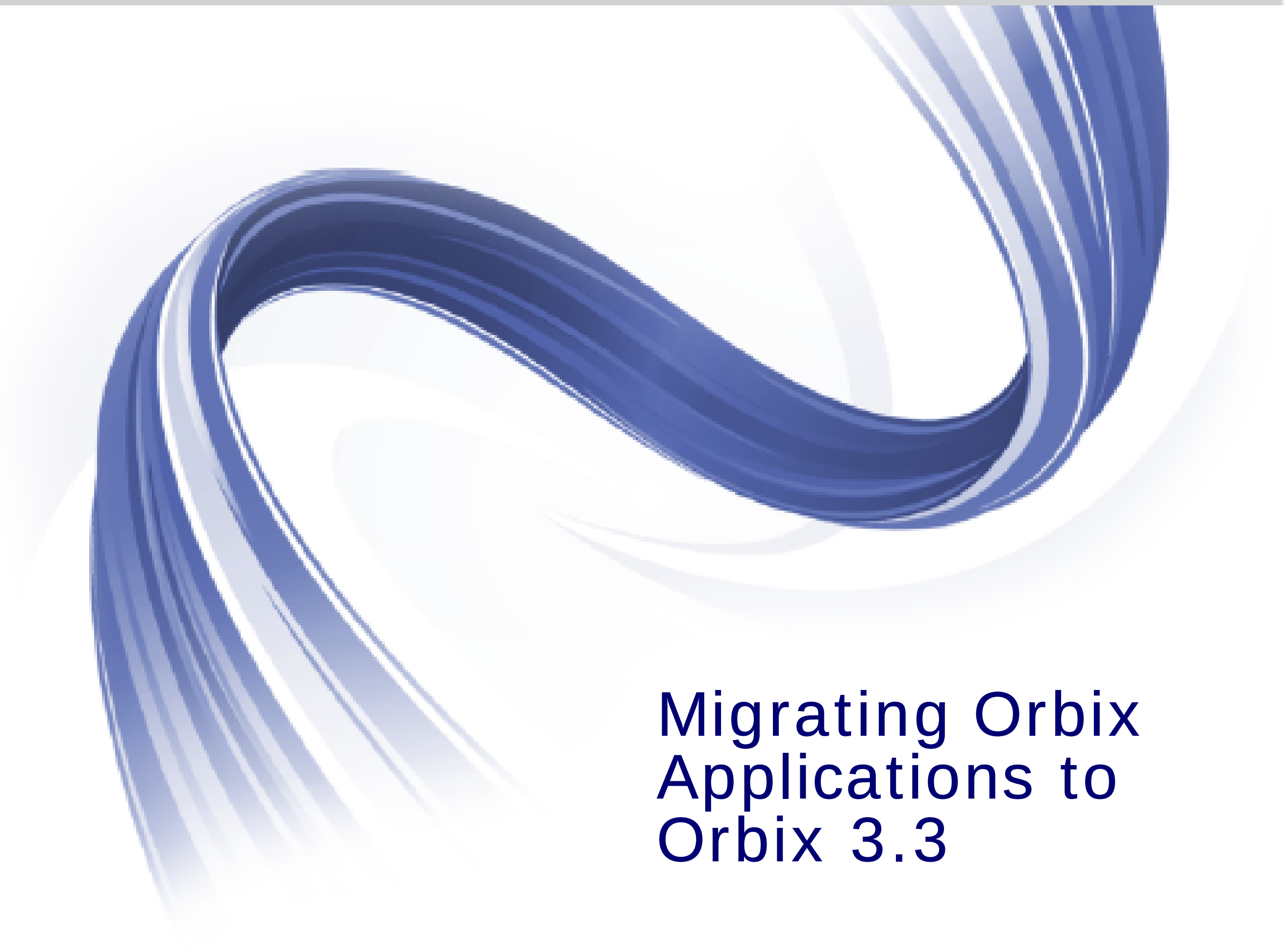
Orbix 3.3.13



Migrating Orbix Applications to Orbix 3.3

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK

http://www.microfocus.com

9/9/15

Contents

Preface

This document explains how to migrate applications from the discontinued OrbixWeb products, and earlier versions of Orbix 3, to Orbix 3.3.

Audience

This document is aimed at *C++ or Java programmers* who are already familiar with Micro Focus's Orbix or OrbixWeb product and who now want to migrate all or part of a system to use Orbix 3.3.

Parts of this document are relevant also to *administrators* familiar with Orbix (and the older OrbixWeb) administration.

Typographical Conventions

This guide uses the following typographical conventions:

`Constant width`	Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class.
	Constant width paragraphs represent code examples or information a system displays on the screen. For example:
	`#include <stdio.h>`
Italic	Italic words in normal text represent *emphasis* and *new terms*.
	Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:
	`% cd /users/`*your_name*
	Note: some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying Conventions

This guide may use the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
`%`	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
`#`	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
`>`	The notation > represents the Windows command prompt.
... . . .	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.
|	A vertical bar separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.

Contacting Micro Focus

Our Web site gives up-to-date details of contact numbers and addresses.

Further Information and Product Support

Additional technical information or advice is available from several sources.

The product support pages contain a considerable amount of additional information, such as:

- The WebSync service, where you can download fixes and documentation updates.
- The Knowledge Base, a large collection of product tips and workarounds.
- Examples and Utilities, including demos and additional product documentation.

