and Their Replacements*

Deprecated APIs	Replacement APIs
OBJGET	OBJ2STR
OBJGETM	OBJGTID
OBJGETO	<i>Not replaced</i>
OBJLEN	<i>Not replaced</i>
OBJLENO	<i>Not replaced</i>
OBJSET	STR2OBJ
OBJSETM	<i>Not replaced</i>
PODALOC	MEMALOC
PODEBUG	MEMDEBUG
PODEXEC (3 parameters)	PODEXEC (4 parameters)
PODFREE	MEMFREE
PODHOST	<i>Not Replaced</i>
PODINIT	PODRUN
PODRASS	PODERR
PODREGI	PODREG + OBJNEW

Refer to the *PL/I Programmer's Guide and Reference* for full details of all the PL/I APIs supported.

PODSTAT in Orbix 6.2

Overview

The `PODSTAT` API is not optional in Orbix 6.2.

This section discusses the following topics:

- [PODSTAT Functionality](#)
 - [Orbix 2.3.x and PODSTAT](#)
 - [Orbix 6.2 and PODSTAT](#)
 - [Migration Impact](#)
 - [Workaround](#)
-

PODSTAT Functionality

The `PODSTAT` API is used to register the `POD_STATUS_INFORMATION` block with the PL/I runtime. This structure (`POD_STATUS_INFORMATION`) is defined in the CORBA supplied include member and allows the runtime to report exceptions.

Orbix 2.3.x and PODSTAT

In Orbix 2.3.x, if `PODSTAT` is not called and the PL/I runtime encounters an exception, the runtime doesn't exit, but just ignores the exception.

Orbix 6.2 and PODSTAT

In Orbix 6.2, this is not the case. When the Orbix 6.2 PL/I runtime encounters an exception, and the `POD_STATUS_INFORMATION` block is not registered with the runtime, that is, the `PODSTAT` API is not called, the runtime exits.

Migration Impact

This change only affects applications that don't call the `PODSTAT` API, and that encounter a runtime. In this situation the PL/I runtime outputs the following message and exits completely:

```
An exception has occurred but PODSTAT has not been called. Place
the PODSTAT API call in your application, compile and rerun.
Exiting now.
```

Workaround

To workaround this problem perform the following steps:

1. Place the `PODSTAT` API call in your application.
2. Recompile and run the application.

PODEXEC and User Exception parameters

In This Section

This section discusses the following topics:

- [PODEXEC in Orbix 2.3.x](#)
 - [PODEXEC in Orbix 6.2](#)
 - [Migration Impact](#)
 - [In Summary](#)
-

PODEXEC in Orbix 2.3.x

The `PODEXEC` function in Orbix 2.3.x takes three parameters.

PODEXEC in Orbix 6.2

The `PODEXEC` function in Orbix 6.2 takes four parameters instead of three. The fourth parameter is the user exception identifier.

Migration Impact

Any existing application code that calls `PODEXEC` must be modified to include this extra parameter. This change has been introduced to comply with the OMG specification for `PODEXEC`.

For operations which do not have user expectations, the fourth parameter is `no_user_exceptions`.

For operations which can return a user exception, the fourth parameter is `addr(IFNAME_user_exceptions)` where `IFNAME` is the first six characters of your interface name (or the name specified by the `-o` argument in the IDL compiler if it is used).

In Summary

Affects PL/I clients only. Requires minor code change.

Server Accessor (Z Member)

In This Section

This section discusses the differences between the Orbix 2.3.x server implementation and the Orbix 6.2 server implementation in regard to the server accessor (Z member).

This section discusses the following topics:

- [Migration Impact](#)
 - [Migration Sample IDL](#)
 - [Orbix 2.3.x Compiler Output](#)
 - [Orbix 6.2 Compiler Output](#)
 - [Contents of the DISPINIT Member](#)
-

Migration Impact

For Orbix 6.2 applications, the server accessor is replaced. A new include member, `DISPINIT`, has been added to the server implementation (that is, the `idlmembernameI` member) to replace server accessor functionality. In Orbix 2.3.x applications, `genpli` generates the server accessor (that is, the `idlmembernameZ` member). The Orbix 6.2 IDL compiler does not generate an `idlmembernameZ` member. The `idlmembernameI` member is coded differently to the Orbix 2.3.x server implementation. These differences are:

- Every Orbix 6.2 server implementation requires this definition which must be placed after the procedure statement.:

```
DISPTCH: ENTRY;
```

- The Orbix 6.2 server implementation has no parameters.
- For Orbix 6.2 the operation declaration for operations has been moved into the `DISPINIT` member.

- For Orbix 6.2 a new include statement for the include member, `DISPINIT`, has been added to the server implementation. The `DISPINIT` member contains the core functionality of the server accessor, that is, the call to `PODREQ` and the extraction of the operation name, which is used by the select statement in the select include member.

Note: Customers who are manually editing Orbix 2.3.x server implementations when migrating to Orbix 6.2 need to be aware of the differences in the two implementations that are described in the preceding four bullet points.

Migration Sample IDL

Consider the following IDL, called `simple`,

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

Orbix 2.3.x Compiler Output

Server mainline output for the `simple` interface, `SIMPLEZ`, with the Orbix 2.3.x IDL compiler (for Batch) is as follows:

```
SIMPLEZ: PROC;

/*The following line enables the POD to link into this procedure*/
DISPTCH: ENTRY;

dcl operation                char(256)      init('');
dcl operation_length         fixed bin(31)  init(256);

dcl SIMPLEI                  ext entry(char(*));

dcl addr                     builtin;
dcl low                      builtin;
dcl sysnull                  builtin;

#include CORBA;
#include SIMPLER;

call podreq(reqinfo);
if check_errors('podreq') ^= completion_status_yes then return;

call strget(operation_name,
              operation,
              operation_length);
if check_errors('strget') ^= completion_status_yes then return;

call SIMPLEI(operation);

END SIMPLEZ;
```

Server implementation output for the `simple` interface, `SIMPLEI`, with the Orbix 2.3.x IDL compiler (for Batch and CICS) is as follows:

Note: The IMS server implementation is identical to batch and CICS except that it includes the extra line:

```
%include IMSPCB;
```

Example 8: Server implementation output for the simple interface, *SIMPLEI* generated by *genpli*

```

SIMPLEI: PROC(OPERATION);

dcl OPERATION                char(*);
dcl addr                     builtin;
dcl low                       builtin;
dcl sysnull

                               builtin;

%include CORBA;
%include SIMPLER;
%include SIMPLEM;
/*===== Start of global user code =====*/
/*===== End of global user code =====*/
%include SIMPLED;
/*-----*/
/*                                     */
/* Procedures for Operations          */
/*                                     */
/*-----*/
/*-----*/
/* Operation : call_me               */
/*-----*/
proc_call_me: PROC(P_ARGS);

    dcl p_args                ptr;
    dcl l_args aligned based(p_args)
                               like call_me_type;

/*===== Start of operation specific code =====*/
/*===== End of operation specific code =====*/

end proc_call_me;

end SIMPLEI;

```

Orbix 6.2 Compiler Output

Server implementation output for the `simple` interface, `SIMPLEI`, with the Orbix 6.2 IDL compiler (for Batch, CICS and IMS) is as follows:

Example 9: *Server implementation output for the simple interface, SIMPLEI generated by the Orbix 6.2 IDL compiler (Sheet 1 of 2)*

```

SIMPLEI: PROC;

/*The following line enables the runtime to call this procedure */
DISPTCH: ENTRY;

dcl (addr,low,sysnull)          builtin;

#include CORBA;
#include CHKERRS;
#include SIMPLEM;
#include DISPINIT;

/* ===== Start of global user code ===== */
/* ===== End of global user code ===== */
/*-----*/
/*                                     */
/* Dispatcher : select(operation)      */
/*                                     */
/*-----*/
#include SIMPLED;
/*-----*/
/* Interface:                          */
/*   Simple/SimpleObject                */
/*                                     */
/* Mapped name:                         */
/*   Simple_SimpleObject                */
/*                                     */
/* Inherits interfaces:                 */
/*   (none)                             */
/*-----*/
/* Operation:   call_me                 */
/* Mapped name: call_me                 */
/* Arguments:   None                     */
/* Returns:    void                      */
/*-----*/
proc_Simple_SimpleObject_c_c904: PROC(p_args);

```

Example 9: Server implementation output for the simple interface, SIMPLEI generated by the Orbix 6.2 IDL compiler (Sheet 2 of 2)

```
dcl p_args          ptr;
dcl 1 args          aligned based(p_args)
                   like Simple_SimpleObject_c_ba77_type;

/* ===== Start of operation specific code ===== */

/* ===== End of operation specific code ===== */

END proc_Simple_SimpleObject_c_c904;

END SIMPLEI;
```

Contents of the DISPINIT Member The contents of the DISPINIT Member are:

Example 10: The contents of the DISPINIT Member

```
/* ===== */
/* Copyright 2002 IONA Technologies PLC. All Rights Reserved. */
/* */
/* Member : DISPINIT */
/* Purpose : Retrieve the current server request and operation. */
/* ===== */
/* ===== */
/* reqinfo is used to store information about the current request */
/* ===== */
dcl 1 reqinfo,
    3 interface_name ptr init(sysnull()),
    3 operation_name ptr init(sysnull()),
    3 principal ptr init(sysnull()),
    3 target ptr init(sysnull());

dcl operation char(256);
dcl operation_length fixed bin(31) init(256);

call podreq(reqinfo);
if check_errors('podreq') ^= completion_status_yes then return;

call strget(operation_name,
            operation,
            operation_length);
if check_errors('strget') ^= completion_status_yes then return;
```

PL/I IMS Server Migration Issues

Overview

This section describes the source code changes required when migrating PL/I IMS Orbix 2.3.x servers to PL/I IMS Orbix 6.2 servers.

Note: This section must be read in conjunction with the other PL/I migration issues outlined in this document.

In This Section

This section discusses the following topics:

Server Mainline Module	page 223
Access to the Program Communication Block	page 228

Server Mainline Module

Overview

In Orbix 2.3.x for IMS, a combined server mainline and accessor is generated for all IMS PL/I server programs, as well as an optional server implementation. In Orbix 6.2, by contrast, a server mainline (required) and an optional combined server accessor and implementation is generated.

This section discusses the following topics:

- [Migration Impact](#)
- [Migration Sample IDL](#)
- [Orbix 2.3.x Compiler Output](#)
- [Orbix 6.2 IDL Compiler Output](#)

Migration Impact

The migration impact is that every Orbix 2.3.x IMS PL/I server mainline must be regenerated using the Orbix 6.2 IDL compiler. Refer to the *PL/I Programmer's Guide and Reference* for more details of compiler arguments.

Migration Sample IDL

Consider the following IDL, called `simple`,

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

Orbix 2.3.x Compiler Output

Server mainline output for the `simple` interface, `SIMPLEZ`, with the Orbix 2.3.x IDL compiler is as follows:

Example 11: *Server Mainline Output for the Simple Interface, SIMPLEZ (Sheet 1 of 2)*

```

SIMPLEZ: PROC OPTIONS(MAIN,NOEXECOPS);

/*The following line enables the POD to link to this procedure*/

DISPTCH: ENTRY;

dcl operation                char(256)      init('');
dcl operation_length         fixed bin(31)  init(256);
dcl emptyQ                   bit(1)        init('0'B);

dcl SIMPLEI                  ext entry(char(*));

dcl addr                      builtin;
dcl low                       builtin;
dcl sysnull                   builtin;

#include CORBA;
#include SIMPLER;

dcl ws_interface             char(256);
dcl ws_interface_len         fixed bin(31)  init(256);

alloc pod_status_information set(pod_status_ptr);

call podstat(pod_status_ptr);
if check_errors('podstat') ^= completion_status_yes then return;

do while (^emptyQ);
  call podreq(reqinfo);
  if check_errors('podreq') ^= completion_status_yes then return;

  call strget(interface_name,ws_interface,ws_interface_len);
  if check_errors('strget') ^= completion_status_yes then return;

  call strget(operation_name,
                operation,
                operation_length);
  if check_errors('strget') ^= completion_status_yes then return;

```

Example 11: *Server Mainline Output for the Simple Interface, SIMPLEZ (Sheet 2 of 2)*

```

select(ws_interface);
  when('Simple/SimpleObject') call SIMPLEI(operation);
  otherwise emptyQ='1'B; /* multi-tran test for IMS status QC*/
end;
end;

free pod_status_information;
END SIMPLEZ;

```

Orbix 6.2 IDL Compiler Output

The compiler output for the Orbix 6.2 IDL compiler produces one module for the `simple` interface: a server mainline, `SIMPLEV`. If the `-s` argument is supplied, a skeleton server implementation module, `SIMPLEI`, is also generated.

By default, the Orbix 6.2 IDL compiler generates an `io_pcb_ptr` and an `alt_pcb_ptr` parameter, and then the number of additional `pcb` pointers specified on the command line. To aid migration of Orbix 2.3 PL/I server code to Orbix 6.2, you can specify the `-TIMSG` option with the Orbix IDL compiler, to prevent the generation of `io_pcb_ptr` and `alt_pcb_ptr` identifiers.