To connect, enter http://www.microfocus.com in your browser to go to the Micro Focus home page.

Note:
Some information may be available only to customers who have maintenance agreements.

If you obtained this product directly from Micro Focus, contact us as described on the Micro Focus Web site, http://www.microfocus.com. If you obtained the product from another source, such as an authorized distributor, contact them for help first. If they are unable to help, contact us.

Information We Need

However you contact us, please try to include the information below, if you have it. The more information you can give, the better Micro Focus SupportLine can help you. But if you don't know all the answers, or you think some are irrelevant to your problem, please give whatever information you have.

- The name and version number of all products that you think might be causing a problem.
- Your computer make and model.
- Your operating system version number and details of any networking software you are using.
- The amount of memory in your computer.
- The relevant page reference or section in the documentation.
- Your serial number. To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Contact information

Our Web site gives up-to-date details of contact numbers and addresses.

Additional technical information or advice is available from several sources.

The product support pages contain considerable additional information, including the WebSync service, where you can download fixes and documentation updates. To connect, enter http://www.microfocus.com in your browser to go to the Micro Focus home page.

If you are a Micro Focus SupportLine customer, please see your SupportLine Handbook for contact information. You can download it from our Web site or order it in printed form from your sales representative. Support from Micro Focus may be available only to customers who have maintenance agreements.

You may want to check these URLs in particular:

- http://www.microfocus.com/products/corba/orbix/orbix-3.aspx (trial software download and Micro Focus Community files)
- https://supportline.microfocus.com/productdoc.aspx. (documentation updates and PDFs)

To subscribe to Micro Focus electronic newsletters, use the online form at:

http://www.microfocus.com/Resources/Newsletters/infocus/newsletter-subscription.asp

Introduction

Orbix 3.3 is the current version of Micro Focus' established Orbix 3 product.

This chapter discusses the following topics:

- Upgrading Orbix
- Migration Resources
- Migration Options

Upgrading Orbix

The recommended path for customers upgrading to a new version of Orbix is to move to Orbix 6. Because Orbix 6 is a CORBA 2.6-compliant ORB, it offers many new features. However, Orbix 3.3 is for those Orbix customers who have deployed applications in the field and who want to stay with a CORBA 2.1-based system.

Migration Resources

Micro Focus 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 guide.

  This technical document provides detailed guidance on converting source code to Orbix 3.3. The document aims to provide comprehensive coverage of migration issues, and to demonstrate how features supported in earlier Orbix versions can be mapped to Orbix 3.3 features.

Migration Options

The possible migration options are:

- Migrating to Orbix 3.3
- Mixed Deployment

Migrating to Orbix 3.3

Migrating to Orbix 3.3 is appropriate in some cases, where the effort of migration to Orbix 6 is not justified. For example, Orbix 3.3 might be an appropriate migration choice for CORBA applications that are approaching the end of their deployed lifespan. Micro Focus Customer Support are available to advise you on the most appropriate migration strategy for your system.

Orbix 3.3 is the current release of Micro Focus's CORBA 2.1-based ORB technology. The Orbix 3.3 product includes both a C++ ORB, formerly Orbix, and a Java ORB, formerly OrbixWeb, in a single package. A number of services are bundled with the Orbix 3.3 product, including, the CORBA Naming Service and the CORBA Security Service (OrbixSSL).

Note: A number of additional services, including the CORBA Events Service, a DCOM bridge (OrbixCOMet), and an IIOP firewall (Orbix Wonderwall) were available with earlier versions of Orbix 3.3 but are not supported by the latest release, Orbix 3.3 SP13.

If you choose the migration path to Orbix 3.3, chances are that you will need to deploy Orbix in a mixed Orbix 3.3 / Orbix 6 environment at some point in the future. Consequently, Orbix 3.3 has been optimized to achieve the best possible degree of on-the-wire interoperability with Orbix 6.

The main issue for migration to Orbix 3.3 (affecting both Orbix and OrbixWeb legacy code) is that `_bind()` calls must be modified to use the fully qualified form of `_bind()`. This is described in detail in the section "Modifications to _bind() / bind() in C++ and Java" in "Changes to APIs and Features".

Mixed Deployment

Mixed Deployment is appropriate when a number of CORBA applications are in deployment simultaneously. Some applications might be upgraded to use Orbix 6 whilst others continue to use Orbix 3.x and OrbixWeb 3.x. This kind of mixed environment requires on-the-wire compatibility between the generation 3 products and Orbix 6. Extensive testing has been done to ensure interoperability with Orbix 6.

On-the-Wire Interoperability

Both Orbix 3.3 and Orbix 6 have been modified to achieve an optimum level of on-the-wire compatibility between the two products. For more information on interoperability, see the *Interoperability* section of the ***Migrating from Orbix 3.3 to Orbix 6.3*** manual, in the Orbix 6 documentation set.

Migration to Orbix 3.3

This chapter discusses the issues involved in migrating from Orbix 3.0 to Orbix 3.3.

Issues for migrating from Orbix 3.0 to Orbix 3.3 can be broken down into the following categories:

1. The replacement of the Baltimore Security Toolkit in the Orbix 3.3 SP13 Java runtime. See "Java Security Toolkit" for details.
2. Other APIs and features in Orbix 3.0 that have been eliminated or affected by changes in Orbix 3.3. See "Changes to APIs and Features" for details.
3. Considerations for interoperating with other Orbix systems, especially in a heterogeneous environment involving a mixture of Orbix 3.0, 3.3, 6.*x*, and 2000 clients and servers. See see the *Interoperability* section of the ***Migrating from Orbix 3.3 to Orbix 6.3*** manual, in the Orbix 6 documentation set, for details.