Example 12: *The Server Mainline, SIMPLEV, for the simple interface (Sheet 1 of 2)*

```

SIMPLEV: PROC(IO_PCB_PTR,ALT_PCB_PTR) OPTIONS(MAIN NOEXECOPS);

dcl (io_pcb_ptr,alt_pcb_ptr)      ptr;

dcl arg_list                      char(01)      init('');
dcl arg_list_len                  fixed bin(31)  init(0);
dcl orb_name                      char(10)      init('simple_orb');
dcl orb_name_len                  fixed bin(31)  init(10);
dcl srv_name                      char(256)     var;
dcl server_name                   char(07)      init('simple ');
dcl server_name_len               fixed bin(31)  init(6);

dcl Simple_SimpleObject_objid     char(27)
  init('Simple/SimpleObject_object ');
dcl Simple_SimpleObject_obj       ptr;
dcl (addr,length,low,sysnull)     builtin;

```

Example 12: *The Server Mainline, SIMPLEV, for the simple interface (Sheet 2 of 2)*

```

#include CORBA;
#include CHKERRS;
#include IMSPCB;
#include SIMPLET;
#include SIMPLEX;
pcblist.io_pcb_ptr = io_pcb_ptr;
pcblist.alt_pcb_ptr = alt_pcb_ptr;

pcblist.num_db_pcbs = 0;
alloc pod_status_information set(pod_status_ptr);

call podstat(pod_status_ptr);
if check_errors('podstat') ^= completion_status_yes then return;

/* Initialize the server connection to the ORB */
call orbargs(arg_list,arg_list_len,orb_name,orb_name_len);
if check_errors('orbargs') ^= completion_status_yes then return;

call podsrvr(server_name,server_name_len);
if check_errors('podsrvr') ^= completion_status_yes then return;

/* Register interface : Simple/SimpleObject */
call podreg(addr(Simple_SimpleObject_interface));
if check_errors('podreg') ^= completion_status_yes then return;

call objnew(server_name,
            Simple_SimpleObject_intf,
            Simple_SimpleObject_objid,
            Simple_SimpleObject_obj);
if check_errors('objnew') ^= completion_status_yes then return;
/* Server is now ready to accept requests */
call podrun;
if check_errors('podrun') ^= completion_status_yes then return;
call objrel(Simple_SimpleObject_obj);
if check_errors('objrel') ^= completion_status_yes then return;

free pod_status_information;

END SIMPLEV;

```

The server implementation, SIMPLEI, for the simple interface is as follows:

Example 13: *The Server Implementation, SIMPLEI, for the simple Interface*

```

SIMPLEI: PROC;
/*The following line enables the runtime to call this procedure*/
DISPTCH: ENTRY;

dcl (addr,low,sysnull)          builtin;
%include CORBA;
%include CHKERS;
%include IMSPCB;
%include SIMPLEM;
%include DISPINIT;

/* ===== Start of global user code ===== */
/* ===== End of global user code ===== */
/*-----*/
/* Dispatcher : select(operation) */
/*-----*/
%include SIMPLED;
/*-----*/
/* Interface: */
/* Simple/SimpleObject */
/* */
/* Mapped name: */
/* Simple_SimpleObject */
/* */
/* Inherits interfaces: */
/* (none) */
/*-----*/
/* Operation: call_me */
/* Mapped name: call_me */
/* Arguments: None */
/* Returns: void */
/*-----*/
proc_Simple_SimpleObject_c_c904: PROC(p_args);

dcl p_args ptr;
dcl 1 args aligned based(p_args)
like Simple_SimpleObject_c_ba77_type;

/* ===== Start of operation specific code ===== */
/* ===== End of operation specific code ===== */

END proc_Simple_SimpleObject_c_c904;

END SIMPLEI;

```

Access to the Program Communication Block

In This Section

This section discusses the following topics:

- [Server Implementation Code](#)
 - [Server Mainline Code](#)
 - [The Format of IMSPCB](#)
-

Server Implementation Code

Orbix 6.2 IDL compiler output server implementation code has access to the program communication block through the static structures stored in `IMSPCB`.

Server Mainline Code

Orbix 6.2 IDL compiler output server mainline code allows access to the program communication block by setting the addresses of the PCB pointers to the structure `pcblist`, declared in `IMSPCB`. The number of database pointers is also set.

Note: The server implementation to access program communication block data must have an include statement for `IMSPCB` added if the `:-S:-TIMS` options are not used to generate the server implementation, that is, if the server implementation migration changes are coded manually.

The Format of IMSPCB

`IMSPCB` has the format:

```

/*****
/* The PCBLIST allows access to the PCB pointers from anywhere*/
/* within the PL/I IMS server code */
/*****
DCL 1 PCBLIST STATIC EXT,
    3 IO_PCB_PTR          PTR          INIT(SYSNULL()),
    3 ALT_PCB_PTR         PTR          INIT(SYSNULL()),
    3 PCB_PTR(64)         PTR          INIT((64)SYSNULL()),
    3 NUM_DB_PCBS         FIXED BIN(31) INIT(0);

DCL 1 IO_PCB BASED(PCBLIST.IO_PCB_PTR),
    3 LTERM               CHAR(08),
    3 FILLER              CHAR(02),
    3 STATUS_CODE         CHAR(02),
    3 MSG_DATE            FIXED DEC(7,0),
    3 MSG_TIME            FIXED DEC(7,0),
    3 MSG_SEQ_NO          FIXED BIN(31),
    3 MOD_NAME            CHAR(08),
    3 USERID              CHAR(08),
    3 GROUP_NAME          CHAR(08);

DCL 1 ALT_PCB BASED(PCBLIST.ALT_PCB_PTR),
    3 LTERM               CHAR(08),
    3 FILLER              CHAR(02),
    3 STATUS_CODE         CHAR(02);

```

PL/I IMS Client Migration issues

Overview

This section describes the source code changes required when migrating PL/I IMS Orbix 2.3.x clients to PL/I IMS Orbix 6.2 clients.

Note: This section must be read in conjunction with the other PL/I migration issues outlined in this document.

Note: The `DISPTCH` reference must be removed from client code and replaced with the line `%client_only='yes';`. Refer to [“DISPTCH Reference” on page 244](#) for further details.

In This Section

This section discusses the following topics:

Program Communication Block Definitions Modifications	page 231
DLIDATA Include Member Modifications	page 234
Error Checking Generation at Runtime for IMS Clients	page 235

Program Communication Block Definitions Modifications

Overview

Program communication block definitions in an Orbix 2.3.x client implementation and program communication block definitions in an Orbix 6.2 client implementation are not the same.

This section discusses the following topics:

- [Orbix 6.2 client implementation sample](#)
 - [Orbix 2.3 client implementation sample](#)
 - [Migration impact](#)
-

Orbix 6.2 client implementation sample

In Orbix 6.2, the program communication blocks are defined as:

```

/*****
/* The PCBLIST allows access to the PCB pointers from anywhere*/
/* within the PL/I IMS server code */
*****/
DCL 1 PCBLIST STATIC EXT,
    3 IO_PCB_PTR PTR INIT(SYSNULL()),
    3 ALT_PCB_PTR PTR INIT(SYSNULL()),
    3 PCB_PTR(64) PTR INIT((64)SYSNULL()),
    3 NUM_DB_PCBS FIXED BIN(31) INIT(0);
DCL 1 IO_PCB BASED(PCBLIST.IO_PCB_PTR),
    3 LTERM CHAR(08),
    3 FILLER CHAR(02),
    3 STATUS_CODE CHAR(02),
    3 MSG_DATE FIXED DEC(7,0),
    3 MSG_TIME FIXED DEC(7,0),
    3 MSG_SEQ_NO FIXED BIN(31),
    3 MOD_NAME CHAR(08),
    3 USERID CHAR(08),
    3 GROUP_NAME CHAR(08);
DCL 1 ALT_PCB BASED(PCBLIST.ALT_PCB_PTR),
    3 LTERM CHAR(08),
    3 FILLER CHAR(02),
    3 STATUS_CODE CHAR(02);

```

Orbix 2.3 client implementation sample

In Orbix 2.3.x the program communication blocks are defined as:

```
dcl iopcb_ptr          ptr;
dcl 1 iopcb based(iopcb_ptr),
    3 lterm_name      char(08),
    3 filler1         char(02),
    3 tpstatus        char(02),
    3 filler2         char(20);
```

Migration impact

Migration impact is to replace the code shown in the:

- Replace

```
dcl iopcb_ptr          ptr;
dcl 1 iopcb based(iopcb_ptr),
    3 lterm_name      char(08),
    3 filler1         char(02),
    3 tpstatus        char(02),
    3 filler2         char(20);
```

with %include IMSPCB;

- Replace

```
SIMPLEC: PROC(IOPCB_PTR) OPTIONS(MAIN, NOEXECOPS);
dcl iopcb_ptr          ptr;
```

with

```
SIMPLEC: PROC(IO_PCB_PTR,ALT_PCB_PTR) OPTIONS(MAIN
NOEXECOPS);
dcl (io_pcb_ptr,alt_pcb_ptr)          ptr;
```

- Replace

```
call plitdli(three,get_unique,IOPCB_PTR,input_msg);
if tpstatus ^= '' then call write_dc_text('Segment read
failed',19);
```

with

```
%include GETUNIQ;  
...  
pcblast.io_pcb_ptr = io_pcb_ptr;  
pcblast.alt_pcb_ptr = alt_pcb_ptr;  
call get_uniq;
```

DLIDATA Include Member Modifications

Overview

This subsection describes migration for the `DLIDATA` include member from Orbix 2.3.x to Orbix 6.2.

This subsection discusses the following topics:

- [Orbix 2.3.x](#)
- [Orbix 6.2](#)
- [Migration impact](#)

Orbix 2.3.x

In Orbix 2.3.x, the definition `dcl plitdli ext entry;` is located in the client mainline.

Orbix 6.2

In Orbix 6.2, the definition `dcl plitdli ext entry;` is located in the `DLIDATA` include member.

Migration impact

The Orbix 6.2 `DLIDATA` include member must be used and the definition `dcl plitdli ext entry;` must be removed from the client mainline.

Error Checking Generation at Runtime for IMS Clients

Overview

This sections summarizes the differences between an Orbix 2.3.x client and an Orbix 6.2 client in relation to the `CHECK_ERRORS` function used for error checking.

This section discusses the following topics:

- [IMS clients in Orbix 2.3.x](#)
- [IMS clients in Orbix 6.2](#)
- [Migration impact](#)

IMS clients in Orbix 2.3.x

There is no member shipped for error-checking for IMS client code in Orbix 2.3.x. Customers are required to implement their own error checking procedure.

IMS clients in Orbix 6.2

For IMS clients a static member called `CHKCLIMS` is shipped which contains a `CHECK_ERRORS` function and is located in the `orbixhlq.INCLUDE.COPYLIB` in Orbix 6.2.

Migration impact

There is no migration impact, however IONA recommend you use the `CHKCLIMS` member which shows the system exception encountered in a more user-friendly format.

PL/I CICS Server Migration Issues

Overview

This section describes the source code changes required when migrating PL/I CICS Orbix 2.3.x servers to PL/I CICS Orbix 6.2 servers.

Note: This section must be read in conjunction with the other PL/I migration issues outlined in this document.

In this section

This section discusses the following topics:

Server Mainline Program Requirements for CICS Servers	page 237
Access to the EXEC Interface Block Data Structure	page 242

Server Mainline Program Requirements for CICS Servers

Overview

In Orbix 2.3.x for CICS, a combined server mainline and accessor is generated for all CICS PL/I server programs, as well as an optional server implementation. In Orbix 6.2, in contrast, a server mainline (required) and an optional combined server accessor and implementation is generated.

This subsection discusses the following topics:

- [Migration Impact](#)
- [Migration Sample IDL](#)
- [Orbix 2.3.x Compiler Output](#)
- [Orbix 6.2 IDL Compiler Output](#)

Migration Impact

The migration impact is that every Orbix 2.3.x IMS PL/I server mainline has to be regenerated using the Orbix 6.2 IDL compiler. Refer to the *PL/I Programmer's Guide and Reference* for more details of compiler arguments.

Also the Orbix 2.3.x server mainline for CICS contains a CICS program pointer which is passed into the program. This pointer is not supported in Orbix 6.2.

Migration Sample IDL

Consider the following IDL, called `simple`,

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

Orbix 2.3.x Compiler Output

Server mainline output for the `simple` interface, `SIMPLEZ`, with the Orbix 2.3.x IDL compiler is as follows:

Example 14: *Orbix 2.3.x Compiler Output for the simple IDL*

```

SIMPLEZ: PROC OPTIONS(MAIN,NOEXECOPS);

/*The following line enables the POD to link to this procedure*/

DISPTCH: ENTRY;

dcl operation                char(256)    init('');
dcl operation_length         fixed bin(31)  init(256);

dcl SIMPLEI                  ext entry(char(*),ptr);
dcl PODCICS                  ext entry;

dcl addr                     builtin;
dcl low                      builtin;
dcl sysnull                  builtin;

%include CORBA;
%include SIMPLER;

alloc pod_status_information set(pod_status_ptr);

call podstat(pod_status_ptr);
if check_errors('podstat') ^= completion_status_yes then return;

call podreq(reqinfo);
if check_errors('podreq') ^= completion_status_yes then return;

call strget(operation_name,
              operation,
              operation_length);
if check_errors('strget') ^= completion_status_yes then return;

call SIMPLEI(operation,p_prgptr);

free pod_status_information;

END SIMPLEZ;

```

Orbix 6.2 IDL Compiler Output

The compiler output for the Orbix 6.2 IDL compiler produces a module for the `simple` interface: a server mainline, `SIMPLEV`. If the `-s` argument is supplied a combined server accessor and implementation module, `SIMPLEI`, is also generated.