Note: Most of the improvements made to Orbix 3.3 to improve interoperability with Orbix 2000 (item 2 above), were retrofitted into Orbix 3.0.1 at patch 20.

Many necessary changes are flagged as compile-time errors. It is relatively easy to find and correct these kinds of error and make the function calls conform to the new APIs. A few items have to be searched for manually, because they do not generate compile-time errors. For each of the changed items, the following sections indicate whether the compiler detects the API change or whether a manual search is required.

Java Security Toolkit

This chapter discusses changes resulting from the replacement of the Baltimore Security Toolkit in the Java runtime

The Baltimore security toolkit which was provided in earlier Orbix 3.3 releases is replaced with the JCA/JSSE Java Security Toolkit (JSSE), introduced in Orbix 3.3.13.

Security Protocols

By default Orbix 3.3.13 supports the following TLS protocols:

- TLS v1.0
- TLS v1.1
- TLS v1.2

Note: Due to various security vulnerabilities, the SSLv3 protocol is disabled by default in Orbix 3.3.13. Use of the SSL protocol should be avoided in favor of TLS. It is possible, though strongly not recommended, to enable the SSLv3 protocol to interoperate with endpoints that only support the SSLv3 protocol. This can be achieved by setting the `OrbixSSL.IT_PROTOCOLS` configuration variable.

Cipher Suites

Specify the cipher suites that you wish to use, by setting the `IT_CIPHERSUITES` and `IT_ALLOWED_CIPHERSUITES` configuration variables.

An example of setting the cipher suites in configuration is as follows:

```
IT_CIPHERSUITES =
    "RSA_WITH_AES_128_CBC_SHA,RSA_WITH_AES_256_CBC_SHA,RSA_
    WITH_AES_128_CBC_SHA256,RSA_WITH_AES_256_CBC_SHA256";
```

If you are using either of the following cipher suites, note that your JDK must have the JCE Unlimited Strength Jurisdiction Policy Files installed:

- `IT_SSLCipherSuite.IT_RSA_WITH_AES_256_CBC_SHA`
- `IT_SSLCipherSuite.IT_RSA_WITH_AES_256_CBC_SHA256`

Note: Some normally supported cipher suites cannot be used with specific versions of Java:

- When using the IBM JDK on AIX, the following cipher suites are not permitted with TLSv1, TLSv1.1, or TLSV1.2 when using the JCE unlimited strength jurisdiction policy files:
  - `SSL_RSA_WITH_RC4_128_MD5`
  - `SSL_RSA_WITH_RC4_128_SHA`
- Java 8 may not provide support for some of the cipher suites supported by Orbix 3.3. For example, Java 8 may fail to handshake when using:
  - `SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`
  - `SSL_RSA_EXPORT_WITH_RC4_40_MD5`

Known issues

- On HPUX 11iv3 (B.11.31) 64 bit, PKCS12 certificates generated by Netscape might not be readable by Orbix.
- Certificates that are signed with MD5 (MD5WithRSA) are not permitted under IBM Java versions 7 or 8. This is not an issue with other JCA implementations (Oracle and HP), nor with versions of the TLS protocols lower than TLS 1.2.

  Micro Focus highly recommends that any certificates used in secure Orbix applications that are signed with an MD5 digest signature are regenerated to use at least a SHA-1 digest signature.
- If a private key is in PEM encoding and contains data similar to the following, then Orbix Java is unable to read the key:

  ```
  Proc-Type: 4,ENCRYPTED
  DEK-Info: DES-EDE3-CBC,FE4CB4E10993F9D1
  ```

  Some workarounds are:
  - Convert the private key to DER encoding.
  - Convert the private key to PKC12 encoding.
  - If the KEY must be in PEM encoding, then generate the private key so it does not contain the data above. For example, using OpenSSL use:

    ```
    openssl genrsa -out private_key.pem 2048
    ```

  Note that Orbix Java does not support PKCS8 encoded private keys.
- The Baltimore toolkit provided a means for caching sessions. The equivalent functionality is not provided by JSSE.
- The certificate verification performed by the Baltimore toolkit differs from that provided by JSSE. The exceptions thrown by certificate verification may differ from those thrown when the Baltimore toolkit was used.

Impact on APIs

The change in security toolkits has an impact on the behavior of some of the APIs described in the ***OrbixSSL Programmer's and Administrator's Guide C++ Edition*** and the ***OrbixSSL Programmer's and Administrator's Guide Java Edition***.

Function	Description
`IE.Iona.OrbixWeb.SSL.IT_AVA.convert()`	The byte array returned to the caller is generated from the JSSE `X509Certificate` class. This array differs from the array returned when it was generated using a Baltimore class.
`IE.Iona.OrbixWeb.SSL.IT_X509Cert.getExtensions()`	When using JSSE, the number of extensions returned as well as the content of the extension data may differ from the Baltimore toolkit.