Example 15: *The Server Mainline, `SIMPLEV`, for the `simple` interface (Sheet 1 of 2)*

```
SIMPLEV: PROC OPTIONS(MAIN NOEXECOPS);

dcl arg_list          char(01)      init('');
dcl arg_list_len     fixed bin(31)  init(0);
dcl orb_name         char(10)       init('simple_orb');
dcl orb_name_len     fixed bin(31)  init(10);
dcl srv_name         char(256)      var;
dcl server_name      char(07)       init('simple ');
dcl server_name_len  fixed bin(31)  init(6);

dcl Simple_SimpleObject_objid char(27)
      init('Simple/SimpleObject_object ');
dcl Simple_SimpleObject_obj  ptr;
dcl (addr,length,low,sysnull) builtin;

%include CORBA;
%include CHKERS;
%include SIMPLET;
%include SIMPLEX;
```

Example 15: *The Server Mainline, SIMPLEV, for the simple interface (Sheet 2 of 2)*

```

alloc pod_status_information set(pod_status_ptr);

call podstat(pod_status_ptr);
if check_errors('podstat') ^= completion_status_yes then return;

/* Initialize the server connection to the ORB */
call orbargs(arg_list,arg_list_len,orb_name,orb_name_len);
if check_errors('orbargs') ^= completion_status_yes then return;

call podsrvr(server_name,server_name_len);
if check_errors('podsrvr') ^= completion_status_yes then return;

/* Register interface : Simple/SimpleObject */
call podreg(addr(Simple_SimpleObject_interface));
if check_errors('podreg') ^= completion_status_yes then return;

call objnew(server_name,
            Simple_SimpleObject_intf,
            Simple_SimpleObject_objid,
            Simple_SimpleObject_obj);
if check_errors('objnew') ^= completion_status_yes then return;
/* Server is now ready to accept requests */
call podrun;
if check_errors('podrun') ^= completion_status_yes then return;
call objrel(Simple_SimpleObject_obj);
if check_errors('objrel') ^= completion_status_yes then return;

free pod_status_information;

END SIMPLEV;

```

The server accessor and implementation, SIMPLEI, is as follows:

Example 16: *The Server Implementation, SIMPLEI, for the simple Interface (Sheet 1 of 2)*

```

SIMPLEI: PROC;

/*The following line enables the runtime to call this procedure*/
DISPTCH: ENTRY;

dcl (addr,low,sysnull)          builtin;

```

Example 16: *The Server Implementation, SIMPLEI, for the simple Interface (Sheet 2 of 2)*

```

#include CORBA;
#include CHKERRS;
#include SIMPLEM;
#include DISPINIT

/* ===== Start of global user code ===== */
/* ===== End of global user code ===== */

/*-----*/
/*                                     */
/* Dispatcher : select(operation)      */
/*                                     */
/*-----*/
#include SIMPLED;

/*-----*/
/* Interface:                          */
/*   Simple/SimpleObject                */
/*                                     */
/* Mapped name:                         */
/*   Simple_SimpleObject                */
/*                                     */
/* Inherits interfaces:                 */
/*   (none)                             */
/*-----*/
/* Operation:   call_me                */
/* Mapped name: call_me                */
/* Arguments:   None                    */
/* Returns:     void                    */
/*-----*/
proc_Simple_SimpleObject_c_c904: PROC(p_args);

dcl p_args          ptr;
dcl 1 args          aligned based(p_args)

      likeSimple_SimpleObject_c_ba77_type;
/* ===== Start of operation specific code ===== */
/* ===== End of operation specific code ===== */

END proc_Simple_SimpleObject_c_c904;

END SIMPLEI;

```

Access to the EXEC Interface Block Data Structure

Overview

This subsection describes the migration impact for CICS PL/I servers whose implementation requires access to the EXEC interface block (EIB) data structure. It discusses the following topics:

- [Migration Impact](#)
- [Required Code](#)

Migration Impact

Because Orbix 6.2 requires that all CICS PL/I servers have a server mainline, the implementation program is now a sub-program that is entered via a DISPTCH entry point. By default, the CICS program does not pass along the address of the EIB structure. Therefore, you must add some additional code to your PL/I server implementation programs.

Required Code

Add the following line of code after the DISPTCH entry point:

```
EXEC CICS ADDRESS EIB(DFHEIPTR);
```

PL/I CICS Client Migration Issues

Overview

This section describes the source code changes required when migrating PL/I CICS Orbix 2.3.x clients to PL/I CICS Orbix 6.2 clients.

Note: This section must be read in conjunction with the other PL/I migration issues outlined in this document.

This section discusses the following topics:

- [CICS clients in Orbix 2.3.x and error checking](#)
- [CICS clients in Orbix 6.2 and error checking](#)
- [Migration impact for error checking code](#)
- [DISPTCH reference](#)

CICS clients in Orbix 2.3.x and error checking

There is no member shipped for error-checking for CICS client code in Orbix 2.3.x. Customers are required to implement their own error checking procedure.

CICS clients in Orbix 6.2 and error checking

For CICS clients a static member called `CHKCLCIC` shipped which contains a `CHECK_ERRORS` function and is located in the `orbixhlq.INCLUDE.PLINCL` in Orbix 6.2.

Migration impact for error checking code

There is no migration impact, however IONA recommend you use the `CHKCLCIC` member which shows the system exception encountered in a more user-friendly format.

Note: `CHKCLCIC` is relevant to CICS clients only. It contains a PL/I function that has been translated by the CICS TS 1.3 translator. This function can be called by the client, to check if a system exception has occurred and report it.

DISPTCH reference

The `DISPTCH` reference must be removed from client code and replaced with the line `%client_only='yes';`. Refer to [“DISPTCH Reference” on page 244](#) for further details.

Miscellaneous

In This Section

This section discusses the following topics:

- [Interface Repository Server](#)
- [Command-Line Arguments](#)
- [DISPTCH Reference](#)
- [Inherited interfaces](#)
- [Orbix PL/I include file re-arrangement](#)
- [Generation of mapping files](#)

Interface Repository Server

In Orbix 2.3.x, `genpli` requires the Interface Repository (IFR) server to be running to access the IDL source registered with the IFR server.

The Orbix 6.2 IDL Compiler accesses the IDL source directly, from the input IDL member (data set), and therefore does not need to access the IFR. Hence IDL members can be accessed independently (and IDL to PL/I development can proceed) without the need for any Orbix 6.2 services to be running.

Command-Line Arguments

The command-line arguments for the Orbix 6.2 IDL Compiler are different in some cases to the `genpli` arguments. However, functionality common to both compilers can be achieved.

DISPTCH Reference

Orbix 2.3.x required both clients and servers to have the label `DISPTCH` defined at the start of the client program and server accessor code (`idlmembername2`). For Orbix 6.2, you *must* remove this line, `DISPTCH: ENTRY`, from the client code and replace it with:

```
%client_only='yes';
```

In Orbix 6.2 PL/I it is defined in the server implementation (the `DISPTCH` label is still required by the server mainline) and can only be defined once in a program.

The reason for making the change is that when your client program is compiled, it then only pulls in client-specific functionality of the PL/I runtime, resulting in smaller load module size.

Inherited interfaces

The IDL-PL/I generator now generates only one instance of a PL/I typedef per IDL type. In previous releases, if a type was inherited, the PL/I generator created a typedef for both the base class's instance of the type and also one for each inherited type. This was unnecessary as both generated typedefs would always be the same, apart from the name of the typedef. It also resulted in the generation of large include files in the cases of IDL with complex structs, for example. For programs where a pre-Orbix 6.2 generated server implementation is used and new include files need to be generated, the `-Li` option has been introduced.

Orbix PL/I include file re-arrangement

Three PL/I include members (`CORBA`, `READIOR` and `SETUPCL`) have been reorganized, to decrease the number of instances where the compilation of an Orbix PL/I program results in a return code of 4, due to the pre-processor check for `client_only`. The reorganization has been designed so that there would not be a migration hit for existing Orbix PL/I applications. Additionally, a new include file, `SETUPSV`, has been added, to declare `client_only` and set it to "no" in Orbix PL/I server applications. For further details about the include members, see the *Orbix PL/I Programmers Guide and Reference*.

Generation of mapping files

In previous versions of the Orbix PL/I generator, if the `-M` option was specified and the IDL had operation names that were identical in several interfaces, no warning was produced if the names mapped to a non-unique name. For example, no warning was produced if the generated mapping file contained:

```
interfaceA/ping ping
interfaceB/ping ping
```

The Orbix 6.2 PL/I generator will still generate the preceding mapping file but also outputs a warning about the generated mapping file. The generator will also give a return code of 4, to alert the developer that two or more operations have been mapped to the same name.

Diagnostic Output

This chapter summarizes the differences between how diagnostic data is output for Orbix 2.3.x and Orbix 6.2.

In this chapter

This chapter discusses the following topics:

CORBA::Orbix::setDiagnostics () Availability	page 248
Orbix Diagnostic Messages	page 248
Orbix 6.2 Default Diagnostic Output	page 248
Logging Severity Levels	page 248
COBOL and PL/I	page 248
Further Reading	page 248

CORBA::Orbix::setDiagnostics () Availability

`CORBA::Orbix::setDiagnostics()` is not available in Orbix 6.2, because it is not CORBA-compliant. Instead, diagnostic output is controlled from within the Orbix 6.2 configuration. This allows easy manipulation of diagnostic output. In addition, the diagnostic output of each Orbix 6.2 plugin can be controlled separately, allowing for informative and selective diagnostic output.

Orbix Diagnostic Messages

The following table compares Orbix diagnostic messages to their equivalent configuration settings in Orbix 6.2:

Orbix Diagnostic Setting	Orbix 6.2 Configuration Setting
<code>setDiagnostics(0)</code>	No logging plug-ins loaded.
<code>setDiagnostics(1)</code>	<code>event_log:filters=["*=FATAL+ERROR"];</code>
<code>setDiagnostics(2)</code>	<code>event_log:filters=["*="];</code>

Orbix 6.2 Default Diagnostic Output

By default, diagnostic output goes to standard error, but it can be directed to a file with the `local_log_stream` configuration variable as follows:

```
plugins:local_log_stream:filename = /var/adm/Orbix2000.log
```

Logging Severity Levels

There are four levels of logging severity within Orbix 6.2. These are:

- Informational
- Warning
- Error
- Fatal Error

COBOL and PL/I

COBOL and PL/I now have the flexibility to control the diagnostic level.

Note: `setDiagnostics` in the preceding example is specific to C++.

Note: The PL/I debug library is no longer shipped with Orbix 6.2.

Further Reading

Refer to the *CORBA Administrator's Guide* for further details on diagnostic output.

CORBA Services

This chapter summarizes the differences in CORBA services between Orbix 2.3.x and Orbix 6.2.

In this chapter

This chapter discusses the following topics:

Naming Service	page 250
Interface Repository	page 251
IMS Adapter	page 252
CICS Adapter	page 254

Naming Service

Backward compatibility

The Orbix 6.2 Naming Service is backward compatible with Orbix 2.3.x in two respects:

- *Source code backward compatibility*: source code that is written to use the standard naming service interfaces can be migrated to Orbix 6.2 without modification.
 - *On-the-wire backward compatibility*: Orbix 2.3.x applications can interoperate with the Orbix 6.2 naming service. If you need to interoperate Orbix 2.3.x applications, it is recommended that you recompile the naming stub code from the Orbix 6.2 IDL files.
-

New interface

Orbix 6.2 adds a new interface, `CosNaming::NamingContextExt`, which is defined by the CORBA Interoperable Naming Service specification. This interface adds support for using names in stringified format.

Load balancing

The naming service load-balancing extensions provided in Orbix 2.3.x are also present in Orbix 6.2. The Orbix 6.2 load-balancing interfaces are only slightly different from Orbix 2.3.x, requiring small modifications to your source code.

Interface Repository

Migration

Migrating source code that uses the Interface Repository (IFR) to Orbix 6.2 is straightforward. Link the migrated application against the stub code derived from the Orbix 6.2 version of the interface repository. No further changes should be necessary.

IMS Adapter

Overview

In Orbix 2.3.x, Orbix IMS adapter functionality is controlled via a series of command-line arguments that can be specified to the adapter at start-up. In Orbix 6.2, Orbix IMS adapter functionality is controlled via a series of configuration items in the adapter's configuration domain.

This section provides a comparison table of the 2.3.x-based adapter arguments and 6.2 adapter configuration items.

Differences in controlling OTMA-based IMS adapter functionality

[Table 17](#) outlines the 2.3.x command-line arguments that correspond to the 6.2 configuration items for the purposes of controlling the functionality of OTMA-based IMS adapters.

Table 17: *Differences in Controlling OTMA-Based IMS Adapters*

2.3.x Arguments	6.2 Configuration Items
-A	<code>plugins:imsa:display_timings = "yes";</code>
-f	<code>plugins:imsa:mapping_file = " ";</code>
-G	<code>plugins:ims_otma:xcf_group_name = "IMSG";</code>
-X	<code>plugins:ims_otma:xcf_adapter_member_name = "ORXIMSG";</code>
-M	<code>plugins:ims_otma:xcf_ims_member_name = "IMS";</code>
-T	<code>plugins:ims_otma:xcf_tpipe_prefix = "ORX1";</code>
-w	<code>plugins:ims_otma:timeout = "30";</code>
-o	<code>plugins:ims_otma:output_segment_num = "2";</code>
-l	<code>plugins:ims_otma:mq_length = "1024";</code>
-p	<code>thread_pool:initial_threads = "8";</code>

Differences in controlling APPC-based IMS adapter functionality

Table 18 outlines the 2.3.x command-line arguments that correspond to the 6.2 configuration items for the purposes of controlling the functionality of APPC-based IMS adapters.

Table 18: *Differences in Controlling APPC-Based IMS Adapters*

2.3.x Arguments	6.2 Configuration Items
-A	<code>plugins:imsa:display_timings = "yes";</code>
-n	<code>plugins:ims_appc:ims_destination_name = "ORBIXIMS";</code>
-L	<code>plugins:ims_appc:appc_outbound_lu_name = " ";</code>
-w	<code>plugins:ims_appc:timeout = "30";</code>
-l	<code>plugins:ims_appc:mq_length = "1024";</code>
-P	<code>thread_pool:initial_threads = "8";</code>

CICS Adapter

Overview

In Orbix 2.3.x, Orbix CICS adapter functionality is controlled via a series of command-line arguments that can be specified to the adapter at start-up. In Orbix 6.2, Orbix CICS adapter functionality is controlled via a series of configuration items in the adapter's configuration domain.

This section provides a comparison table of the 2.3.x-based adapter arguments and 6.2 adapter configuration items.

Differences in controlling EXCI-based CICS adapter functionality

Table 19 outlines the 2.3.x command-line arguments that correspond to the 6.2 configuration items for the purposes of controlling the functionality of EXCI-based CICS adapters.

Table 19: *Differences in Controlling EXCI-Based CICS Adapters*

2.3.x Arguments	6.2 Configuration Items
-A	<code>plugins:cicsa:display_timings = "yes";</code>
-f	<code>plugins:cicsa:mapping_file = " ";</code>
-n	<code>plugins:cics_exci:applid = "CICSTS1";</code>
-N	<code>plugins:cics_exci:pipe_name = "ORXPIPE1";</code>
-m	<code>plugins:cics_exci:default_tran_id = "ORX1";</code>
-l	<code>plugins:cics_exci:max_comm_area_length = "32000";</code>
-p	<code>thread_pool:initial_threads = "8";</code>

Differences in controlling APPC-based CICS adapter functionality

Table 20 outlines the 2.3.x command-line arguments that correspond to the 6.2 configuration items for the purposes of controlling the functionality of APPC-based CICS adapters.

Table 20: *Differences in Controlling APPC-Based CICS Adapters*

2.3.x Arguments	6.2 Configuration Items
-A	plugins:cicsa:display_timings = "yes";
-f	plugins:cicsa:mapping_file = " ";
-n	plugins:cics_appc:cics_destination_name = "ORBIXCIC";
-L	plugins:cics_appc:appc_outbound_lu_name = "ORXLU02";
-w	plugins:cics_appc:timeout = "6";
-l	plugins:cics_appc:segment_length = "32767";
-p	thread_pool:initial_threads = "8";

Administrative Tools

This chapter summarizes the differences between Orbix 2.3.x and Orbix 6.2 administration tools.

In this chapter

This chapter discusses the following topics:

Orbix 2.3.x Administration Tools	page 258
Orbix 6.2 Administration Tools	page 258
The itadmin Tool and z/OS	page 258
z/OS UNIX System Services Single Command Line	page 258
z/OS UNIX System Services Interactive Shell Mode	page 258
z/OS Native	page 259
Further Reading	page 259

Orbix 2.3.x Administration Tools

Orbix 2.3.x supplies various utilities to administer its various components. Among these tools, for example, are `putit` and `rmit` used to administer the implementation repository, `putidl` and `rmidl` are used to administer the interface repository, and `lsns` and `putns` are used to administer the Naming Service.

Orbix 6.2 Administration Tools

Orbix 6.2 unifies all administrative commands under a single tool, `itadmin`, that can manage all IONA services.

The itadmin Tool and z/OS

The `itadmin` tool is used on z/OS in different ways depending on the environment. There are three environments which dictate the way it is used. These are:

- z/OS UNIX System Services:
 - ◆ single command line.
 - ◆ interactive shell mode.
 - z/OS native:
 - ◆ batch mode.
-

z/OS UNIX System Services Single Command Line

On z/OS UNIX System Services the `itadmin` tool can be run on the command line as in the following example:

```
$ itadmin help
$ itadmin poa -help
```

z/OS UNIX System Services Interactive Shell Mode

On z/OS UNIX System Services interactive shell mode, multiple `itadmin` commands can be invoked within the same shell process. For example:

```
$ itadmin
% poa list -active
% ifr show grid
% ns newnc
% exit
```

z/OS Native

On z/OS native, the `itadmin` tool can be run in batch by executing the IONA supplied `ORXADMIN PROC` in your JCL. One or more `itadmin` commands can be specified in the `SYSIN DD` concatenation. For example in the following JCL excerpt:

```
//REG      EXEC PROC=ORXADMIN
//SYSIN DD  *
orbname create simple_orb
poa create -orbname simple_orb simple_persistent
/*
```

Further Reading

Refer to the *CORBA Administrator's Guide* for further information about using the `itadmin` tool.

Interoperability

This chapter describes the issues relating to interoperability when migrating from an Orbix 2.3-based IONA mainframe solution to Orbix Mainframe 6.2.

In this chapter

This chapter discusses the following topics:

Use of the Orbix Protocol	page 262
GIOP Versions	page 263
Launch and Invoke Rights	page 265
Codeset Negotiation	page 267

Use of the Orbix Protocol

Overview

This section discusses migration from IONA's proprietary Orbix protocol to CORBA-compliant transport protocols.

This section discusses the following topics:

- [Orbix 6.2 and Transport Protocols](#)
- [Migration Impact](#)

Orbix 6.2 and Transport Protocols

Orbix 6.2 supports only CORBA-compliant transport protocols such as IIOP.

Migration Impact

If you have old (pre-Orbix 2.3.x) code that relies on the Orbix Protocol, or code that calls `CORBA::Orbix.bindUsingIIOP(0)`, you must change it to use IIOP. Otherwise, the Orbix client cannot invoke on any Orbix 6.2 component.

GIOP Versions

GIOP version of a connection

The GIOP version used by a client-server connection is determined by the client. When a client is about to open a connection to a CORBA object, the client examines the version information in the object's IOR:

- If the GIOP version in the IOR is greater than or equal to the default GIOP version of the client, the client initiates a connection using the client's default GIOP version.
- Otherwise, the client initiates a connection using the GIOP version in the IOR.

Effect of GIOP version

The GIOP version of a connection is important, because some CORBA features are not supported in early GIOP versions. [Table 21](#) shows the minimum GIOP version required for some CORBA features, according to the CORBA specification.

Table 21: *CORBA-Specified Minimum GIOP Versions*

CORBA Feature	CORBA-Specified Minimum GIOP Version
fixed type	1.1
wchar and wstring types	1.1
codeset negotiation (Orbix 6.2 only)	1.1

Orbix-specific minimum GIOP versions

Notwithstanding the CORBA-specified minimum GIOP versions, Orbix allows some features to be used at a lower GIOP version (in some cases requiring specific configuration variables to be set). Table 22 shows the Orbix-specific minimum GIOP versions.

Table 22: *Orbix-Specific Minimum GIOP Versions*

CORBA Feature	Orbix-Specific Minimum GIOP Version
fixed type	1.0
wchar and wstring types	1.0
codeset negotiation (Orbix 6.2 only)	1.1

For more details on these CORBA features, see the following sections in the Orbix 6.2 *Migrating from Orbix 3.3. to Orbix 6.2* guide at

<http://www.iona.com/support/docs/orbix/6.2/migrate.xml>:

- "Fixed Data Type and Interoperability".
- "Use of wchar and wstring".
- "Introduction to Codeset Negotiation".

Table of default GIOP versions

Table 19 shows the default GIOP versions for different Orbix clients when opening a connection to a server.

Table 23: *Default GIOP Version Used by Orbix Clients*

Client Version	Default GIOP Version
Orbix 3.0.1-82	1.0
OrbixWeb 3.2-15	1.0
Orbix 3.3 C++ Edition	1.1
Orbix 3.3 Java Edition	1.0
Orbix 6.2	1.1

Launch and Invoke Rights

Overview

When an Orbix 6.2 client attempts to open a connection to an Orbix 2.3.x server you must ensure that the system is configured such that the Orbix 6.2 client has launch and invoke rights.

Role of launch and invoke rights

In Orbix 2.3.x, the `orbixd` daemon process is responsible both for launching servers and for redirecting client requests to servers. These two functions are governed by *launch rights* and *invoke rights*, respectively.

Launch and invoke rights on Orbix 3.3 servers are based on the idea that the client *userID* is transmitted along with request messages. The field of the request message that contains the user ID is known as the Principal of the invocation.

If launch and invoke rights are not configured correctly, the Orbix 6.2 client raises a `CORBA::OBJECT_NOT_EXIST` system exception.

Setting launch rights

The launch rights associated with an Orbix 3.3 server specify which users are allowed to cause automatic launching of the server. Launch rights in Orbix 3.3 are granted with the following form of `chmodit`:

```
chmodit l+userID ServerName
```

Setting invoke rights

The invoke rights associated with an Orbix 3.3 server are used to determine which users are allowed to invoke on the server. Invoke rights are granted using:

```
chmodit i+userID ServerName
```

Orbix 6.2 and Orbix 3.3

The configuration must be altered for an Orbix 6.2 client invoking on an Orbix 3.3 server. There are two possible approaches to fix the launch and invoke rights:

- Alter the configuration of the Orbix 6.2 client.
- Relax the security on the `orbixd` daemon.

Alter the configuration of the Orbix 6.2 client

Four configuration variables must be made (or changed) in the Orbix 6.2 configuration file:

```
# Orbix 6.2 Configuration File
policies:giop:interop_policy:send_locate_request = "false";
policies:giop:interop_policy:send_principal = "true";
policies:giop:interop_policy:enable_principal_service_context =
"true";
policies:giop:interop_policy:ignore_message_not_consumed =
"true";
```

The `policies:giop:interop_policy:send_locate_request` option controls whether Orbix 6.2 sends `LocateRequest` messages before sending initial `Request` messages. This option must be set to `"false"` because `LocateRequest` messages do not contain a `Principal` field.

Note: To allow Orbix 2.3.5 or higher Orbix servers interoperate with Orbix 6.2 clients, you must set the `policies:giop:interop_policy:send_locate_request` configuration item to `"false"`.

The `policies:giop:interop_policy:send_principal` option controls whether Orbix 6.2 sends `Principal` information containing the current user name in GIOP 1.0 and GIOP 1.1 requests. The user name is matched against the launch and invoke rights listed in the `orbixd` daemon, to determine the permissions of the Orbix 6.2 client.

Relax the security on the orbixd daemon

Alternatively, you can relax the security on the `orbixd` daemon so that all clients have launch and invoke rights. For example, use the `chmodit` command line utility to change the launch and invoke rights:

```
chmodit l+all ServerName
chmodit i+all ServerName
```

These commands give permission for any client to invoke or launch the server `ServerName`. Permissions are granted even if the `Principal` value is left blank in the incoming requests.

Codeset Negotiation

Overview

Codeset negotiation enables CORBA applications to agree on a common character set for transmission of narrow and wide characters.

In this section

This section discusses the following topics:

Introduction to Codeset Negotiation	page 268
Configuring Codeset Negotiation	page 269
Default Codesets	page 270
Configuring Legacy Behavior	page 273

Introduction to Codeset Negotiation

Overview

The CORBA codeset conversion framework enables applications to ensure that they communicate using compatible character formats for both narrow characters, `char`, and wide characters, `wchar`.

Support for codeset negotiation

Orbix 2000 (version 2.0 and later) and Orbix 6.2 support codeset negotiation, as defined by the CORBA 2.4 specification.

Orbix 2.3.x does not support codeset negotiation.

Servers and codeset negotiation

A server that supports codeset negotiation appends a list of supported codesets (character formats) to the interoperable object references (IORs) it generates. The codesets are placed in standard `IOP::TAG_CODE_SETS` components in the IOR.

Clients and codeset negotiation

A client that supports codeset negotiation examines an IOR to check the list of codesets supported by the server. The client compares this list with its own list of supported codesets and, if a match is found, the client chooses the pair of transmission codesets (narrow character format and wide character format) to use for that particular connection.

When sending a `Request` message, the client appends an `IOP::CodeSets` service context that tells the server which codesets are used. The client continues to include an `IOP::CodeSets` service context in `Request` messages until the first `Reply` message is received from the server. Receipt of the first server `Reply` message implicitly indicates that codeset negotiation is complete. The same character formats are used for subsequent communication on the connection.

Configuring Codeset Negotiation

Overview

Orbix 6.2 features greatly enhanced support for internationalization and codeset negotiation. In particular, it is now possible to specify explicitly the codesets that a server exports in an IOR.

CORBA configuration variables

[Table 24](#) gives the configuration variables that are used to specify the codesets for an Orbix 6.2 CORBA application.

Table 24: *CORBA Codeset Configuration Variables (Orbix 6.2)*

Configuration Variable	Description
<code>plugins:codeset:char:ncs = "<codeset>";</code>	Specifies the native narrow character codeset.
<code>plugins:codeset:char:ccs = "<codeset1>", "<codeset2>", ...];</code>	Specifies the list of conversion narrow character codesets supported.
<code>plugins:codeset:wchar:ncs = "<codeset>";</code>	Specifies the native wide character codeset.
<code>plugins:codeset:wchar:ccs = "<codeset1>", "<codeset2>", ...];</code>	Specifies the list of conversion wide character codesets supported.
<code>plugins:codeset:always_use_default = "<boolean>";</code>	Specifies that hardcoded default values are used and the preceding variables are ignored, if set in the same configuration scope or higher.

Default Codesets

Overview

This section describes the default codesets used by the Orbix 6.2 product. The following default codesets are defined:

- CORBA C++ codesets for non-MVS platforms.
- CORBA C++ codesets for MVS platform.
- CORBA Java codesets for US-ASCII locale.
- CORBA Java codesets for Shift_JIS locale.
- CORBA Java codesets for EUC-JP locale.
- CORBA Java codesets for other locales.

Native and conversion codesets

Native codesets are used by the application to pass `char` and `wchar` data to the ORB.

Conversion codesets are used, where necessary, to facilitate interoperability with other ORBs or platforms.

CORBA C++ codesets for non-MVS platforms

[Table 25](#) shows the default codesets for Orbix 6.2 C++ applications on non-MVS platforms (Latin-1 locale).

Table 25: CORBA C++ Codesets (Non-MVS Platforms)

Codeset Type	Codeset
Native codeset for char (NSC-C)	ISO-8859-1
Conversion codesets for char (CCS-C)	<i>none</i>
Native codeset for wchar (NCS-W)	UCS-2 or UCS-4
Conversion codesets for wchar (CCS-W)	UTF-16

In Orbix 6.2, the choice of native wide character codeset, UCS-2 or UCS-4, is based on the size of `CORBA::wchar` (either 2 or 4 bytes). UCS-2 is used on Windows. UCS-4 is used on most UNIX platforms.

CORBA C++ codesets for MVS platform

[Table 26](#) shows the default codesets for Orbix 6.2 C++ applications on the MVS platform.