Function	Description
IE.Iona.OrbixWeb.SSL.IT_X509Cert.getIssuer()	The value in the returned IT_AVAList when using JSSE differs from the Baltimore toolkit. In particular, the DER value for each IT_AVA in the IT_AVA_LIST will be for the entire issuer.
IE.Iona.OrbixWeb.SSL.IT_X509Cert.getSubject()	The value in the returned IT_AVAList when using JSSE differs from the Baltimore toolkit. In particular, the DER value for each IT_AVA in the IT_AVA_LIST will be for the entire subject
IE.Iona.OrbixWeb.SSL.IT_X509Cert.getVersion()	When using the Baltimore toolkit, the version returned was the value in the certificate. For example, a version 1 certificate has a value of 0, so 0 is returned on the call. With JSSE, the actual version number is returned. For a version 1 certificate, the value 1 is returned.
IE.Iona.OrbixWeb.SSL.IT_X509Cert.parseExtensions()	When using JSSE, the number of extensions returned may differ from the Baltimore toolkit.
IE.Iona.OrbixWeb.SSL.IT_SSL.getNegotiatedCipherSuite()	If the following cipher suites have been set with a call to IE.Iona.OrbixWeb.SSL.IT_SSL.specifyCipherSuites(): • IT_SSLCipherSuite.IT_RSA_WITH_AES_128_CBC_SHA • IT_SSLCipherSuite.IT_RSA_WITH_AES_256_CBC_SHA • IT_SSLCipherSuite.IT_RSA_WITH_AES_128_CBC_SHA256 • IT_SSLCipherSuite.IT_RSA_WITH_AES_256_CBC_SHA256 The IT_SSLCipherSuite .name() value returned by getNegotiatedCipherSuite() will be: • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_RSA_WITH_AES_256_CBC_SHA • TLS_RSA_WITH_AES_128_CBC_SHA256 • TLS_RSA_WITH_AES_256_CBC_SHA256 However, when using an IBM JDK, the values will be: • SSL_RSA_WITH_AES_128_CBC_SHA • SSL_RSA_WITH_AES_256_CBC_SHA • SSL_RSA_WITH_AES_128_CBC_SHA256 • SSL_RSA_WITH_AES_256_CBC_SHA256

Changes to APIs and Features

Support for some long-deprecated functions and features has been dropped from Orbix 3.3. Other features have been changed to improve compatibility and interoperability with Orbix 2000.

Modifications to _bind() / bind() in C++ and Java

The _bind function is generated for each class representing an IDL interface. `_bind() / bind()` provides a primitive, non-CORBA-compliant, way to connect an initial client proxies with a server object. However, the preferred, CORBA-compliant approaches for establishing initial client proxies are to use `resolve_initial_references()`, the Naming Service, Trader Service, or an application-level factory/finder interface. The `_bind()` function has been deprecated for some time.

In Orbix 3.3, `_bind` has been significantly changed, but not altogether eliminated. `_bind` had various complex and esoteric forms, which have been eliminated. The most common and straightforward use of `_bind`, called *fully qualified* `_bind`, can still be used. For functionality beyond fully qualified `_bind`, you need to change code to use fully qualified `_bind` or one of the preferred alternative approaches.

The `_bind` function takes three arguments, and implies a fourth. For example:

```
CORBA::Object_var a = Account::_bind("M:S", "H");
```

supplies the following four pieces of data to the ORB runtime:

- `Account` is the name of the target object interface, or parent interface. The result can be stored in an `Account_var` or a base class pointer of `Account` (for example `Object_var`).
- `M` is the marker of the target object.
- `S` is the name of the CORBA server in which to look for the target object.
- `H` is the host name of the machine on which to look for the CORBA server.

Unsupported Forms of _bind() / bind()

In Orbix 3.0, some `_bind` arguments could be omitted, leading to various `_bind` modes. All of these partially qualified `_bind` modes have been eliminated.

- *Polymorphic bind*—The interface name could be any base interface of the target object.
- *Anonymous bind*—The marker could be omitted, implying that the ORB could choose any object that satisfied the other criteria.
- *Implied server bind*—The server name could be omitted, implying that the ORB would use the interface name as the server name (Account in this case).

- *Locator bind*—The host name could be omitted, implying that the ORB would use the Locator to examine the host or hostgroup files (or a user algorithm) to determine the host the server might be running on.

The above forms of `_bind` could even be combined, such as anonymous polymorphic `_bind`, or anonymous locator `_bind`, and so on.

Fully Qualified _bind() / bind()

Fully qualified `_bind` is the only mode supported in Orbix 3.3. It requires the following:

1. The interface name is exactly the most derived interface of the target object.
2. The marker is specified for the target object.
3. The server is specified for the target object.
4. The host is specified for the target object.

With fully qualified `_bind`, it is generally necessary to set the marker explicitly. Look for where objects are instantiated in the server, with either the BOA or TIE approach, and confirm that a marker string is supplied to the constructor or used in a call to `_marker()`.

It is important to understand what functionality occurs on the client and what functionality occurs on the server:

- The marker, server, and host parameters are verified on the client. If an Orbix 3.3 client calls `_bind` without a marker, server or host, `_bind` throws a `CORBA::BAD_PARAM` SystemException.
- The interface type is verified on the server. If the interface is not specified correctly, an Orbix 3.3 server throws a `CORBA::INV_OBJREF` SystemException.

These semantics have an impact on mixed environments, where Orbix 3.0 / OrbixWeb 3.2 (or earlier) interoperates with Orbix 3.3. When interoperating with an Orbix 3.3 server, an Orbix 3.0 / OrbixWeb 3.2 (or earlier) client:

1. Can use fully qualified `bind`, without modifying the client.
2. Can omit the host name and server name without modifying the client, because these parameters are resolved on the client.
3. Can omit the marker (anonymous bind) without modifying the client, provided the *`Orbix.ENABLE_ANON_BIND_SUPPORT`* environment variable is set to `TRUE` on the server (default is `TRUE`). Setting this environment variable to FALSE improves the Orbix 3.3 speed.

   The anonymous bind is the most common of the esoteric bind modes. This switchable backwards compatibility eases migration while allowing you, eventually, to take advantage of the Orbix 3.3 performance improvements related to fully qualified `_bind`.
4. Cannot use polymorphic bind. The Orbix 3.3 server would return a `CORBA::INV_OBJREF` system exception in this case. This case requires the client to be modified.

If `CORBA::ORB::collocated` is set to TRUE, the fully qualified bind requirements are reduced to specifying only the marker and exact interface. Because an inprocess object lookup is going to be performed, the host is ignored and the server name can be omitted. Alternatively, if the server name is present, it must be the server name associated with the current server process.

C++ Function Signatures for _bind()

The IDL compiler generates a set of overloaded _bind functions to handle the various forms of _bind. Some of these have changed because they are no longer needed:

- `_bind(const char* markerServer, const char* host, const CORBA::Context& ctx)` remains unchanged.
- `_bind()` is removed.
- `_bind(const char* markerServer = 0, const char* host = 0)` is changed to the following (no longer has default args): `_bind(const char* markerServer, const char* host)`

When explicitly binding to the Orbix daemon, orbixd, use a 0 (zero) marker value:

```
IT_daemon::_bind("0:IT_daemon", host);
```

When explicitly binding to the IFR, use a marker of the IDL type for the repository object (all IFR object markers are IFR type names):

```
Repository::_bind("IDL\\:iona.com/Repository:IFR", host);
```

Java Method Signatures for bind()

The idlj compiler also generates a set of overloaded bind() methods to handle the various bind forms. The following have been removed and the remaining bind() calls remain the same.

- `bind ()` is removed.
- `bind(org.omg.CORBA.ORB orb)` is removed.

Locator

The Orbix locator is a non-CORBA-compliant feature that resolves the host name in `_bind` when no host name is explicitly provided. This functionality is no longer needed with fully qualified _bind. The Orbix Locator is actually a set of features, which have changed as described in the following sections.

The CORBA::LocatorClass

The CORBA::LocatorClass has been removed because it is not needed, now that bind is fully qualified. If you have legacy code that uses a CORBA::LocatorClass to provide host resolution logic, you should move that logic to an independent class, and invoke the behavior to resolve the host name before calling _bind. Alternatively, you can use configuration variables to specify the host or, preferably, the IOR, of well-known server objects:

```
IT_<ServiceName>_HOST = ". . .";
Common.Services.ServiceName = "IOR: . . .";
```

These objects are accessed through the CORBA-compliant `CORBA::ORB::resolve_initial_references() function`, instead of `_bind`.

The Locator Files and Associated Utilities

The files associated with the locator, `Orbix.hst` and `Orbix.grp`, are of limited usefulness in Orbix 3.3 because they are no longer used by `_bind()`. However, these files are still accessible through operations defined on the *`IT_daemon`* IDL interface—for example `lookUp()`, `addHostsToServer()`, `addHostsToGroup()` and so on.

The utilities that edit the locator files, `serverhosts`, `servergroups`, `grouphosts`, `lhosts`, are no longer provided with Orbix 3.3. The `Orbix.hst` and `Orbix.grp` files can be edited using a regular text editor instead.

It is no longer meaningful to have an `IT_daemon` entry in the locator files.

Non-Native C++ Exceptions

Only native C++ exceptions are supported. This means that the `TRY/CATCH` macros are no longer supported and exceptions are not raised via the `CORBA::Environment` variable argument. However, the `CORBA::Environment` variable can still be used as the mechanism to pass a per-call timeout value, if such functionality is needed.

You should search your client and server source code for `TRY/CATCH` macros and convert it to use C++ try/catch. Code with `TRY/CATCH` macros will no longer compile. The following example shows a code fragment before and after being migrated to use `TRY/CATCH` .

Example 1

Consider the following original code, which uses the `TRY/CATCH` macros to handle an exception raised by an `Account::withdraw()` operation:

Original Code

```
Account_var a = . . .;
TRY
{
    a->withdraw(100.00, IT_X);
}
CATCH(Bank::InsufficientFunds, e)
{
    cout << "insufficient funds" << endl;
}
ENDTRY
```

The `TRY/CATCH` macros declare and use a variable named `IT_X`, which is used to propagate exception information.

Compare the original code with the following revised code, which has been modified to use the native C++ try/catch:

Revised Code

```
Account_var a = . . .;
try
{
    a->withdraw(100.00);
}
catch (const Bank::InsufficientFunds& e)
{
    cout << "insufficient funds" << endl;
}
```

Example 2

Another possibility is that the client exception-handling code is written directly, using the `CORBA::Environment` variable without the `TRY/CATCH` macros. This typically means the exception handling logic is an if-block, testing the `CORBA::Environment` variable. For example:

Original Code

```
Account_var a = . . .;
CORBA::Environment e;
a->withdraw(100.00, e);
if (e.is_exception("Bank::InsufficientFunds"))
{
    cout << "insufficient funds" << endl;
}
```

Compare this with the following revised code, which has been modified to use the native C++ try/catch:

Revised Code

```
Account_var a = . . .;
try
{
    a->withdraw(100.00);
}
catch (Bank::InsufficientFunds& e)
{
    cout << "insufficient funds" << endl;
}
```

The second example (using `CORBA::Environment`) is not as easy to search for as the first example (using `TRY/CATCH` macros), because there are no `TRY/CATCH` macros to search for. It is best also to search for explicit usages of `CORBA::Environment` and `is_exception`.