Table 26: *CORBA C++ Codesets (MVS Platform)*

Codeset Type	Codeset
Native codeset for char (NSC-C)	EBCDIC
Conversion codesets for char (CCS-C)	ISO-8859-1
Native codeset for wchar (NCS-W)	UCS-2 or UCS-4
Conversion codesets for wchar (CCS-W)	UTF-16

CORBA Java codesets for US-ASCII locale

[Table 27](#) shows the codesets supported by Orbix 6.2 Java applications in a US-ASCII locale.

Table 27: *CORBA Java Codesets (ISO-8859-1/Cp-1292/US-ASCII locale)*

Codeset Type	Codeset
Native codeset for char (NSC-C)	ISO-8859-1
Conversion codesets for char (CCS-C)	UTF-8
Native codeset for wchar (NCS-W)	UTF-16
Conversion codesets for wchar (CCS-W)	UCS-2

CORBA Java codesets for Shift_JIS locale

[Table 28](#) shows the codesets supported by Orbix 6.2 Java applications in a Shift_JIS locale.

Table 28: *CORBA Java Codesets (Shift_JIS locale)*

Codeset Type	Codeset
Native codeset for char (NSC-C)	UTF-8
Conversion codesets for char (CCS-C)	ISO-8859-1 or Shift_JIS or euc_JP

Table 28: *CORBA Java Codesets (Shift_JIS locale)*

Codeset Type	Codeset
Native codeset for wchar (NCS-W)	UTF-16
Conversion codesets for wchar (CCS-W)	UCS-2 or Shift_JIS or euc_JP

CORBA Java codesets for EUC-JP locale

[Table 29](#) shows the codesets supported by Orbix 6.2 Java applications in an EUC-JP locale.

Table 29: *CORBA Java Codesets (EUC_JP locale)*

Codeset Type	Codeset
Native codeset for char (NSC-C)	UTF-8
Conversion codesets for char (CCS-C)	ISO-8859-1 or Shift_JIS or euc_JP
Native codeset for wchar (NCS-W)	UTF-16
Conversion codesets for wchar (CCS-W)	UCS-2 or Shift_JIS or euc_JP

CORBA Java codesets for other locales

[Table 30](#) shows the codesets supported by Orbix 6.2 Java applications in other locales..

Table 30: *CORBA Java Codesets (other locale)*

Codeset Type	Codeset
Native codeset for char (NSC-C)	UTF-8
Conversion codesets for char (CCS-C)	ISO-8859-1 or <i>file encoding</i>
Native codeset for wchar (NCS-W)	UTF-16
Conversion codesets for wchar (CCS-W)	UCS-2 or <i>file encoding</i>

Configuring Legacy Behavior

Default behavior

By default, the `IOP::TAG_CODE_SETS` tagged component is included in generated IORs and the transmission codesets are negotiated by clients and transmitted through an `IOP::CodeSets` service context. This is the CORBA-defined behavior.

Legacy behavior

Orbix 6.2 (all versions) also provides legacy behavior, to support the scenario where wide character data is communicated between Orbix 6.2 and Orbix 3.3 Java Edition.

Disabling codeset negotiation

The following configuration variable can be used to explicitly disable the codeset negotiation mechanism:

```
# Orbix 6.2 Configuration File
policies:giop:interop_policy:negotiate_transmission_codeset =
"false";
```

The default is `true`.

This is a proprietary setting provided for interoperability with legacy implementations, such as Orbix 3.3 Java Edition. The native codeset for character data, ISO-8859-1 (Latin-1), is used and the overhead of full negotiation is avoided. If wide character data is used, Orbix 6.2 reverts to the UTF-16 transmission codeset.

Enabling wchar transmission on a GIOP 1.0 connections

Passing `wchar` data over GIOP 1.0 can be enabled using the following configuration variable:

```
# Orbix 6.2 Configuration File
policies:giop:interop_policy:allow_wchar_types_in_1_0 = "true";
```

The default is `false`.

The transmission of `wchar` data is not legal in GIOP 1.0, by default.

`send_locate_request`

Part 3

Migrating from 5.1

In this part

This part contains the following chapters:

Upgrading from Mainframe Edition 5.x
--

page 277

Upgrading from Mainframe Edition 5.x

Migrating Orbix E2A Mainframe Edition 5.x-based applications to Orbix Mainframe 6.2 is a simpler process than migrating Orbix 2.3.x-based applications. Many differences that exist between Orbix 2.3.x and Orbix 6.2 do not exist between Orbix E2A 5.x and Orbix 6.2. Therefore, much fewer changes are required to migrate an Orbix E2A 5.x Mainframe Edition solution to Orbix Mainframe 6.2. This chapter outlines the requirements for upgrading from an Orbix E2A Mainframe Edition 5.x-based solution to Orbix Mainframe 6.2.

In this chapter

This chapter discusses the following topics:

Installation Requirements	page 279
Configuration Changes	page 281
New Node Daemon	page 289
Database Migration	page 290

C++ Migration	page 291
COBOL Migration	page 292
PL/I Migration	page 293

Installation Requirements

Overview

This section outlines the installation requirements for migrating from an Orbix E2A Mainframe Edition 5.x-based solution to Orbix Mainframe 6.2. It discusses the following topics:

- [Installing on native z/OS.](#)
 - [Installing on UNIX System Services.](#)
 - [Standard Customization Tasks.](#)
 - [Other Customization Tasks.](#)
 - [PDS names.](#)
-

Installing on native z/OS

Even though you have already installed a previous version of IONA's mainframe product, you must perform in full the tasks described in the 6.2 version of the *Mainframe Installation Guide* that pertain to installing on z/OS, because of the inherent differences between this and previous versions.

You must perform all these installation tasks in the order in which they are described in the *Mainframe Installation Guide*. Some tasks might not be relevant to your setup, but this is highlighted where appropriate.

Installing on UNIX System Services

If you choose to install Orbix Mainframe 6.2 on z/OS UNIX System Services as well as on native z/OS, you must perform in full the tasks described in the 6.2 version of the *Mainframe Installation Guide* that pertain to installing on z/OS UNIX System Services.

Standard Customization Tasks

After successfully installing Orbix Mainframe 6.2 on z/OS (and on z/OS UNIX System Services if you want), you must perform in full the standard customization tasks described in the 6.2 version of the *Mainframe Installation Guide*.

You must perform all these standard customization tasks in the order in which they are described in the *Mainframe Installation Guide*. Some tasks might not be relevant to your setup, but this is highlighted where appropriate. See [“Orbix Mainframe Configuration” on page 289](#) for customization details relating to your Orbix Mainframe configuration file.

Other Customization Tasks

Depending on your setup, there are additional customization tasks that you might also need to perform. These customization tasks relate to:

- Naming Service and Interface Repository customization.
- IMS adapter customization.
- CICS adapter customization.

If you need to perform any of these tasks, you must perform them in the order in which they are described in the *Mainframe Installation Guide*.

PDS names

In Orbix Mainframe 6.2, PDS naming conventions are different from those in Orbix E2A 5.x. The differences can be summarized as follows:

Table 31: *Differences in PDS Naming Conventions*

5.x	6.2
COBOL	CBL
JCL	JCLLIB
LIB	OBJLIB
LOAD	LOADLIB
LPA	LPALIB
PROCS	PROCLIB
RUN	LOADLIB

Configuration Changes

Overview

Orbix Mainframe 6.2 represents a major version upgrade, so Orbix 6.2 configuration is not backwards compatible with Orbix E2A 5.x configuration domains. This means that you cannot run Orbix 6.2 programs, using an Orbix E2A 5.x configuration file. This section outlines the changes that have been made to Orbix configuration, with particular emphasis on the configuration items relating to CICS and IMS integration.

This section discusses the following topics:

- [Migrating Core Orbix Configuration.](#)
- [Insecure deployments.](#)
- [Secure deployments.](#)
- [Migrating Your IMS or CICS Configuration.](#)
- [IMS Server Adapter Configuration Changes.](#)
- [CICS Server Adapter Configuration Changes.](#)
- [IMS and CICS client adapter configuration changes.](#)

Migrating Core Orbix Configuration

Many changes have been made to the core Orbix configuration infrastructure in Orbix 6.2. These changes relate to new or modified settings for shared library names, plug-in names, initial references, and other miscellaneous items. Because of the extents of these changes, there is no easy way to migrate an existing 5.x domain to the new Orbix 6.2 structure. The deployment phase for new configuration domains has been improved, however, to make the process more automated and to facilitate upgrades in the future. See [“Insecure deployments”](#) and [“Secure deployments”](#) next for more details. All feedback on the new configuration changes is welcome by contacting support@iona.com. IONA’s aim here is that, for future releases, customers will be able to simply replace the `HLQ.ORBIX62.CONFIG(ORXINTRL)` file without having to re-configure or copy your existing customized settings to IONA’s files.

Insecure deployments

In the Orbix 6.2 release, all internal settings are now stored in `HLQ.ORBIX62.CONFIG(ORXINTRL)`. The old `HLQ.ORBIX51.CONFIG(FILETMPL)` has now been renamed to `BASETMPL` in the `HLQ.ORBIX62.CONFIG` PDS. When you deploy an insecure configuration, `BASETMPL` is copied to `HLQ.ORBIX62.DOMAINS(FILEDOMA)` as in the 5.1 release. The `HLQ.ORBIX62.CONFIG(DEFAULT@)` member now includes both `HLQ.ORBIX62.DOMAINS(FILEDOMA)` and `HLQ.ORBIX62.CONFIG(ORXINTRL)`. See the *Mainframe Installation Guide* for more details of the customization tasks that are required for Orbix Mainframe 6.2.

Secure deployments

For a secure deployment, the process has been enhanced even further. The `HLQ.ORBIX62.CONFIG(TLSTMPL)` now only contains the specific TLS settings that you would need to use to make your system fully secure. By default, IONA now deploys a fully secure environment instead of a semi-secure environment. The new Security configuration also uses the concept of reopening scopes by sitting on top of the `ORXINTRL` and `BASETMPL` files. During the deployment process, `HLQ.ORBIX62.CONFIG(BASETMPL)` is copied to the `HLQ.ORBIX62.DOMAINS(TLSBASE)`, and `HLQ.ORBIX62.CONFIG(TLSTMPL)` is copied to `HLQ.ORBIX62.DOMAINS(TLSDOMA)`. The `HLQ.ORBIX62.CONFIG(DEFAULT@)` would then include `ORXINTRL`, `TLSBASE`, and `TLSDOMA`.

The `DEPLOY` process has also been enhanced so that you can provide the name of your keyring ring during deployment. This is done during the `MAKECON` step of the `HLQ.ORBIX62.JCLLIB(DEPLOYT)` job by updating the

LOCAL_SSL_USER_SAF_KEYRING to the name of your keyring. The corresponding configuration setting for this information has also been changed from `plugins:iiop_tls:racf_keyring` to `plugins:systemssl_toolkit:saf_keyring`.

See the *Mainframe Installation Guide* for more details of the customization tasks that are required for Orbix Mainframe 6.2.

Migrating Your IMS or CICS Configuration

Very few changes have been made to the configuration scopes that are specific to the IMS server adapter and CICS server adapter. Therefore, most of the customizations made in an Orbix E2A 5.x installation can be copied directly to an Orbix 6.2 configuration. This includes configuration items relating APPC, OTMA, XCF settings, and so on.

IMS and CICS client support has been re-designed and takes advantage of dynamic type support offered in the CICS and IMS server adapters. The `iona_services.mfu` scope has been removed and there are two new scopes: `iona_services.ims_client` and `iona_services.cics_client`. Configuration details are discussed below.

The default dynamic type support mechanism has been changed in Orbix 6.2 from the IFR to a file-based method. The following settings have been updated for CICS and IMS server adapters and for client adapters:

- IMS server adapters:


```
plugins:imsa:repository_id = "type_info";
plugins:imsa:type_info:source = "DD:TYPEINFO";
```
- CICS server adapters:


```
plugins:cicsa:repository_id = "type_info";
plugins:cicsa:type_info:source = "DD:TYPEINFO";
```
- Client adapters:


```
plugins:client_adapter:repository_id = "type_info";
plugins:client_adapter:type_info:source = "DD:TYPEINFO";
```

To use this feature you will need to pass the `-mfa:-inf` flag to the Orbix IDL compiler to generate type information. When you start the IMS server adapter, you will need to update your JCL so that the DD card `TYPEINFO` points to the data set where you stored your JCL. All Orbix Mainframe demonstrations are configured to use this process, so you can use any of them as an example. Alternatively, you can change the `repository_id` setting to `"ifr"` and remove the `type_info:source` setting, to continue using the IFR.

IMS Server Adapter Configuration Changes

The IMS server adapter is configured within the `iona_services.imsa` scope. The following configuration items have been modified since Orbix E2A 5.x:

`binding:client_binding_list` In Orbix E2A 5.x, this list contained bindings for the `ESIOP_IMS` interceptor. This interceptor is not used by the IMS server adapter in Orbix 6.2. Therefore, in general, there is no need to specify this variable anymore within the `imsa` scope. You can use the setting from the global scope instead.

`initial_references:IT_MFA:reference` In Orbix E2A 5.x, this reference was set in the `iona_services.imsa` scope. In Orbix 6.2, this setting is now in the new `iona_utilities.imsa` scope for use by clients of the IMS server adapter (for example, `itadmin/ORXADMIN` clients).

The following configuration items are new in Orbix 6.2:

`mf_subsystems` This specifies the Orbix Mainframe subsystem that is in use. In this case, it must be set to `"adapter"`. This configuration item is required. The IMS server adapter cannot start if this item is not set to `"adapter"`.

`plugins:imsa:imsraw_api_support` This can be used to expose the legacy, unscoped `imsraw` API. This item is optional, and the default is to expose the scoped `IT_MFA_IMS::imsraw` API. Valid values are `scoped`, `unscoped`, and `both`.

`plugins:ims_otma:use_sync_level_one` This allows you to disable sync level one processing in the IMS server adapter's communications with IMS over OTMA. This item is optional, and the default is to use sync level one processing. Valid values are `"true"` and `"false"`.

```
plugins:imsa:check_security_
  credentials
```

To illustrate integration with the IONA Security Framework (ISF), a sample iS2 configuration domain is included in the TLS template configuration. This variable is used to instruct the IMS server adapter to check for received credentials, to determine the user ID to be used for performing SAF checks.