Note: The second example still compiles in its original form. It is valid to declare and use `CORBA::Environment`, but it cannot be used for exceptions.

The impact of not changing the code in the second example can be severe. If the call to `withdraw()` raises an exception, in the original code the C++ runtime looks for the nearest enclosing try/catch block and does not consider the subsequent if statement.

Throwing Exceptions

Throwing an exception within a server is not done by setting the `CORBA::Environment` variable, but using a C++ throw.

Exception Handling in Filters

The `CORBA::Environment` variable can still be used to test and set exceptions within filters, as in Orbix 3.0.

CORBA::ORB::useNativeExceptions

Because only native C++ exceptions are supported, the following functions have been removed:

```
CORBA::Boolean CORBA::ORB::nativeExceptions()
CORBA::Boolean CORBA::ORB::nativeExceptions(Boolean)
```

Code that formerly depended on these functions can assume that `nativeExceptions()` always returns `TRUE`. For example:

Original Code

```
if (CORBA::Orbix.nativeExceptions())
{
    ... //code for native exceptions being TRUE
}
else
{
    ... //code for native exceptions being FALSE
}
```

Revised Code

```
... //code for native exceptions being TRUE
```

Class CORBA::NatExcResetter

Because only native C++ exceptions are supported, the `CORBA::NatExcResetter()` class has been removed, as it is no longer meaningful. Code that uses the `CORBA::NatExcResetter()` class should be deleted.

CORBA::Object Class

The following functions have been removed from `CORBA::Object`, as they are no longer meaningful. `CORBA::Object` is the base class of all generated classes for IDL interfaces. The following were never documented APIs, so, in general, should not have been used by Orbix developers. Any code that uses the following functions, should be deleted.

- `void CORBA::Object::_restate()`
- `void CORBA::Object::_marshall()`
- `void CORBA::Object::_unmarshall()`
- `void CORBA::Object::_fixOnAccess()`
- `CORBA::PPTR* CORBA::Object:_makeDummyPptr()`
- `Enumeration CORBA::Object::OBJECT_STATE`
- `CORBA::Object::Object(const Object*)`

Note: `CORBA::Object::Object(const CORBA::Object&)` is still available. However, `CORBA::Object::operator=(const CORBA::Object&)` is not available.

Thread Model

The internal thread model has been updated in Orbix 3.3. This has no direct impact on application-level threads, but has some implications for Orbix configuration.

- New Internal Thread Model—replaces the old internal thread model for monitoring the network.
- New Thread API—controls the number of threads and file descriptors (FDs) used for network connections.
- Functions Dropped from the Old Thread API—functions associated with the old internal thread model are no longer supported.
- New IOCallback Functions—warn when a process is running low on FDs or has reached a hard FD limit (all FDs used).
- Lock Model—The `mt.h` and ThreadArch.cxx source files are no longer supplied with Orbix 3.3.

New Internal Thread Model

The internal thread model for Orbix has been re-designed. This has no effect on the application level thread model that the user interacts with via the `CORBA::ThreadFilter` class. All `ThreadFilter` models, such as per-request, per-object, per-client, and so on, are still usable. The internal thread model is used by Orbix to listen for network connections from clients and to read and write network messages on established connections. One visible advantage of the new thread model is that it is easier for you to configure.

When you run an Orbix application, Orbix starts a number of internal threads in a thread pool. These threads work together to listen for incoming connection attempts from clients and read requests from the network. Ultimately, requests are processed by an application thread, using a thread model written by the user.

The internal network threads use a leader-follower design. This means that one thread in the pool is blocked on a call to the low-level TCP/IP `poll()`, and when activity occurs, this thread processes it. Simultaneously another thread is dispatched from the pool to perform another low-level TCP/IP `poll()`. When a thread completes its current task, it is returned to the pool.

New Thread API

The size of the internal network thread pool is controlled by the `IT_DEF_NUM_NW_THREADS` configuration parameter. The default value is `1`. The user can change this default if a larger initial internal network thread pool is needed.

The following new function can be used to control the number of threads in the internal thread pool:

```
CORBA::Boolean CORBA::ORB::add_nw_threads(
    CORBA::ULong num_threads
)
```

The `add_nw_threads()` function can be used to increase the number of threads in the internal network thread pool at any time. The `num_threads` parameter specifies the number of threads to add to the thread pool—the size of the thread pool can only be increased, not reduced.

The default thread pool size, `1`, is the best setting for most applications. A network thread is responsible for only a little bit of work, which consists of reading the TCP/IP buffer and depositing the message on an event queue for processing by an application thread.

Note: The network thread is also responsible for re-combining any IIOP-fragment messages (that is, what the network thread hands off is a complete IIOP message). However, IIOP fragments are rarely used—in particular, in Orbix 3, they are never generated (Orbix 3 has the ability to process IIOP fragments but not generate them). It is also important to note that unmarshalling the IIOP message occurs in the application thread, after the hand off from the network thread. So network threads do very little work. The cost of additional network thread on a single-processor machine is a context switch (which is relatively expensive); on multi-processor machines, the network threads could be distributed, across the processors. Of course, with any system, increasing the number of threads is not a guaranteed increase in performance, and depends on hardware and operating system.

In general terms, the number of network threads should only be increased if both of the following conditions hold:

1. There are lots of simultaneous requests/replies to a process.
2. A single network thread has insufficient capacity to service the TCP/IP buffers.

Functions Dropped from the Old Thread API

The following API functions associated with the old thread model have been removed:

- `void CORBA::ORB::maxConnectionThreads(CORBA::ULong max)`
- `CORBA::ULong CORBA::ORB::maxConnectionThreads() const`
- `void CORBA::ORB::maxFDsPerConnectionThread(CORBA::ULong max)`
- `CORBA::ULong CORBA::ORB::maxFDsPerConnectionThread() const`

New IOCallback Functions

As Orbix opens and closes connections, it consumes file descriptors. File descriptions (FDs) are process-level resources, and are also used for non-Orbix activities, such as file I/O, database access, and so on. The number of FDs in a process is a limited resource, and in general Orbix cannot assume that all available FDs can be used by Orbix (for example, some may need to be reserved for database activity).

Orbix allows a client or server to receive a callback for certain connection and file descriptor (FD) events. Callbacks exist for opening and closing a connection to another Orbix program. For the new thread model, additional callbacks have been developed to allow the user to monitor the consumption of FDs. The user can specify both soft and hard limits on the number of FDs Orbix can use.

To receive the new callbacks, define a class that inherits from the Orbix `CORBA::IT_IOCallback` class. The `CORBA::IT_IOCallback` class has been extended with three new callback events that allow the user to monitor the consumption of FDs:

```
// C++
class IT_IOCallback
{
    public:
        ...
        // The following functions are called when the number
        // of FDs used by Orbix hits a soft or hard limit set
        // by the user.
        // The low-watermark (soft limit) has been reached
        virtual void AtOrbixFDLowLimit(int numFDsUsed);
        // The hard limit has been reached.
        // This implies that Orbix is no longer listening for
        // new connections (which would consume another FD).
        virtual void StopListeningAtFDHigh(int numFDsUsed);
        // Orbix has resumed listening after the number of FDs
        // has gone below the hard limit.
        virtual void ResumeListeningBelowFDHigh(
        int numFDsUsed
        );
};
```

The `AtOrbixFDLowLimit()`, `StopListeningAtFDHigh()`, and `ResumeListeningBelowFDHigh()` functions, combined with new configuration variables `IT_FD_WARNING_NUMBER` and `IT_FD_STOP_LISTENING_POINT`, give users flexibility to monitor consumption of FDs:

- When the number of Orbix FDs reaches `IT_FD_WARNING_NUMBER`, either on the way up or the way down, `AtOrbixFDLowLimit()` is called.
- When the number of Orbix FDs reaches `IT_FD_STOP_LISTENING_POINT`, `StopListeningAtFDHigh()` is called.
- When an Orbix FD is freed up or the number of FDs made available to Orbix is increased, `ResumeListeningBelowFDHigh()` is called.

Lock Model

The internal lock model has been changed to use the Orbix 2000 lock classes. The `mt.h` and `ThreadArch.cxx` files are no longer supplied with Orbix 3.3. Legacy code that uses the classes in mt.h must use the previous versions of these files (and consider it application code, not Orbix code), or change the code to use a different mechanism.

CORBA::ORB::defaultTxTimeout

The single `CORBA::ORB::defaultTxTimeout()` function has been replaced by two functions. Originally, the function signature was:

```
// C++
// Original 'defaultTxTimeout()' signature
CORBA::ULong
CORBA::ORB::defaultTxTimeout(
    CORBA::ULong val = CORBA::INIFINITE_TIMEOUT,
    CORBA::Environment& env = CORBA::IT_chooseDefaultEnv
);
```

This has been replaced by two functions, one that is an accessor and one that is a mutator:

```
// C++
// Accessor function
CORBA::Ulong
CORBA::ORB::defaultTxTimeout();

// Mutator function
CORBA::ULong
CORBA::ORB::defaultTxTimeout(
    CORBA::ULong val,
    CORBA::Environment& env = CORBA::IT_chooseDefaultEnv
);
```

The original accessor-like functionality would also mutate the timeout to C`ORBA::INFINITE_TIMEOUT`, for example:

```
CORBA::ULong t = CORBA::Orbix.defaultTxTimeout();
```

Accessing the value and changing it are now clearly separated. It is unlikely that this change affects any client code, but you should verify this by searching for all calls to `defaultTxTimeout()`.

CORBA::Environment Class

Changes have been made to the `CORBA::Environment` class that affect some data members and member functions.

Accessing Data Members

Data members of the `CORBA::Environment` class that used to be public have been made private. The data members are now accessed using accessor/mutator function pairs. For example, the `m_request` data member:

```
// C++
CORBA::Request* CORBA::Environment::m_request
```

is now accessed using the following functions:

```
// C++
CORBA::Request* CORBA::Environment::request();
void CORBA::Environment::request(CORBA::Request*);
```

The `m_timeout` data member:

```
// C++
CORBA::ULong CORBA::Environment::m_timeout
```

is now accessed using the following functions:

```
// C++
CORBA::ULong CORBA::Environment::timeout() const;
void CORBA::Environment::timeout(CORBA::ULong val);
```

Attempting to access the `m_request` or `m_timeout` member variables directly generates compiler errors. Your code should be changed to use the accessor/mutator functions instead.