This item should only be used in an iS2-enabled configuration with the `use_client_principal` setting. This item is optional, and the default is to not check security credentials. Valid values are `"true"` and `"false"`.

The following configuration item has been deprecated in Orbix 6.2:

```
plugins:portable_interceptor:
  additional_dlls
```

This was used in Orbix E2A 5.x to enable an existing Orbix program to load a DLL containing a portable interceptor. This item is no longer supported. See the *IMS Adapters Administrator's Guide* for more details about how to add a portable interceptor to the IMS server adapter in Orbix 6.2.

CICS Server Adapter Configuration Changes

The CICS server adapter is configured within the `iona_services.cicsa` scope.

The following configuration items have been modified since Orbix E2A 5.x:

`binding:client_binding_list` In Orbix E2A 5.x, this list contained bindings for the `ESIOP_CICS` interceptor. This interceptor is not used by the CICS server adapter in Orbix 6.2. Therefore, in general, there is no need to specify this variable anymore within the `cicsa` scope. You can use the setting from the global scope instead.

`initial_references:IT_MFA:reference` In Orbix E2A 5.x, this reference was set in the `iona_services.cicsa` scope. In Orbix 6.2, this setting is now in the new `iona_utilities.cicsa` scope for use by clients of the CICS server adapter (for example, `itadmin/ORXADMIN` clients).

The following configuration items are new in Orbix 6.2:

`mf_subsystems` This specifies the Orbix Mainframe subsystem that is in use. In this case, it must be set to `"adapter"`. This configuration item is required. The CICS server adapter cannot start if this item is not set to `"adapter"`.

`plugins:cicsa:cicsraw_api_support` This can be used to expose the legacy, unscoped `cicsraw` API. This item is optional, and the default is to expose the scoped `IT_MFA_CICS::cicsraw` API. Valid values are `scoped`, `unscoped`, and `both`.

<pre>plugins:cics_exci:check_if_cics_ available</pre>	<p>In Orbix E2A 5.x, the EXCI version of the CICS server adapter automatically attempted to contact the CICS subsystem upon starting. In Orbix 6.2, you can set this item to "true" to maintain this behavior. This item is optional, and the default is to not have the adapter check to see if CICS is available upon starting. Valid values are "true" and "false".</p>
<pre>plugins:cicsa:check_security_ credentials</pre>	<p>To illustrate integration with the IONA Security Framework (ISF), a sample iS2 configuration domain is included in the TLS template configuration. This variable is used to instruct the CICS server adapter to check for received credentials, to determine the user ID to be used for performing SAF checks.</p> <p>This item should only be used in an iS2-enabled configuration with the <code>use_client_principal</code> setting. This item is optional, and the default is to not check security credentials. Valid values are "true" and "false".</p>

The following configuration item has been deprecated in Orbix 6.2:

<pre>plugins:portable_interceptor: additional_dlls</pre>	<p>This was used in Orbix E2A 5.x to enable an existing Orbix program to load a DLL containing a portable interceptor. This item is no longer supported. See the <i>CICS Adapters Administrator's Guide</i> for more details about how to add a portable interceptor to the CICS server adapter in Orbix 6.2.</p>
--	---

IMS and CICS client adapter configuration changes

With Orbix Mainframe 6.2, the client adapter has been refactored into a subsystem that can be loaded either as a standalone process or alongside the CICS and IMS server adapters. By default, it is loaded as a standalone process. This is controlled by the `mf_subsystems = ["client_adapter"]` configuration item within the `orbixhlq.CONFIG(ORXINTRL)` configuration file. You can choose to load the "client_adapter" subsystem alongside the "adapter" subsystem, to have one process act as both a server adapter and a client adapter. For more information see the *CICS Adapters Administrator's Guide* or the *IMS Adapters Administrator's Guide*, or contact support@iona.com.

Technically speaking, you could have an IMS client talk to a CICS client adapter, or alternatively have a CICS client talk to an IMS client adapter, as long as the client adapter was configured to listen on the correct LU. The demonstration configuration breaks them into two to provide a symmetrical example and, as a convenience for users who might not want to have one client adapter talking to both IMS and CICS.

Sample JCL to run the new client adapters is provided in `HLQ.ORBIX62.JCLLIB(CICSCA)` and `HLQ.ORBIX62.JCLLIB(IMSCA)`.

New Node Daemon

Overview

Orbix Mainframe 6.2 features a new node daemon, which has been modified to provide more reliable monitoring of server processes. This section outlines the various migration issues that this gives rise to. It discusses the following topics:

- [Wider deployment of node daemons.](#)
- [Incompatibility with old server binaries.](#)
- [Incompatibility of node daemon database.](#)

Wider deployment of node daemons

When upgrading your system to Orbix 6.2, it might be necessary to deploy a node daemon to some hosts where, previously, none was required.

Prior to Orbix 6.0, a node daemon was required on a host only if you needed the capability to automatically start (or restart) a CORBA server in response to incoming invocations. Monitoring the state of a server process could be performed by a single central node daemon, which monitored the server through a remote connection.

With Orbix 6.2, a node daemon is required on every machine that hosts servers with persistent POAs (a persistent POA is a POA whose `PortableServer::LifespanPolicy` is set to `PERSISTENT`). Monitoring the state of a server process through a local node daemon is more reliable than monitoring by a remote node daemon.

Incompatibility with old server binaries

Because the internal service interfaces for the locator, node daemon, and POA have changed significantly, the new node daemon is incompatible with old (pre-Orbix 6.0) server binaries. It is, therefore, necessary to rebuild old application binaries before deploying them to an Orbix 6.2 configuration domain.

Incompatibility of node daemon database

You cannot copy an old node daemon database (usually located in `ASPIInstallDir/var/DomainName/dbs/node_daemon`) to a new Orbix 6.2 node daemon database, because the node daemon database schema has changed significantly in Orbix 6.2.

Database Migration

Migration impact

You cannot copy Orbix E2A 5.1 database files to a new Orbix 6.2 database structure, because the database schemas have changed significantly. As a result, you must re-deploy your applications and re-register any named keys, naming service entries, and IFR entries accordingly.

C++ Migration

C++ runtime support

Orbix Mainframe 6.2 supports the IBM z/OS V1.4 ANSI C++ compiler, the z/OS 1.5 ANSI C++ compiler, and the z/OS 1.6 ANSI C++ compiler. Because IBM no longer supports the OS/390 V2R10 compiler, IONA has also removed support and certification for this compiler in this release.

Environment targets

Orbix Mainframe 6.2 supplies a C++ options file in `HLQ.ORBIX62.CONFIG(ORXCppo)` that includes the "TARGET" compiler option of "ZOSV1R2". The "TARGET" compiler option enables the development of applications on higher release levels of z/OS for subsequent use on platforms that are running lower release levels of z/OS or OS/390. You can update the `ORXCppo` member with "ZOSV1R4", "ZOSV1R5" or "ZOSV1R6", as appropriate. Alternatively, you can remove the "TARGET" option from the `ORXCppo` member. This means that it uses the "CURRENT" option by default, which corresponds to the z/OS release on which your system is running. For more information see the IBM publication: *C/C++ User's Guide*.

Compiler options

Orbix 6.2 uses the Run Time Type Identification (RTTI) z/OS compiler option. This allows you to take advantage of C++ dynamic casting. In previous releases of Orbix, this option was not used because the OS/390 V2R10 compiler did not support it.

Migrating your C++ applications

Orbix Mainframe 6.2 is not binary compatible with Orbix Mainframe 5.1. You must therefore rebuild your C++ applications when migrating to Orbix Mainframe 6.2. To do this you must do the following:

1. The IDL configuration file in `HLQ.ORBIX62.CONFIG(IDL)` no longer includes the "-xRTTI" option on the `PresetOptions`. If you are using a local copy of this file, ensure that you make the appropriate updates.
2. Update your compiler options to use RTTI. The `HLQ.ORBIX62.PROCLIB(ORXCppo)` file contains this option by default. You may use it as an example.
3. Regenerate your IDL stub code and recompile your application.

COBOL Migration

Migration of applications

It is not necessary to fully rebuild your COBOL applications, to migrate them from Orbix E2A Mainframe Edition 5.x to Orbix Mainframe 6.2. To migrate your COBOL applications:

1. Re-link your applications with the Orbix 6.2 libraries. You do not need to recompile them.
 2. Update any JCL that you have stored in non-IONA libraries, to ensure that your applications subsequently compile and link correctly with Orbix Mainframe 6.2.
-

Generation of mapping files

In previous versions of the Orbix COBOL generator, if the `-M` option was specified and the IDL had operation names that were identical in several interfaces, no warning was produced if the names mapped to a non-unique name. For example, no warning was produced if the generated mapping file contained:

```
interfaceA/ping ping
interfaceB/ping ping
```

The Orbix 6.2 COBOL generator will still generate the preceding mapping file but also outputs a warning about the generated mapping file. The generator will also produce a return code of 4, to alert the developer that two or more operations have been mapped to the same name.

PL/I Migration

Migration of applications

It is not necessary to fully rebuild your PL/I applications, to migrate them from Orbix E2A Mainframe Edition 5.x to Orbix Mainframe 6.2. To migrate your PL/I applications:

1. Re-link your applications with the Orbix 6.2 libraries. You do not need to recompile them.
 2. Update any JCL that you have stored in non-IONA libraries, to ensure that your applications subsequently compile and link correctly with Orbix Mainframe 6.2.
-

Inherited interfaces

The IDL-PL/I generator now generates only one instance of a PL/I typedef per IDL type. In previous releases, if a type was inherited, the PL/I generator created a typedef for both the base class's instance of the type and also one for each inherited type. This was unnecessary as both generated typedefs would always be the same, apart from the name of the typedef. It also resulted in the generation of large include files in the cases of IDL with complex structs, for example. For programs where a pre-Orbix 6.2 generated server implementation is used and new include files need to be generated, the `-Li` option has been introduced.

Orbix PL/I include file re-arrangement

Three PL/I include members (`CORBA`, `READIOR` and `SETUPCL`) have been reorganized, to decrease the number of instances where the compilation of an Orbix PL/I program results in a return code of 4, due to the pre-processor check for `client_only`. The reorganization has been designed so that there would not be a migration hit for existing Orbix PL/I applications. Additionally, a new include file, `SETUPSV`, has been added, to declare `client_only` and set it to `"NO"` in Orbix PL/I server applications. For further details about the include members, see the *Orbix PL/I Programmers Guide and Reference*.

Generation of mapping files

In previous versions of the Orbix PL/I generator, if the `-m` option was specified and the IDL had operation names that were identical in several interfaces, no warning was produced if the names mapped to a non-unique name. For example, no warning was produced if the generated mapping file contained:

```
interfaceA/ping ping
interfaceB/ping ping
```

The Orbix 6.2 PL/I generator will still generate the preceding mapping file but also outputs a warning about the generated mapping file. The generator will also produce a return code of 4, to alert the developer that two or more operations have been mapped to the same name.

Part 4

Migrating from 6.0

In this part

This part contains the following chapters:

Upgrading from Orbix Mainframe 6.0
--

page 299

Upgrading from Orbix Mainframe 6.0

Migrating Orbix 6.0 applications to Orbix Mainframe 6.2 is a simpler process than migrating Orbix 2.3.x-based applications. Many differences that exist between Orbix 2.3.x and Orbix 6.2 do not exist between Orbix 6.0 and Orbix 6.2. Therefore, much fewer changes are required to migrate an Orbix Mainframe 6.0 solution to Orbix Mainframe 6.2. This chapter outlines the requirements for upgrading from Orbix Mainframe 6.0 to Orbix Mainframe 6.2.

In this chapter

This chapter discusses the following topics:

Installation Requirements	page 301
Configuration Changes	page 303
Database Migration	page 308
C++ Migration	page 309
COBOL Migration	page 311

PL/I Migration	page 312
JCL Updates	page 314

Installation Requirements

Overview

This section outlines the installation requirements for migrating from an Orbix Mainframe 6.0 solution to Orbix Mainframe 6.2. It discusses the following topics:

- [Installing on native z/OS.](#)
 - [Installing on UNIX System Services.](#)
 - [Standard Customization Tasks.](#)
 - [Other Customization Tasks.](#)
 - [PDS names.](#)
-

Installing on native z/OS

Even though you have already installed a previous version of IONA's mainframe product, you must perform in full the tasks described in the 6.2 version of the *Mainframe Installation Guide* that pertain to installing on z/OS, because of the inherent differences between this and previous versions.

You must perform all these installation tasks in the order in which they are described in the *Mainframe Installation Guide*. Some tasks might not be relevant to your setup, but this is highlighted where appropriate.

Installing on UNIX System Services

If you choose to install Orbix Mainframe 6.2 on z/OS UNIX System Services as well as on native z/OS, you must perform in full the tasks described in the 6.2 version of the *Mainframe Installation Guide* that pertain to installing on z/OS UNIX System Services.

Standard Customization Tasks

After successfully installing Orbix Mainframe 6.2 on z/OS (and on z/OS UNIX System Services if you want), you must perform in full the standard customization tasks described in the 6.2 version of the *Mainframe Installation Guide*.

You must perform all these standard customization tasks in the order in which they are described in the *Mainframe Installation Guide*. Some tasks might not be relevant to your setup, but this is highlighted where appropriate. See [“Orbix Mainframe Configuration” on page 289](#) for customization details relating to your Orbix Mainframe configuration file.

Other Customization Tasks

Depending on your setup, there are additional customization tasks that you might also need to perform. These customization tasks relate to:

- Naming Service and Interface Repository customization.
- IMS adapter customization.
- CICS adapter customization.

If you need to perform any of these tasks, you must perform them in the order in which they are described in the 6.2 version of the *Mainframe Installation Guide*.