Member Functions Removed

Three `CORBA::Environment` functions, which were only needed to support the TRY/CATCH macros, have been removed:

```
// C++
void CORBA::Environment::propagate()
void CORBA::Environment::acknowledge()
CORBA::Boolean CORBA::Environment::uncaught()
```

The following function has been removed:

```
// C++
void Request::mk_arg(CORBA::TypeCode_ptr, void*)
```

It was supplied only on NT, and was redundant. This should not affect your code.

CORBA::CollocateResetter Class

The default `CORBA::Environment` parameter in the `CollocateResetter` constructor has been removed. The function signature is now:

```
// C++
CORBA::CollocateResetter::CollocateResetter(Boolean tmpSetting)
```

This is unlikely to affect your code. Any occurrences will be flagged as compiler errors, which can be easily fixed by removing the `CORBA::Environment` argument passed to the constructor.

Fixed Data Type

Orbix 3.0 and 3.3 support the IDL fixed data type. This data type maps to a C++ class. The function signatures for the Orbix 3.0 fixed data type class conform to the original OMG specification, but it turns out there were errors in the specification. Orbix 3.3 corrects these errors by changing the function signatures.

The changes primarily concern the use of references in return types. For example, the original specification uses:

```
// C++
template<unsigned short d, short s>
class CORBA_Fixed<d, s>
{
    public:
    template<unsigned short d, short s>
    CORBA_Fixed<d, s> operator= (const CORBA_Fixed<d, s>& val);
};
```

which defines `operator=()` with the wrong return type. A basic assignment would work, but complex (rarely coded) expressions would potentially fail. For example, consider the following assignment statement:

```
// C++
CORBA_Fixed<d, s> x = 0;
CORBA_Fixed<d, s> y = 0;

(x = y)++; //expect x equal to 1, y equal to 0;
           //in reality x would be 0, and y would be 0.
```

The expression fails, because the assignment return value is a new (temporary) instance of `CORBA_Fixed<d, s>`, instead of a `CORBA_Fixed<d, s>&` reference to the left-hand side, `x`, of the expression.

The `operator=()` assignment operator should have the following signature:

```
// C++
template<unsigned short d, short s>
CORBA_Fixed<d, s>& operator=(const CORBA_Fixed<d, s>& val
);
```

The implementation of the fixed data type class now throws a `CORBA::DATA_CONVERSION` system exception whenever the attempted operation would exceed the bounds described by the IDL fixed data type.

Processing CORBA.h

The CORBA.h header aggregates the various CORBA header files. The included class declarations have been segmented into more files, resulting in a greater number of files, although the total number of declarations in those files has decreased. The increased segmentation should help you to locate specific header files more easily (for example, when confirming an API signature).

Access to the runtime API is provided by a single #include <CORBA.h> line, as before—no changes are needed as a result of this reorganization.

In Orbix 3.0, users have to #define EXCEPTIONS to use the full range of CORBA system exceptions. This is no longer necessary in Orbix 3.3. Continuing to #define EXCEPTIONS does no harm (code still compiles and runs), but it is superfluous.

In Orbix 3.0, users have to #define WANT_ORBIX_FDS to use the full range of APIs for Orbix internal file descriptors. This is no longer necessary in Orbix 3.3.

Continuing to #define WANT_ORBIX_FDS does no harm (code still compiles and runs), but it is superfluous.

Some operating system header files conflict with CORBA.h. This problem occurred in Orbix 3.0 as well, but in Orbix 3.3 the user has more control over the mechanism for resolving the conflict. For example, some operating systems headers have a line, #define minor, in them. But minor is used as a function name on the exception class. Since the preprocessor makes the macro substitution first, this creates an error in the code. Orbix 3.0 would #undef the conflicting macros. This is still done in Orbix 3.3. However, all of the #undefs have been grouped together in CORBA.h, and are controlled by an ORBIX_DONT_UNDEF macro, for example:

```
// In CORBA.h
#ifndef ORBIX_DONT_UNDEF
#undef minor
. . .
#endif
```

Nothing needs to be done if you want the standard symbols to be #undef'ed as they always have been. However, if you want more control over this (for example, to #undef the symbols yourself, and then re-#define them later) you can #define ORBIX_DONT_UNDEF prior to #include <CORBA.h>.

DEF_TIE and TIE macros

The original versions of the `DEF_TIE` and `TIE` macros were superseded by new versions in Orbix 2.0. The original macros have been removed from Orbix 3.3. Legacy code using the original macros should be modified to use the newer macros. This requires a straightforward search-and-replace.

For an IDL interface, `Account`, and an implementation class, `Account_i`, the original `DEF_TIE` and `TIE` macros were of the following form, taking both the interface name and implementation class name as parameters:

```
DEF_TIE(Account, Account_i)
TIE(Account, Account_i)
```

The newer `DEF_TIE` and `TIE` macros (introduced in Orbix 2.0) use the interface name as part of the macro name, and only have the implementation class name as a parameter:

```
DEF_TIE_Account(Account_i)
TIE_Account(Account_i)
```

Your code should be searched for `TIE`, and any occurrences of the old macros changed to the newer form. The old macros generate a compile-time error in Orbix 3.3.

Index