PDS names

In Orbix Mainframe 6.2, PDS naming conventions are different from those in Orbix Mainframe 6.0. The differences can be summarized as follows:

Table 32: *Differences in PDS Naming Conventions*

6.0	6.2
COBOL	CBL
JCL	JCLLIB
LIB	OBJLIB
LOAD	LOADLIB
LPA	LPALIB
PROCS	PROCLIB
RUN	LOADLIB

Configuration Changes

Overview

Orbix Mainframe 6.2 represents a minor version upgrade, so binary compatibility has been maintained with the Orbix Mainframe 6.0 release. Although this is a minor version upgrade, there have been many changes to the configuration files which are not backwards compatible with Orbix 6.0. This means that you cannot run Orbix 6.2 programs, using an Orbix 6.0 configuration file. The changes made to the Orbix 6.2 configuration files have been designed to facilitate future migrations. This section outlines the changes that have been made to Orbix 6.2 configuration.

This section discusses the following topics:

- [Migrating Core Orbix Configuration.](#)
- [Insecure deployments.](#)
- [Secure deployments.](#)
- [Migrating Your IMS or CICS Configuration.](#)
- [IMS Server Adapter Configuration Changes.](#)
- [CICS Server Adapter Configuration Changes.](#)
- [IMS and CICS client adapter configuration changes.](#)

Migrating Core Orbix Configuration

Many changes have been made to the core Orbix configuration infrastructure in Orbix 6.2. These changes relate to new or modified settings for shared library names, plug-in names, initial references, and other miscellaneous items. Because of the extents of these changes, there is no easy way to migrate an existing 6.0 domain to the new Orbix 6.2 structure. The deployment phase for new configuration domains has been improved, however, to make the process more automated and to facilitate upgrades in the future. See [“Insecure deployments”](#) and [“Secure deployments”](#) next for more details. All feedback on the new configuration changes is welcome by contacting support@iona.com. IONA’s aim here is that, for future releases, customers will be able to simply replace the `HLQ.ORBIX62.CONFIG(ORXINTRL)` file without having to re-configure or copy your existing customized settings to IONA’s files.

Insecure deployments

In the Orbix 6.2 release, all internal settings are now stored in `HLQ.ORBIX62.CONFIG(ORXINTRL)`. The old `HLQ.ORBIX60.CONFIG(FILETMPL)` has now been renamed to `BASETMPL` in the `HLQ.ORBIX62.CONFIG` PDS. When you deploy an insecure configuration, `BASETMPL` is copied to `HLQ.ORBIX62.DOMAINS(FILEDOMA)` as in the 6.0 release. The `HLQ.ORBIX62.CONFIG(DEFAULT@)` member now includes both `HLQ.ORBIX62.DOMAINS(FILEDOMA)` and `HLQ.ORBIX62.CONFIG(ORXINTRL)`. See the *Mainframe Installation Guide* for more details of the customization tasks that are required for Orbix Mainframe 6.2.

Secure deployments

For a secure deployment, the process has been enhanced even further. The `HLQ.ORBIX62.CONFIG(TLSTMPL)` now only contains the specific TLS settings that you would need to use to make your system fully secure. By default, IONA now deploys a fully secure environment instead of a semi-secure environment. The new Security configuration also uses the concept of reopening scopes by sitting on top of the `ORXINTRL` and `BASETMPL` files. During the deployment process, `HLQ.ORBIX62.CONFIG(BASETMPL)` is copied to the `HLQ.ORBIX62.DOMAINS(TLSBASE)`, and `HLQ.ORBIX62.CONFIG(TLSTMPL)` is copied to `HLQ.ORBIX62.DOMAINS(TLSDOMA)`. The `HLQ.ORBIX62.CONFIG(DEFAULT@)` would then include `ORXINTRL`, `TLSBASE`, and `TLSDOMA`.

The `DEPLOY` process has also been enhanced so that you can provide the name of your keyring ring during deployment. This is done during the `MAKECON` step of the `HLQ.ORBIX62.JCLLIB(DEPLOYT)` job by updating the

LOCAL_SSL_USER_SAF_KEYRING to the name of your keyring. The corresponding configuration setting for this information has also been changed from `plugins:iiop_tls:racf_keyring` to `plugins:systemssl_toolkit:saf_keyring`.

See the *Mainframe Installation Guide* for more details of the customization tasks that are required for Orbix Mainframe 6.2.

Migrating Your IMS or CICS Configuration

Very few changes have been made to the configuration scopes that are specific to the IMS server adapter and CICS server adapter. Therefore, most of the customizations made in an Orbix 6.0 installation can be copied directly to an Orbix 6.2 configuration. This includes configuration items relating APPC, OTMA, XCF settings, and so on.

IMS and CICS client support has been re-designed and takes advantage of dynamic type support offered in the CICS and IMS server adapters. The `iona_services.mfu` scope has been removed and there are two new scopes: `iona_services.ims_client` and `iona_services.cics_client`. Configuration details are discussed below.

The default dynamic type support mechanism has been changed in Orbix 6.2 from the IFR to a file-based method. The following settings have been updated for CICS and IMS server adapters and for client adapters:

- IMS server adapters:

```
plugins:imsa:repository_id = "type_info";
plugins:imsa:type_info:source = "DD:TYPEINFO";
```

- CICS server adapters:

```
plugins:cicsa:repository_id = "type_info";
plugins:cicsa:type_info:source = "DD:TYPEINFO";
```

- Client adapters:

```
plugins:client_adapter:repository_id = "type_info";
plugins:client_adapter:type_info:source = "DD:TYPEINFO";
```

To use this feature you will need to pass the `-mfa:-inf` flag to the Orbix IDL compiler to generate type information. When you start the IMS server adapter, you will need to update your JCL so that the DD card `TYPEINFO` points to the data set where you stored your JCL. All Orbix Mainframe demonstrations are configured to use this process, so you can use any of them as an example. Alternatively, you can change the `repository_id` setting to `ifr` and remove the `type_info:source` setting, to continue using the IFR.

IMS Server Adapter Configuration Changes

No new configuration items have been introduced or modified for IMS since version 6.0. The following configuration item has been deprecated in Orbix 6.2:

`plugins:portable_interceptor:
additional_dlls` This was used in Orbix 6.0 to enable an existing Orbix program to load a DLL containing a portable interceptor. This item is no longer supported. See the *IMS Adapters Administrator's Guide* for more details about how to add a portable interceptor to the IMS server adapter in Orbix 6.2.

CICS Server Adapter Configuration Changes

No new configuration items have been introduced or modified for CICS since version 6.0. The following configuration item has been deprecated in Orbix 6.2:

`plugins:portable_interceptor:
additional_dlls` This was used in Orbix 6.0 to enable an existing Orbix program to load a DLL containing a portable interceptor. This item is no longer supported. See the *CICS Adapters Administrator's Guide* for more details about how to add a portable interceptor to the CICS server adapter in Orbix 6.2.

IMS and CICS client adapter configuration changes

With Orbix Mainframe 6.2, the client adapter has been refactored into a subsystem that can be loaded either as a standalone process or alongside the CICS and IMS server adapters. By default, it is loaded as a standalone process. This is controlled by the `mf_subsystems = ["client_adapter"]` configuration item within the `orbixhlq.CONFIG(ORXINTRL)` configuration file. You can choose to load the "client_adapter" subsystem alongside the "adapter" subsystem, to have one process act as both a server adapter and a client adapter. For more information see the *CICS Adapters Administrator's Guide* or the *IMS Adapters Administrator's Guide*, or contact support@iona.com.

Technically speaking, you could have an IMS client talk to a CICS client adapter, or alternatively have a CICS client talk to an IMS client adapter, as long the client adapter was configured to listen on the correct LU. The demonstration configuration breaks them into two to provide a symmetrical example and, as a convenience for users who might not want to have one client adapter talking to both IMS and CICS.

Sample JCL to run the new client adapters is provided in `HLQ.ORBIX62.JCLLIB(CICSCA)` and `HLQ.ORBIX62.JCLLIB(IMSCA)`.

Database Migration

Migration impact

In Orbix 6.0, when you deployed your locator and node daemon, several databases were automatically created on z/OS UNIX System Services with the following naming conventions:

- `%{LOCAL_HFS_ROOT}/filedomain/dbs/locator`
- `%{LOCAL_HFS_ROOT}/filedomain/dbs/locator_priv`
- `%{LOCAL_HFS_ROOT}/filedomain/dbs/node_daemon`

If you also deployed an IFR and a Naming Service, the following were also created:

- `%{LOCAL_HFS_ROOT}/filedomain/dbs/ifr`
- `%{LOCAL_HFS_ROOT}/filedomain/dbs/naming`

These directory paths contain the database files corresponding to the relevant service. On upgrading to Orbix 6.2, you can choose to either copy these files or use them directly by using the same `LOCAL_HFS_ROOT` in 6.2 that you also used in 6.0. There have been some database schema changes made internally in Orbix 6.2. Once Orbix 6.2 uses your Orbix 6.0 files for the first time, Orbix 6.0 cannot use those files again. If you need to run Orbix 6.0 and Orbix 6.2 in parallel, copy the files by copying `%{LOCAL_HFS_ROOT}/filedomain/dbs` and all of its sub-components to a new location, and then update the `LOCAL_HFS_ROOT` in Orbix 6.2 accordingly. If you are migrating directly to Orbix 6.2, IONA recommends that you make a backup copy of your files before you begin.

C++ Migration

C++ runtime support

Orbix Mainframe 6.2 supports the IBM z/OS V1.4 ANSI C++ compiler, the z/OS 1.5 ANSI C++ compiler, and the z/OS 1.6 ANSI C++ compiler. Because IBM no longer supports the OS/390 V2R10 compiler, IONA has also removed support and certification for this compiler in this release.

Environment targets

Orbix Mainframe 6.2 supplies a C++ options file in `HLQ.ORBIX62.CONFIG(ORXCPP0)` that includes the "TARGET" compiler option of "ZOSV1R2". The "TARGET" compiler option enables the development of applications on higher release levels of z/OS for subsequent use on platforms that are running lower release levels of z/OS or OS/390. You can update the `ORXCPP0` member with "ZOSV1R4", "ZOSV1R5" or "ZOSV1R6", as appropriate. Alternatively, you can remove the "TARGET" option from the `ORXCPP0` member. This means that it uses the "CURRENT" option by default, which corresponds to the z/OS release on which your system is running. For more information see the IBM publication: *C/C++ User's Guide*.

Compiler options

Orbix 6.2 uses the Run Time Type Identification (RTTI) z/OS compiler option. This allows you to take advantage of C++ dynamic casting. In previous releases of Orbix, this option was not used because the OS/390 V2R10 compiler did not support it.

Migrating your C++ applications

Orbix 6.2 is binary compatible with Orbix 6.0. However, to take advantage of the C++ RTTI feature, binary compatibility had to be broken. Therefore, you must rebuild and recompile your C++ applications to use the RTTI feature. To do this you must do the following:

1. The IDL configuration file in `HLQ.ORBIX62.CONFIG(IDL)` no longer includes the `"-xRTTI"` option on the `PresetOptions`. If you are using a local copy of this file, ensure that you make the appropriate updates.
2. Update your compiler options to use RTTI. The `HLQ.ORBIX62.PROCLIB(ORXCPP0)` file contains this option by default. You may use it as an example.
3. Regenerate your IDL stub code and recompile your application.

COBOL Migration

Migration of COBOL applications

It is not necessary to rebuild your COBOL applications when migrating from Orbix Mainframe 6.0 to Orbix Mainframe 6.2. COBOL applications built with Orbix Mainframe 6.0 will run without any updates in an Orbix Mainframe 6.2 runtime environment.

Generation of mapping files

In previous versions of the Orbix COBOL generator, if the `-m` option was specified and the IDL had operation names that were identical in several interfaces, no warning was produced if the names mapped to a non-unique name. For example, no warning was produced if the generated mapping file contained:

```
interfaceA/ping  ping
interfaceB/ping  ping
```

The Orbix 6.2 COBOL generator will still generate the preceding mapping file but also outputs a warning about the generated mapping file. The generator will also produce a return code of 4, to alert the developer that two or more operations have been mapped to the same name.

PL/I Migration

Migration of PL/I applications

It is not necessary to rebuild your PL/I applications when migrating from Orbix Mainframe 6.0 to Orbix Mainframe 6.2. PL/I applications built with Orbix Mainframe 6.0 will run without any updates in an Orbix Mainframe 6.2 runtime environment.

Inherited interfaces

The IDL-PL/I generator now generates only one instance of a PL/I typedef per IDL type. In previous releases, if a type was inherited, the PL/I generator created a typedef for both the base class's instance of the type and also one for each inherited type. This was unnecessary as both generated typedefs would always be the same, apart from the name of the typedef. It also resulted in the generation of large include files in the cases of IDL with complex structs, for example. For programs where a pre-Orbix 6.2 generated server implementation is used and new include files need to be generated, the `-Li` option has been introduced.

Orbix PL/I include file re-arrangement

Three PL/I include members (`CORBA`, `READIOR` and `SETUPCL`) have been reorganized, to decrease the number of instances where the compilation of an Orbix PL/I program results in a return code of 4, due to the pre-processor check for `client_only`. The reorganization has been designed so that there would not be a migration hit for existing Orbix PL/I applications. Additionally, a new include file, `SETUPSV`, has been added, to declare `client_only` and set it to `"no"` in Orbix PL/I server applications. For further details about the include members, see the *Orbix PL/I Programmers Guide and Reference*.

Generation of mapping files

In previous versions of the Orbix PL/I generator, if the `-M` option was specified and the IDL had operation names that were identical in several interfaces, no warning was produced if the names mapped to a non-unique name. For example, no warning was produced if the generated mapping file contained:

```
interfaceA/ping ping
interfaceB/ping ping
```

The Orbix 6.2 PL/I generator will still generate the preceding mapping file but also outputs a warning about the generated mapping file. The generator will also produce a return code of 4, to alert the developer that two or more operations have been mapped to the same name.

JCL Updates

Migratoin

If you are using your own customized JCL or JCL procedures, you will need to update them with the following details to use Orbix 6.2:

1. PDS names have changed in Orbix 6.2, as outlined in [“PDS names” on page 302](#). As a result, you will need to update any of your own JCL that refers to these.
2. As discussed in [“C++ Migration” on page 309](#), IONA requires that you rebuild your C++ applications to take advantage of the RTTI option. Ensure that the `TARGET` compiler option is set accordingly.
3. The Orbix IDL compiler has been updated to read IDL options from a DD card, called `IDLARGS`. To take advantage of this new feature, you will need to add the `IDLARGS` DD to your JCL. As an example, please see the supplied `ORXIDL` procedure in `HLQ.ORBIX62.PROCLIB(ORXIDL)`, and any of the demonstration IDL compilations (that is, `HLQ.ORBIX62.DEMOS.CBL.BLD.JCLLIB(SIMPLIDL)`).
4. If you are processing transactions in a C++ client or server (that is, two-phase commit processing), you will need to link in the following side decks to your program:

```
//          DD  DSN=&ORBIX..LKED(ORXOTS) ,DISP=SHR
//          DD  DSN=&ORBIX..LKED(ORXOTSP) ,DISP=SHR
```

See `HLQ.ORBIX62.PROCLIB(ORXLINK)` for an example.

5. The `ARITH(EXTEND)` option has been added to the supplied COBOL compilation procedures (that is, `ORXCBCCC`, `ORXCBCSC`, `ORXCBLCC`, `ORXCBLSC`) to support arithmetic extended types. This is required if you are using fixed types greater than 18 digits. You only need to update this if you plan on using fixed types greater than 18 digits.
6. The supplied PL/I compilation procedures have been enhanced to accommodate the enterprise PL/I compiler. Support has also been added for fixed 31 and long long support. To use these features you will need to update the PL/I compiler options:


```
"LIMITS(FIXEDDEC(31),FIXEDBIN(63))' ".
```

 Sample procedures can be

found in

HLQ.ORBIX62.PROCLIB(ORXPLCCC,ORXPLCSC,ORXPLICC,ORXPLISC). For an example of usage, refer to any of the PL/I demonstrations (for example, HLQ.ORBIX62.DEMOS.PLI.BLD.JCLLIB(SIMPLESB)). To use the new enterprise compiler procedures, change the "PLIPRFX" to "IBMZ".

7. A new procedure called ORXICONV has been added to facilitate converting files from one code page to another. Currently the procedure is designed to convert members of a PDS. See HLQ.ORBIX62.JCLLIB(UPDLICEN) for an example.
8. The MFACL JCL is no longer shipped in HLQ.ORBIX62.JCLLIB. This is due to the re-design of the client adapters discussed in ["Configuration Changes" on page 303](#). If you have your own JCL to start your client adapter, you will need to use the IMS or CICS server adapter and pass in the ORB name with your client configuration details. See ["Configuration Changes" on page 303](#) for more details on these re-design and configuration changes.

Part 5

Migrating from Artix Mainframe Developer 2.0

In this part

This part contains the following chapters:

Upgrading from Artix Mainframe Developer 2.0	page 319
--	--------------------------

Upgrading from Artix Mainframe Developer 2.0

The Artix Mainframe Developer 2.0 product has been incorporated into Orbix Mainframe 6.2. This chapter outlines the requirements for upgrading from Artix Mainframe Developer 2.0 to Orbix Mainframe 6.2.

In this chapter

This chapter discusses the following topics:

- [“Existing applications” on page 320.](#)
- [“imsraw and cicsraw” on page 320.](#)
- [“Configuration changes” on page 320.](#)
- [“Demonstration code” on page 320.](#)

Existing applications

If you have existing applications in Artix Mainframe Developer 2.0, you must recompile your IDL files to regenerate the file-based type information store. Also, ensure that your mapping file is up to date.

imsraw and cicsraw

If you are using `imsraw` or `cicsraw`, run the `PREPSOAP` job in `HLQ.ORBIX62.JCLLIB` to regenerate the type information for those endpoints.

Configuration changes

The Artix configuration file is now shipped in `HLQ.ORBIX62.CONFIG(ARTIX)`. Like the configuration changes made to Orbix 6.2, the Artix configuration file also takes advantage of re-opening existing name spaces. To use the IMS or CICS server adapter over HTTP, ensure that the comment character is removed from the following line in your deployed configuration domain:

```
#include "HLQ.ORBIX62.CONFIG(ARTIX)";
```

Update the configuration file with any specific changes for your site. You can then start your IMS or CICS server adapters accordingly.

Note: In a future release, the Artix configuration process will be refined further, so that a copy is stored in the `DOMAINS` PDS. If you have suggestions on how you would like to see this process defined, contact support@iona.com.

Demonstration code

Artix demonstration code is supplied with your Orbix Mainframe 6.2 installation in `HLQ.ORBIX62.DEMOS.ARTIX.*`.

Index

A

addr(IFNAME_user_exceptions) 215
ATM 59
AutomaticWorkQueue policy 55

B

binary compatibility 10
bindUsingIOP() 57
BOA
 activation modes 45
 and Orbix loaders 44
 implementation 46
 servers 48

C

callback objects 39
CBLTDLI 170
CERRSMFA 177
CHECK_ERRORS
 CICS clients 243
 IMS clients 235
 PLI 204
CHECK-STATUS paragraph
 CICS 177
 IMS 168
CHKCLCIC 179
CHKCLIMS 168, 235
CHKERRS 204
 CHECK-STATUS paragraph 125
 CICS equivalent 175
CICS COBOL clients
 error checking 179
 extra copybook 180
CICS PLI client migration issues 243
CloseConnection message 59
COBOL keywords 73, 193
 IDL identifier names D and U 120
 module and interface names 117
code generation toolkit 11
command-line arguments
 and gencl 181
 and genpli 244

COMM_FAILURE exception 38, 59
compile errors 137
configuration
 IIOF 58
 ORB class 57
 thread pools 54
configuration files 22
connection management 58
constant definitions See IDL constant definitions
conversion functions
 PL/I 210
copybook names 89
COPY statement 92, 94
CORBA::ORB 37
CORBA::Orbix.setDiagnostics() 248
CORBA::Orbix object 37
CORBA::Request::operator 41
CORBA copybook 137
CORBA Environment parameter 36
CORBA include member 211
corbaloc
 C++ 31
 COBOL 140
 PLI 206

D

data names 12
 constant definitions (PLI) 192
 IDL compiler 185
 length of (PLI) 188
 uniqueness of 195
 _default_POA() 44
Derived Interface Names 77
destroy() 37
diagnostic output 248
DII calls 41
DISPATCH reference 181
DISPINIT membe 217
DISPINIT Member contents 221
DLIDATA 234
dynamic invocation interface 41

E

Enterprise COBOL compiler
 container names 98
 fieldnames 105
 name scoping 97
 Environment parameter, CORBA 36
 event_log filters 248
 exception handling 36
 exceptions
 and PODSTAT 214
 COMM_FAILURE 38, 59
 INV_OBJREF 37
 no_user_exceptions 215
 runtime reporting of 148
 TRANSIENT 38

F

fabricated object references 140
 factory object 46
 file descriptors 53
 connection management 58
 filters, event_log 248
 filters, Orbix 52
 fixed block data sets 22
 fixed type definitions 21
 FQN
 COBOL data names 64
 derived interface names 77
 IDL constant definitions (COBOL) 73
 IDL constant definitions (PL/I) 194

G

generated member names 96
 GETUNIQUE 169
 global keyword
 COBOL 72
 PLI 193
 global objects 37

H

HTTP 59

I

IBM COBOL compiler
 container names 98
 fieldnames 105
 name scoping 97

string literal character limit 81
 IDL compiler 13
 -J argument 185
 -L argument 185
 -M argument 65, 186
 -M argument and FQN name 194
 -O argument and COBOL 111
 -O argument and PL/I 200
 -O argument and PODEXEC 215
 -S:-TCICS arguments (COBOL) 172
 -S and TIMS arguments (COBOL) 152, 172
 -S argument (PL/I) 225, 239
 -Z:-TCICS arguments 172
 IDL constant definitions
 COBOL 73
 PL/I 192
 IDL file, more than one interface in 201
 IDL filenames
 different from interface names 199
 include filename 196
 length 96
 IDL fixed type definitions 21
 IFNAME 215
 IFR 181
 IIOP
 and Orbix 262
 connection management 58
 IMS COBOL clients
 error checking 168
 extra copybooks 169
 linkage section 166
 IMSPCB module (PL/I) 228
 IMS PLI clients
 DLIDATA changes 234
 error checking 235
 program communication block 231
 INCLUDE.COPYLIB 137
 CHKERRS 125
 INCLUDE.COPYLIB(CICWRITE) 180
 INCLUDE.COPYLIB(IMSWRITE) 169
 INCLUDE.PLINCL(CORBA) 211
 include filenames, and IDL filename 196
 include statement 200, 201
 interface names
 and PL/I keywords 203
 COBOL keywords 117
 interfacename-TYPE (COBOL) 108
 interfacename_type (PLI) 195
 Interface Repository 181

INV_OBJREF exception 37
 IOCallback functionality 59
 IOR configuration 31
 IOR syntax 31
 itadmin tool 258
 itmfaloc 144

J

JCL, and the itadmin tool 259

L

license file 22
 load-balancing 45
 loader architecture 44
 local_log_stream configuration variable 248
 local name 185
 logging severity levels 248
 long IDL data type, ORBALLOC 149
 LSIMSPCB 160, 166

M

main() 37
 ManualWorkQueue policy 55
 maxConnectRetries() 57
 MEMALLOC (COBOL) 146
 MEMALLOC
 PL/I 213
 member names, length restriction 96
 MEMDEBUG 213
 MEMFREE 213
 memory management rules 138
 module keyword
 COBOL 73, 193
 module names
 and COBOL keywords 117
 and PL/I keywords 203
 modules, levels of 197
 multicast protocol 59
 multi-threaded clients 39
 multi-threading capabilities 53

N

Naming Service
 COBOL 142
 PL/I 208
 native exception handling 36
 no_user_exceptions 215

O

OBJ2STR (PL/I) 213
 OBJDUP 138
 Object/Servant Lifecycles 46
 object IDs 44
 ObjectId_to_string() 32
 object map (BOA) 44
 object names, resolving
 COBOL 142
 PL/I 208
 OBJECT_NOT_EXIST exception 37
 object references 48
 creating with POA 46
 fabricated 140
 OBJGET (COBOL) 146
 OBJGET (PLI) 213
 OBJGETI 146
 OBJGETM 146, 213
 OBJGETO 213
 OBJGTID 213
 OBJLEN 213
 OBJLENO 213
 OBJNEW 213
 OBJREL 138
 OBJSET 146, 213
 COBOL 140
 naming service 142
 PL/I 206
 OBJSETM 146, 213
 OMG
 mapping standard for unions and exceptions 120
 ORBALLOC 149
 ORB class 57
 ORB_CTRL_MODEL 53
 ORBEXEC, user exception parameter 147
 ORBFREE 146
 ORB_init() 37
 Orbix.bindUsingIOP(0) 262
 Orbix 6.x ORB class 57
 Orbix filters 52
 Orbix IDL compiler See IDL compiler
 Orbix loader architecture 44
 Orbix locator daemon 31
 Orbix object 37
 Orbix Protocol 262
 OrbixSecurity 53
 ORBIX-STATUS-INFORMATION 148
 ORBREGO 146
 ORB_shutdown(1) 37

ORBSTAT 148
 ORXADMIN PROC 259

P

PCBLIST 229
 piggybacking data 53
 PL/I Data Names, maximum length of 188
 PL/I keywords 203
 PL/I runtime 214
 POA
 activation modes 45
 AutomaticWorkQueue 55
 implementation 46
 multi-threading 53
 servers 48
 workqueue policies 55
 POA names 13
 POA policies 43
 callback objects 39
 overriding default 44
 PODALOC 213
 PODEBUG 213
 PODERR 213
 PODFREE 213
 PODHOST 213
 PODINIT 213
 PODRASS 213
 PODREG 213
 PODREGI 213
 PODRUN 213
 PODSTAT 214
 POD_STATUS_INFORMATION 214
 PortableInterceptor interfaces 52
 PortableServer 37
 program communication block (PL/I) 228
 proxy objects 37
 putidl 181
 and itadmin 258

R

Request::descriptor() 53
 Request::operator 41
 request logging 52
 request processing 37
 reserved COBOL keywords 117
 reserved PL/I keyword 203
 runtime reporting of exceptions 148

S

security features 53
 SEQALLOC 150
 sequence numbers 22
 servant implementation 25
 servant locators 44
 servants, object references 48
 server accessor (PLI) 216
 server names 13
 ServiceContexts 53
 shared memory transport protocol 59
 short IDL data type, ORBALLOC 149
 shutdown, ORB 37
 SINGLE_THREAD_MODEL 53
 SIOP 59
 SOAP 59
 STR2OBJ (PL/I) 206, 213
 Stringified IOR syntax 31
 string literal character limit 81
 string markers 44
 string-object
 (COBOL) 144
 PL/I 210
 string_to_ObjectId() 32
 STRSETSP 146
 synchronization concerns 44

T

TCP/IP information, access to 53, 59
 Temporary Storage labels 12
 ThreadFilters mechanism 53
 thread pools 54
 tie approach 48
 TRANSIENT exception 38
 transport protocols 262
 typecodes
 COBOL mapping 81
 PL/I mapping 195

U

UNIX, file descriptor limits and 59
 unsigned long IDL data type, ORBALLOC 149
 unsigned short IDL data type, ORBALLOC 149
 UPDTPCBS copybook 163
 URL syntax 31
 user exceptions 147
 and PODEXEC 215

V

variable block data sets 22
_var type 50

W

Working Storage labels 12
WorkQueue policies 55
WSCICSL 180
WSIMSCL 169

